

**Oracle® Database**

PL/SQL Packages and Types Reference

11g Release 2 (11.2)

**E25788-04**

December 2011

Oracle Database PL/SQL Packages and Types Reference, 11g Release 2 (11.2)

E25788-04

Copyright © 1996, 2011, Oracle and/or its affiliates. All rights reserved.

Primary Author: Denis Raphaely

Contributing Author: Lance Ashdown, Drue Baker, Donna Carver, Beethoven Cheng, Rhonda Day, Steve Fogel, Bryn Llewellyn, Paul Lane, Tony Morales, Chuck Murray, Sue Pelski, Kathy Rich, Antonio Romero, Vivian Schupmann, Cathy Shea, Margaret Taft, Kathy Taylor, Randy Urbano, Rodney Ward

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

---

---

## Contents

<b>Preface</b> .....	ci
Audience.....	ci
Documentation Accessibility .....	ci
Related Documents .....	ci
Conventions .....	cii
<b>What's New in PL/SQL Packages and Types Reference?</b> .....	cv
New Packages .....	cv
Updated Packages .....	cvii
New Types.....	cviii
Updated Types.....	cviii
<b>1 Introduction to Oracle Supplied PL/SQL Packages &amp; Types</b>	
<b>Package Overview</b> .....	1-2
Package Components .....	1-3
Using Oracle Supplied Packages .....	1-4
Creating New Packages .....	1-5
Referencing Package Contents .....	1-8
<b>Summary of Oracle Supplied PL/SQL Packages and Types</b> .....	1-9
<b>2 APEX_CUSTOM_AUTH</b>	
Documentation of APEX_CUSTOM_AUTH .....	2-2
<b>3 APEX_APPLICATION</b>	
Documentation of APEX_APPLICATION .....	3-2
<b>4 APEX_ITEM</b>	
Documentation of APEX_ITEM .....	4-2
<b>5 APEX_UTIL</b>	
Documentation of APEX_UTIL .....	5-2

<b>6</b>	<b>CTX_ADM</b>	
	Documentation of CTX_ADM.....	6-2
<b>7</b>	<b>CTX_CLS</b>	
	Documentation of CTX_CLS.....	7-2
<b>8</b>	<b>CTX_DDL</b>	
	Documentation of CTX_DDL .....	8-2
<b>9</b>	<b>CTX_DOC</b>	
	Documentation of CTX_DOC.....	9-2
<b>10</b>	<b>CTX_ENTITY</b>	
	Documentation of CTX_ENTITY.....	10-2
<b>11</b>	<b>CTX_OUTPUT</b>	
	Documentation of CTX_OUTPUT.....	11-2
<b>12</b>	<b>CTX_QUERY</b>	
	Documentation of CTX_QUERY .....	12-2
<b>13</b>	<b>CTX_REPORT</b>	
	Documentation of CTX_REPORT .....	13-2
<b>14</b>	<b>CTX_THES</b>	
	Documentation of CTX_THES.....	14-2
<b>15</b>	<b>CTX_ULEXER</b>	
	Documentation of CTX_ULEXER .....	15-2
<b>16</b>	<b>DBMS_ADDM</b>	
	Using DBMS_ADDM.....	16-2
	Security Model.....	16-3
	Summary of DBMS_ADDM Subprograms .....	16-4
	ANALYZE_DB Procedure .....	16-5
	ANALYZE_INST Procedure .....	16-6
	ANALYZE_PARTIAL Procedure .....	16-7
	DELETE Procedure .....	16-8
	DELETE_FINDING_DIRECTIVE Procedure.....	16-9
	DELETE_PARAMETER_DIRECTIVE Procedure .....	16-10
	DELETE_SEGMENT_DIRECTIVE Procedure .....	16-11
	DELETE_SQL_DIRECTIVE Procedure .....	16-12
	GET_ASH_QUERY Function .....	16-13

GET_REPORT Function .....	16-14
INSERT_FINDING_DIRECTIVE Procedure .....	16-15
INSERT_PARAMETER_DIRECTIVE Procedure .....	16-16
INSERT_SEGMENT_DIRECTIVE Procedure .....	16-17
INSERT_SQL_DIRECTIVE Procedure .....	16-19
<b>17 DBMS_ADVANCED_REWRITE</b>	
Using DBMS_ADVANCED_REWRITE.....	17-2
Security Model.....	17-3
<b>Summary of DBMS_ADVANCED_REWRITE Subprograms</b> .....	17-4
ALTER_REWRITE_EQUIVALENCE Procedure.....	17-5
BUILD_SAFE_REWRITE_EQUIVALENCE Procedure.....	17-6
DECLARE_REWRITE_EQUIVALENCE Procedures .....	17-7
DROP_REWRITE_EQUIVALENCE Procedure.....	17-9
VALIDATE_REWRITE_EQUIVALENCE Procedure .....	17-10
<b>18 DBMS_ADVISOR</b>	
Using DBMS_ADVISOR.....	18-2
Deprecated Subprograms .....	18-3
Security Model.....	18-4
<b>Summary of DBMS_ADVISOR Subprograms</b> .....	18-5
ADD_SQLWKLD_REF Procedure.....	18-8
ADD_SQLWKLD_STATEMENT Procedure .....	18-9
ADD_STS_REF Procedure .....	18-11
CANCEL_TASK Procedure.....	18-12
COPY_SQLWKLD_TO_STS Procedure .....	18-13
CREATE_FILE Procedure .....	18-14
CREATE_OBJECT Procedure.....	18-15
CREATE_SQLWKLD Procedure .....	18-17
CREATE_TASK Procedures .....	18-18
DELETE_SQLWKLD Procedure .....	18-20
DELETE_SQLWKLD_REF Procedure.....	18-21
DELETE_SQLWKLD_STATEMENT Procedures.....	18-22
DELETE_STS_REF Procedure .....	18-23
DELETE_TASK Procedure.....	18-24
EXECUTE_TASK Procedure .....	18-25
GET_REC_ATTRIBUTES Procedure.....	18-27
GET_TASK_REPORT Function.....	18-28
GET_TASK_SCRIPT Function.....	18-29
IMPLEMENT_TASK Procedure .....	18-31
IMPORT_SQLWKLD_SCHEMA Procedure.....	18-32
IMPORT_SQLWKLD_SQLCACHE Procedure .....	18-34
IMPORT_SQLWKLD_STS Procedure.....	18-36
IMPORT_SQLWKLD_SUMADV Procedure .....	18-38
IMPORT_SQLWKLD_USER Procedure .....	18-40
INTERRUPT_TASK Procedure .....	18-42
MARK_RECOMMENDATION Procedure .....	18-43

QUICK_TUNE Procedure.....	18-44
RESET_SQLWKLD Procedure .....	18-45
RESET_TASK Procedure.....	18-46
SET_DEFAULT_SQLWKLD_PARAMETER Procedures.....	18-47
SET_DEFAULT_TASK_PARAMETER Procedures .....	18-48
SET_SQLWKLD_PARAMETER Procedures.....	18-49
SET_TASK_PARAMETER Procedure.....	18-50
TUNE_MVIEW Procedure.....	18-63
UPDATE_OBJECT Procedure .....	18-65
UPDATE_REC_ATTRIBUTES Procedure .....	18-67
UPDATE_SQLWKLD_ATTRIBUTES Procedure .....	18-69
UPDATE_SQLWKLD_STATEMENT Procedure .....	18-70
UPDATE_TASK_ATTRIBUTES Procedure.....	18-72

## 19 DBMS\_ALERT

<b>Using DBMS_ALERT</b> .....	19-2
Overview .....	19-3
Security Model.....	19-4
Constants .....	19-5
Restrictions .....	19-6
Exceptions .....	19-7
Operational Notes .....	19-8
Examples .....	19-10
<b>Summary of DBMS_ALERT Subprograms</b> .....	19-11
REGISTER Procedure .....	19-12
REMOVE Procedure .....	19-13
REMOVEALL Procedure .....	19-14
SET_DEFAULTS Procedure .....	19-15
SIGNAL Procedure .....	19-16
WAITANY Procedure .....	19-17
WAITONE Procedure .....	19-18

## 20 DBMS\_APPLICATION\_INFO

<b>Using DBMS_APPLICATION_INFO</b> .....	20-2
Overview .....	20-3
Security Model.....	20-4
Operational Notes .....	20-5
<b>Summary of DBMS_APPLICATION_INFO Subprograms</b> .....	20-6
READ_CLIENT_INFO Procedure .....	20-7
READ_MODULE Procedure .....	20-8
SET_ACTION Procedure .....	20-9
SET_CLIENT_INFO Procedure.....	20-10
SET_MODULE Procedure .....	20-11
SET_SESSION_LONGOPS Procedure .....	20-12

---

## 21 DBMS\_APPLY\_ADM

Using DBMS_APPLY_ADM .....	21-2
Overview .....	21-3
Security Model.....	21-4
Operational Notes .....	21-5
<b>Summary of DBMS_APPLY_ADM Subprograms .....</b>	<b>21-6</b>
ADD_STMT_HANDLER Procedure .....	21-8
ALTER_APPLY Procedure .....	21-10
COMPARE_OLD_VALUES Procedure .....	21-16
CREATE_APPLY Procedure .....	21-18
CREATE_OBJECT_DEPENDENCY Procedure.....	21-24
DELETE_ALL_ERRORS Procedure .....	21-25
DELETE_ERROR Procedure.....	21-26
DROP_APPLY Procedure .....	21-27
DROP_OBJECT_DEPENDENCY Procedure .....	21-29
EXECUTE_ALL_ERRORS Procedure .....	21-30
EXECUTE_ERROR Procedure .....	21-31
GET_ERROR_MESSAGE Function .....	21-34
REMOVE_STMT_HANDLER .....	21-36
SET_CHANGE_HANDLER Procedure .....	21-38
SET_DML_HANDLER Procedure.....	21-41
SET_ENQUEUE_DESTINATION Procedure .....	21-46
SET_EXECUTE Procedure .....	21-48
SET_GLOBAL_INSTANTIATION_SCN Procedure.....	21-50
SET_KEY_COLUMNS Procedures .....	21-53
SET_PARAMETER Procedure .....	21-56
SET_SCHEMA_INSTANTIATION_SCN Procedure.....	21-66
SET_TABLE_INSTANTIATION_SCN Procedure .....	21-69
SET_UPDATE_CONFLICT_HANDLER Procedure.....	21-71
SET_VALUE_DEPENDENCY Procedure .....	21-75
START_APPLY Procedure.....	21-77
STOP_APPLY Procedure .....	21-78

## 22 DBMS\_AQ

Using DBMS_AQ .....	22-2
Constants .....	22-3
Data Structures .....	22-5
Operational Notes .....	22-8
<b>Summary of DBMS_AQ Subprograms.....</b>	<b>22-9</b>
BIND_AGENT Procedure .....	22-10
DEQUEUE Procedure.....	22-11
DEQUEUE_ARRAY Function.....	22-14
ENQUEUE Procedure .....	22-16
ENQUEUE_ARRAY Function.....	22-18
LISTEN Procedures.....	22-19
POST Procedure .....	22-21
REGISTER Procedure .....	22-22

UNBIND_AGENT Procedure .....	22-23
UNREGISTER Procedure .....	22-24

## 23 DBMS\_AQADM

<b>Using DBMS_AQADM</b> .....	23-2
Constants .....	23-3
<b>Subprogram Groups</b> .....	23-4
Queue Table Subprograms .....	23-5
Privilege Subprograms .....	23-6
Queue Subprograms .....	23-7
Subscriber Subprograms .....	23-8
Notification Subprograms .....	23-9
Propagation Subprograms .....	23-10
Oracle Streams AQ Agent Subprograms .....	23-11
Alias Subprograms .....	23-12
<b>Summary of DBMS_AQADM Subprograms</b> .....	23-13
ADD_ALIAS_TO_LDAP Procedure .....	23-15
ADD_SUBSCRIBER Procedure .....	23-16
ALTER_AQ_AGENT Procedure .....	23-17
ALTER_PROPAGATION_SCHEDULE Procedure .....	23-18
ALTER_QUEUE Procedure .....	23-19
ALTER_QUEUE_TABLE Procedure .....	23-20
ALTER_SUBSCRIBER Procedure .....	23-21
CREATE_AQ_AGENT Procedure .....	23-22
CREATE_NP_QUEUE Procedure .....	23-23
CREATE_QUEUE Procedure .....	23-24
CREATE_QUEUE_TABLE Procedure .....	23-26
DEL_ALIAS_FROM_LDAP Procedure .....	23-29
DISABLE_DB_ACCESS Procedure .....	23-30
DISABLE_PROPAGATION_SCHEDULE Procedure .....	23-31
DROP_AQ_AGENT Procedure .....	23-32
DROP_QUEUE Procedure .....	23-33
DROP_QUEUE_TABLE Procedure .....	23-34
ENABLE_DB_ACCESS Procedure .....	23-35
ENABLE_JMS_TYPES Procedure .....	23-36
ENABLE_PROPAGATION_SCHEDULE Procedure .....	23-37
GET_WATERMARK Procedure .....	23-38
GRANT_QUEUE_PRIVILEGE Procedure .....	23-39
GRANT_SYSTEM_PRIVILEGE Procedure .....	23-40
MIGRATE_QUEUE_TABLE Procedure .....	23-41
PURGE_QUEUE_TABLE Procedure .....	23-42
QUEUE_SUBSCRIBERS Function .....	23-44
REMOVE_SUBSCRIBER Procedure .....	23-45
REVOKE_QUEUE_PRIVILEGE Procedure .....	23-46
REVOKE_SYSTEM_PRIVILEGE Procedure .....	23-47
SCHEDULE_PROPAGATION Procedure .....	23-48
SET_WATERMARK Procedure .....	23-50



	START_QUEUE Procedure .....	23-51
	STOP_QUEUE Procedure .....	23-52
	UNSCHEDULE_PROPAGATION Procedure .....	23-53
	VERIFY_QUEUE_TYPES Procedure.....	23-54
<b>24</b>	<b>DBMS_AQELM</b>	
	Summary of DBMS_AQELM Subprograms .....	24-2
	SET_MAILHOST Procedure.....	24-3
	SET_MAILPORT Procedure .....	24-4
	SET_SENDFROM Procedure.....	24-5
<b>25</b>	<b>DBMS_AQIN</b>	
	Using DBMS_AQIN .....	25-2
	Overview .....	25-3
<b>26</b>	<b>DBMS_ASSERT</b>	
	Using DBMS_ASSERT .....	26-2
	Operational Notes .....	26-3
	Summary of DBMS_ASSERT Subprograms .....	26-4
	ENQUOTE_LITERAL Function .....	26-5
	ENQUOTE_NAME Function .....	26-6
	NOOP Functions .....	26-7
	QUALIFIED_SQL_NAME Function .....	26-8
	SCHEMA_NAME Function.....	26-9
	SIMPLE_SQL_NAME Function .....	26-10
	SQL_OBJECT_NAME Function.....	26-11
<b>27</b>	<b>DBMS_AUDIT_MGMT</b>	
	Using DBMS_AUDIT_MGMT .....	27-2
	Overview .....	27-3
	Security Model.....	27-4
	Constants .....	27-5
	Views.....	27-7
	Subprogram Groups .....	27-8
	Audit Trail Management Subprograms.....	27-9
	Audit Trail Cleanup Subprograms.....	27-10
	Summary of DBMS_AUDIT_MGMT Subprograms .....	27-11
	CLEAN_AUDIT_TRAIL Procedure .....	27-12
	CLEAR_AUDIT_TRAIL_PROPERTY Procedure .....	27-14
	CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure .....	27-16
	CREATE_PURGE_JOB Procedure.....	27-17
	DEINIT_CLEANUP Procedure.....	27-19
	DROP_PURGE_JOB Procedure.....	27-20
	GET_AUDIT_COMMIT_DELAY Function.....	27-21
	INIT_CLEANUP Procedure .....	27-22
	IS_CLEANUP_INITIALIZED Function.....	27-24

SET_AUDIT_TRAIL_LOCATION Procedure.....	27-25
SET_AUDIT_TRAIL_PROPERTY Procedure .....	27-27
SET_LAST_ARCHIVE_TIMESTAMP Procedure.....	27-30
SET_PURGE_JOB_INTERVAL Procedure .....	27-32
SET_PURGE_JOB_STATUS Procedure.....	27-33
<b>28 DBMS_AUTO_SQLTUNE</b>	
Using DBMS_AUTO_SQLTUNE.....	28-2
Overview .....	28-3
Security Model.....	28-4
<b>Summary of DBMS_AUTO_SQLTUNE Subprograms.....</b>	<b>28-5</b>
EXECUTE_AUTO_TUNING_TASK Function & Procedure .....	28-6
REPORT_AUTO_TUNING_TASK Function .....	28-7
SET_AUTO_TUNING_TASK_PARAMETER Procedures.....	28-9
<b>29 DBMS_AUTO_TASK_ADMIN</b>	
Using DBMS_AUTO_TASK_ADMIN .....	29-2
Constants .....	29-3
<b>Summary of DBMS_AUTO_TASK_ADMIN Subprograms.....</b>	<b>29-4</b>
DISABLE Procedures.....	29-5
ENABLE Procedures.....	29-6
GET_CLIENT_ATTRIBUTES Procedure .....	29-7
GET_P1_RESOURCES Procedure.....	29-8
OVERRIDE_PRIORITY Procedures .....	29-9
SET_CLIENT_SERVICE Procedure.....	29-10
SET_P1_RESOURCES Procedure.....	29-11
<b>30 DBMS_AUTO_TASK_IMMEDIATE</b>	
<b>Summary of DBMS_AUTO_TASK_IMMEDIATE Subprograms .....</b>	<b>30-2</b>
GATHER_OPTIMIZER_STATS Procedure .....	30-3
<b>31 DBMS_AW_STATS</b>	
Using DBMS_AW_STATS .....	31-2
<b>Summary of DBMS_AW_STATS Subprograms.....</b>	<b>31-3</b>
ANALYZE Procedure.....	31-4
CLEAR Procedure.....	31-7
<b>32 DBMS_CAPTURE_ADM</b>	
Using DBMS_CAPTURE_ADM.....	32-2
Overview .....	32-3
Security Model.....	32-4
<b>Summary of DBMS_CAPTURE_ADM Subprograms .....</b>	<b>32-5</b>
ABORT_GLOBAL_INSTANTIATION Procedure .....	32-7
ABORT_SCHEMA_INSTANTIATION Procedure .....	32-8
ABORT_SYNC_INSTANTIATION Procedure.....	32-9

ABORT_TABLE_INSTANTIATION Procedure.....	32-10
ALTER_CAPTURE Procedure .....	32-11
ALTER_SYNC_CAPTURE Procedure .....	32-16
BUILD Procedure .....	32-18
CREATE_CAPTURE Procedure .....	32-19
CREATE_SYNC_CAPTURE Procedure.....	32-29
DROP_CAPTURE Procedure .....	32-31
INCLUDE_EXTRA_ATTRIBUTE Procedure .....	32-33
PREPARE_GLOBAL_INSTANTIATION Procedure.....	32-35
PREPARE_SCHEMA_INSTANTIATION Procedure.....	32-36
PREPARE_SYNC_INSTANTIATION Function .....	32-37
PREPARE_TABLE_INSTANTIATION Procedure.....	32-38
SET_PARAMETER Procedure .....	32-39
START_CAPTURE Procedure.....	32-49
STOP_CAPTURE Procedure .....	32-50

### 33 DBMS\_CDC\_PUBLISH

<b>Using DBMS_CDC_PUBLISH</b> .....	33-2
Overview .....	33-3
Deprecated Subprograms .....	33-4
Security Model.....	33-5
Views.....	33-6
<b>Summary of DBMS_CDC_PUBLISH Subprograms</b> .....	33-7
ALTER_AUTOLOG_CHANGE_SOURCE Procedure .....	33-8
ALTER_CHANGE_SET Procedure .....	33-10
ALTER_CHANGE_TABLE Procedure .....	33-13
ALTER_HOTLOG_CHANGE_SOURCE Procedure .....	33-16
CREATE_AUTOLOG_CHANGE_SOURCE Procedure.....	33-18
CREATE_CHANGE_SET Procedure .....	33-20
CREATE_CHANGE_TABLE Procedure .....	33-23
CREATE_HOTLOG_CHANGE_SOURCE Procedure.....	33-27
DROP_CHANGE_SET Procedure .....	33-29
DROP_CHANGE_SOURCE Procedure.....	33-30
DROP_CHANGE_TABLE Procedure .....	33-31
DROP_SUBSCRIPTION Procedure.....	33-32
GET_DDLOPER Function.....	33-33
PURGE Procedure.....	33-34
PURGE_CHANGE_SET Procedure.....	33-35
PURGE_CHANGE_TABLE Procedure.....	33-36

### 34 DBMS\_CDC\_SUBSCRIBE

<b>Using DBMS_CDC_SUBSCRIBE</b> .....	34-2
Overview .....	34-3
Deprecated Subprograms .....	34-5
Security Model.....	34-6
Views.....	34-7
<b>Summary of DBMS_CDC_SUBSCRIBE Subprograms</b> .....	34-8

ACTIVATE_SUBSCRIPTION Procedure.....	34-9
CREATE_SUBSCRIPTION Procedure .....	34-10
DROP_SUBSCRIPTION Procedure.....	34-12
EXTEND_WINDOW Procedure .....	34-13
PURGE_WINDOW Procedure.....	34-15
SUBSCRIBE Procedure .....	34-16
<b>35 DBMS_COMPARISON</b>	
<b>Using DBMS_COMPARISON</b> .....	35-2
Overview .....	35-3
Security Model.....	35-5
Constants .....	35-6
Views.....	35-8
Operational Notes .....	35-9
<b>Data Structures</b> .....	35-13
COMPARISON_TYPE Record Type .....	35-14
<b>Summary of DBMS_COMPARISON Subprograms</b> .....	35-15
COMPARE Function.....	35-16
CONVERGE Procedure.....	35-18
CREATE_COMPARISON Procedure.....	35-20
DROP_COMPARISON Procedure .....	35-24
PURGE_COMPARISON Procedure .....	35-25
RECHECK Function .....	35-26
<b>36 DBMS_COMPRESSION</b>	
<b>Using DBMS_COMPRESSION</b> .....	36-2
Overview .....	36-3
Security Model.....	36-4
Constants .....	36-5
Views.....	36-6
<b>Summary of DBMS_COMPRESSION Subprograms</b> .....	36-7
GET_COMPRESSION_RATIO Procedure .....	36-8
GET_COMPRESSION_TYPE Function.....	36-10
<b>37 DBMS_CONNECTION_POOL</b>	
<b>Summary of DBMS_CONNECTION_POOL Subprograms</b> .....	37-2
ALTER_PARAM Procedure .....	37-3
CONFIGURE_POOL Procedure .....	37-4
START_POOL Procedure.....	37-6
STOP_POOL Procedure .....	37-7
RESTORE_DEFAULTS Procedure.....	37-8
<b>38 DBMS_CQ_NOTIFICATION</b>	
<b>Using DBMS_CQ_NOTIFICATION</b> .....	38-2
Overview .....	38-3
Security Model.....	38-4

Constants .....	38-5
Operational Notes .....	38-7
Examples .....	38-8
<b>Data Structures</b> .....	38-15
CQ_NOTIFICATION\$_DESCRIPTOR Object Type .....	38-16
CQ_NOTIFICATION\$_QUERY Object Type .....	38-17
CQ_NOTIFICATION\$_QUERY_ARRAY Object (Array) Type .....	38-18
CQ_NOTIFICATION\$_TABLE Object Type .....	38-19
CQ_NOTIFICATION\$_TABLE_ARRAY Object (Array) Type .....	38-20
CQ_NOTIFICATION\$_ROW Object Type .....	38-21
CQ_NOTIFICATION\$_ROW_ARRAY Object (Array) Type .....	38-22
CQ_NOTIFICATION\$_REG_INFO Object Type .....	38-23
<b>Summary of DBMS_CQ_NOTIFICATION Subprograms</b> .....	38-26
CQ_NOTIFICATION_QUERYID Function .....	38-27
DEREGISTER Procedure .....	38-28
ENABLE_REG Procedure .....	38-29
NEW_REG_START Function .....	38-30
REG_END Procedure .....	38-31
SET_ROWID_THRESHOLD Procedure .....	38-32
<b>39 DBMS_CRYPTO</b>	
<b>Using the DBMS_CRYPTO Subprograms</b> .....	39-2
Overview .....	39-3
Security Model .....	39-4
Types .....	39-5
Algorithms .....	39-6
Restrictions .....	39-8
Exceptions .....	39-9
Operational Notes .....	39-10
Examples .....	39-12
<b>Summary of DBMS_CRYPTO Subprograms</b> .....	39-13
DECRYPT Function .....	39-14
DECRYPT Procedures .....	39-15
ENCRYPT Function .....	39-16
ENCRYPT Procedures .....	39-17
HASH Function .....	39-18
MAC Function .....	39-19
RANDOMBYTES Function .....	39-20
RANDOMINTEGER Function .....	39-21
RANDOMNUMBER Function .....	39-22
<b>40 DBMS_CSX_ADMIN</b>	
<b>Using DBMS_CSX_ADMIN</b> .....	40-2
Overview .....	40-3
Security Model .....	40-4
Constants .....	40-5
Examples .....	40-6

<b>Summary of DBMS_CSX_ADMIN</b> .....	40-7
COPYDEFAULTTOKENTABLESET Procedure .....	40-8
GETTOKENTABLEINFO Procedure & Function .....	40-9
GETTOKENTABLEINFOBYTABLESPACE Procedure .....	40-10
NAMESPACEIDTABLE Function .....	40-11
PATHIDTABLE Function .....	40-12
QNAMEIDTABLE Function.....	40-13
REGISTERTOKENTABLESET Procedure .....	40-14
<b>41 DBMS_CUBE</b>	
<b>Using DBMS_CUBE</b> .....	41-2
Security Model.....	41-3
<b>Using SQL Aggregation Management</b> .....	41-4
Subprograms in SQL Aggregation Management .....	41-5
Requirements for the Relational Materialized View .....	41-6
Permissions for Managing and Querying Cube Materialized Views.....	41-7
Example of SQL Aggregation Management .....	41-8
<b>Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g</b> .....	41-12
Correcting Naming Conflicts .....	41-13
Simple Upgrade.....	41-15
Custom Upgrade .....	41-16
<b>Summary of DBMS_CUBE Subprograms</b> .....	41-18
BUILD Procedure .....	41-19
CREATE_EXPORT_OPTIONS Procedure.....	41-31
CREATE_IMPORT_OPTIONS Procedure.....	41-34
CREATE_MVIEW Function.....	41-36
DERIVE_FROM_MVIEW Function.....	41-41
DROP_MVIEW Procedure.....	41-43
EXPORT_XML Procedure.....	41-45
EXPORT_XML_TO_FILE Procedure.....	41-47
IMPORT_XML Procedure.....	41-49
INITIALIZE_CUBE_UPGRADE Procedure.....	41-52
REFRESH_MVIEW Procedure .....	41-54
UPGRADE_AW Procedure .....	41-56
VALIDATE_XML Procedure.....	41-58
<b>42 DBMS_CUBE_ADVISE</b>	
<b>Using DBMS_CUBE_ADVISE</b> .....	42-2
Security Model.....	42-3
<b>Summary of DBMS_CUBE_ADVISE Subprograms</b> .....	42-4
MV_CUBE_ADVICE Function.....	42-5
SET_CNS_EXCEPTION_LOG Procedure .....	42-8
TRACE Procedure .....	42-9
<b>43 DBMS_CUBE_LOG</b>	
<b>Using DBMS_CUBE_LOG</b> .....	43-2

Logging Types .....	43-3
Logging Targets.....	43-4
Verbosity Levels .....	43-5
Security Model.....	43-6
Creating Cube Logs .....	43-7
Cube Build Log.....	43-8
Cube Dimension Compile Log.....	43-10
Cube Operations Log.....	43-11
Cube Rejected Records Log .....	43-12
<b>Summary of DBMS_CUBE_LOG Subprograms .....</b>	<b>43-13</b>
DEFAULT_NAME Function .....	43-15
DISABLE Procedure .....	43-16
ENABLE Procedure .....	43-17
FLUSH Procedure .....	43-19
GET_LOG Procedure.....	43-20
GET_LOG_SPEC Function.....	43-22
GET_PARAMETER Function.....	43-23
LEVEL_HIGH Function .....	43-24
LEVEL_HIGHEST Function .....	43-25
LEVEL_LOW Function.....	43-26
LEVEL_LOWEST Function.....	43-27
LEVEL_MEDIUM Function.....	43-28
SET_LOG_SPEC Procedure .....	43-29
SET_PARAMETER Procedure .....	43-30
TABLE_CREATE Procedure.....	43-32
TARGET_FILE Function .....	43-33
TARGET_LOB Function.....	43-34
TARGET_TABLE Function.....	43-35
TARGET_TRACE Function .....	43-36
TYPE_BUILD Function.....	43-37
TYPE_DIMENSION_COMPILE Function.....	43-38
TYPE_OPERATIONS Function.....	43-39
TYPE_REJECTED_RECORDS Function .....	43-40
VERSION Function .....	43-41

## 44 DBMS\_DATA\_MINING

Using DBMS_DATA_MINING .....	44-2
Overview .....	44-3
Mining Model Objects .....	44-5
Security Model.....	44-7
Deprecated Subprograms .....	44-8
Mining Functions .....	44-9
Model Settings .....	44-10
Data Types.....	44-20
<b>Summary of DBMS_DATA_MINING Subprograms.....</b>	<b>44-23</b>
ADD_COST_MATRIX Procedure.....	44-25
ALTER_REVERSE_EXPRESSION Procedure .....	44-27

APPLY Procedure .....	44-30
COMPUTE_CONFUSION_MATRIX Procedure.....	44-34
COMPUTE_LIFT Procedure.....	44-40
COMPUTE_ROC Procedure.....	44-45
CREATE_MODEL Procedure.....	44-50
DROP_MODEL Procedure .....	44-54
EXPORT_MODEL Procedure.....	44-55
GET_ASSOCIATION_RULES Function .....	44-58
GET_DEFAULT_SETTINGS Function.....	44-62
GET_FREQUENT_ITEMSETS Function .....	44-63
GET_MODEL_COST_MATRIX Function.....	44-65
GET_MODEL_DETAILS_ABN Function .....	44-67
GET_MODEL_DETAILS_AI Function.....	44-69
GET_MODEL_DETAILS_GLM Function.....	44-71
GET_MODEL_DETAILS_GLOBAL Function .....	44-74
GET_MODEL_DETAILS_KM Function .....	44-77
GET_MODEL_DETAILS_NB Function .....	44-80
GET_MODEL_DETAILS_NMF Function.....	44-82
GET_MODEL_DETAILS_OC Function .....	44-84
GET_MODEL_DETAILS_SVM Function .....	44-88
GET_MODEL_DETAILS_XML Function .....	44-90
GET_MODEL_SETTINGS Function.....	44-92
GET_MODEL_SIGNATURE Function .....	44-93
GET_MODEL_TRANSFORMATIONS Function .....	44-95
GET_TRANSFORM_LIST Procedure.....	44-97
IMPORT_MODEL Procedure.....	44-100
RANK_APPLY Procedure .....	44-104
REMOVE_COST_MATRIX Procedure.....	44-107
RENAME_MODEL Procedure.....	44-108

## 45 DBMS\_DATA\_MINING\_TRANSFORM

Using DBMS_DATA_MINING_TRANSFORM .....	45-2
Overview .....	45-3
Operational Notes .....	45-6
Security Model.....	45-14
Types .....	45-15
Constants .....	45-16
Summary of DBMS_DATA_MINING_TRANSFORM Subprograms.....	45-17
CREATE_BIN_CAT Procedure.....	45-19
CREATE_BIN_NUM Procedure .....	45-21
CREATE_CLIP Procedure .....	45-23
CREATE_COL_REM Procedure .....	45-25
CREATE_MISS_CAT Procedure.....	45-26
CREATE_MISS_NUM Procedure.....	45-28
CREATE_NORM_LIN Procedure .....	45-30
DESCRIBE_STACK Procedure.....	45-32
GET_EXPRESSION Function .....	45-34



INSERT_AUTOBIN_NUM_EQWIDTH Procedure .....	45-35
INSERT_BIN_CAT_FREQ Procedure .....	45-39
INSERT_BIN_NUM_EQWIDTH Procedure .....	45-44
INSERT_BIN_NUM_QTILE Procedure .....	45-48
INSERT_BIN_SUPER Procedure .....	45-51
INSERT_CLIP_TRIM_TAIL Procedure .....	45-55
INSERT_CLIP_WINSOR_TAIL Procedure .....	45-58
INSERT_MISS_CAT_MODE Procedure .....	45-61
INSERT_MISS_NUM_MEAN Procedure .....	45-64
INSERT_NORM_LIN_MINMAX Procedure .....	45-67
INSERT_NORM_LIN_SCALE Procedure .....	45-70
INSERT_NORM_LIN_ZSCORE Procedure .....	45-73
SET_EXPRESSION Procedure .....	45-76
SET_TRANSFORM Procedure .....	45-78
STACK_BIN_CAT Procedure.....	45-80
STACK_BIN_NUM Procedure.....	45-82
STACK_CLIP Procedure .....	45-85
STACK_COL_REM Procedure.....	45-87
STACK_MISS_CAT Procedure .....	45-89
STACK_MISS_NUM Procedure .....	45-91
STACK_NORM_LIN Procedure .....	45-93
XFORM_BIN_CAT Procedure .....	45-95
XFORM_BIN_NUM Procedure.....	45-97
XFORM_CLIP Procedure.....	45-100
XFORM_COL_REM Procedure.....	45-102
XFORM_EXPR_NUM Procedure.....	45-104
XFORM_EXPR_STR Procedure.....	45-106
XFORM_MISS_CAT Procedure .....	45-109
XFORM_MISS_NUM Procedure .....	45-112
XFORM_NORM_LIN Procedure .....	45-114
XFORM_STACK Procedure.....	45-117

## 46 DBMS\_DATAPUMP

Using DBMS_DATAPUMP .....	46-2
Overview .....	46-3
Security Model.....	46-4
Constants .....	46-5
<b>Data Structures</b> .....	46-6
Data Structures - Object Types.....	46-7
<b>Summary of DBMS_DATAPUMP Subprograms</b> .....	46-12
ADD_FILE Procedure.....	46-13
ATTACH Function.....	46-16
DATA_FILTER Procedures .....	46-18
DATA_REMAP Procedure .....	46-21
DETACH Procedure .....	46-23
GET_DUMPFILINFO Procedure.....	46-24
GET_STATUS Procedure .....	46-26

LOG_ENTRY Procedure .....	46-29
METADATA_FILTER Procedure .....	46-30
METADATA_REMAP Procedure .....	46-33
METADATA_TRANSFORM Procedure .....	46-36
OPEN Function.....	46-39
SET_PARALLEL Procedure .....	46-42
SET_PARAMETER Procedures.....	46-44
START_JOB Procedure.....	46-53
STOP_JOB Procedure .....	46-55
WAIT_FOR_JOB Procedure.....	46-57

## 47 DBMS\_DBFS\_CONTENT

<b>Using DBMS_DBFS_CONTENT .....</b>	<b>47-2</b>
Overview .....	47-3
Security Model.....	47-4
Constants .....	47-5
Exceptions .....	47-18
Operational Notes .....	47-19
<b>Data Structures.....</b>	<b>47-23</b>
FEATURE_T Record Type .....	47-24
MOUNT_T Record Type.....	47-25
PATH_ITEM_T Record Type .....	47-26
PROP_ITEM_T Record Type.....	47-28
PROPERTY_T Record Type.....	47-29
STORE_T Record Type.....	47-30
FEATURES_T Table Type.....	47-31
MOUNTS_T Table Type.....	47-32
PATH_ITEMS_T Table Type .....	47-33
PROP_ITEMS_T Table Type.....	47-34
PROPERTIES_T Table Type .....	47-35
STORES_T Table Type.....	47-36
<b>Summary of DBMS_DBFS_CONTENT Subprograms.....</b>	<b>47-37</b>
CHECKACCESS Function .....	47-40
CHECKSPI Functions and Procedures .....	47-41
CREATEDIRECTORY Procedures .....	47-42
CREATEFILE Procedures .....	47-43
CREATELINK Procedures.....	47-44
CREATEREFERENCE Procedures .....	47-45
DECODEFEATURES Function .....	47-46
DELETECONTENT Procedure .....	47-47
DELETEDIRECTORY Procedure.....	47-48
DELETEFILE Procedure.....	47-49
FEATURENAME Function.....	47-50
FLUSHSTATS Function.....	47-51
GETDEFAULTACL Procedure .....	47-52
GETDEFAULTASOF Procedure .....	47-53
GETTDEFAULTCONTEXT Procedure.....	47-54

---

GETDEFAULTOWNER Procedure .....	47-55
GETDEFAULTPRINCIPAL Procedure.....	47-56
GETFEATURESBYMOUNT Function.....	47-57
GETFEATURESBYNAME Function.....	47-58
GETFEATURESBYPATH Function .....	47-59
GETPATH Procedures .....	47-60
GETPATHBYMOUNTID Function .....	47-63
GETPATHBYSTOREID Function .....	47-64
GETPATHNOWAIT Procedures .....	47-65
GETSTOREBYMOUNT Function .....	47-67
GETSTOREBYNAME Function .....	47-68
GETSTOREBYPATH Function.....	47-69
GETSTATS Procedure .....	47-70
GETTRACE Function .....	47-71
GETVERSION Function .....	47-72
LIST Function.....	47-73
LISTALLPROPERTIES Function .....	47-74
LISTALLCONTENT Function.....	47-75
LISTMOUNTS Function.....	47-76
LISTSTORES Function.....	47-77
LOCKPATH Procedure.....	47-78
MOUNTSTORE Procedure.....	47-79
NORMALIZEPATH Functions .....	47-81
PROPANY Functions.....	47-82
PROPERTIESH2T Function .....	47-83
PROPERTIEST2H Function .....	47-84
PROPNUMBER Function.....	47-85
PROPRAW Function.....	47-86
PROPTIMESTAMP Function .....	47-87
PROPVARCHAR2 Function.....	47-88
PURGEALL Procedure.....	47-89
PURGEPATH Procedure .....	47-90
PUTPATH Procedures .....	47-91
REGISTERSTORE Procedure .....	47-93
RENAMEPATH Procedures.....	47-94
RESTOREALL Procedure.....	47-95
RESTOREPATH Procedure .....	47-96
SEARCH Function .....	47-97
SETDEFAULTACL Procedure .....	47-98
SETDEFAULTASOF Procedure .....	47-99
SETDEFAULTCONTEXT Procedure .....	47-100
SETDEFAULTOWNER Procedure .....	47-101
SETDEFAULTPRINCIPAL Procedure.....	47-102
SETPATH Procedures .....	47-103
SETSTATS Procedure .....	47-104
SETTRACE Procedure.....	47-105
SPACEUSAGE Procedure.....	47-106

TRACE Procedure .....	47-107
TRACEENABLED Function .....	47-108
UNLOCKPATH Procedure .....	47-109
UNMOUNTSTORE Procedure .....	47-110
UNREGISTERSTORE Procedure .....	47-111

## 48 DBMS\_DBFS\_CONTENT\_SPI

<b>Using DBMS_DBFS_CONTENT_SPI</b> .....	48-2
Overview .....	48-3
Security Model.....	48-4
Operational Notes .....	48-5
<b>Summary of DBMS_DBFS_CONTENT_SPI Subprograms</b> .....	48-9
CHECKACCESS Function .....	48-11
CREATEDIRECTORY Procedure .....	48-12
CREATEFILE Procedure .....	48-13
CREATELINK Procedure .....	48-14
CREATEREFERENCE Procedure .....	48-15
DELETECONTENT Procedure .....	48-16
DELETEDIRECTORY Procedure .....	48-17
DELETEFILE Procedure.....	48-18
GETFEATURES Function .....	48-19
GETPATH Procedures.....	48-20
GETPATHBYSTOREID Function .....	48-22
GETPATHNOWAIT Procedure.....	48-23
GETSTOREID Function.....	48-24
GETVERSION Function .....	48-25
LIST Function.....	48-26
LOCKPATH Procedure.....	48-27
PURGEALL Procedure.....	48-28
PURGEPATH Procedure .....	48-29
PUTPATH Procedures .....	48-30
RENAMEPATH Procedure .....	48-32
RESTOREALL Procedure.....	48-33
RESTOREPATH Procedure .....	48-34
SEARCH Function .....	48-35
SETPATH Procedure .....	48-36
SPACEUSAGE Procedure.....	48-37
UNLOCKPATH Procedure .....	48-38

## 49 DBMS\_DBFS\_HS

<b>Using DBMS_DBFS_HS</b> .....	49-2
Overview .....	49-3
Security Model.....	49-4
Constants .....	49-5
Operational Notes .....	49-10
<b>Summary of DBMS_DBFS_HS Subprograms</b> .....	49-11
CLEANUPUNUSEDBACKUPFILES Procedure .....	49-12

CREATEBUCKET Procedure .....	49-13
CREATESTORE Procedure.....	49-14
DEREGSTORECOMMAND Function .....	49-16
DROPSTORE Procedure .....	49-17
FLUSHCACHE Procedure.....	49-18
GETSTOREPROPERTY Function .....	49-19
RECONFIGCACHE Procedure.....	49-20
REGISTERSTORECOMMAND Procedure .....	49-21
SENDCOMMAND Procedures.....	49-22
SETSTOREPROPERTY Procedure.....	49-23
STOREPUSH Procedure.....	49-26
<b>50 DBMS_DBFS_SFS</b>	
Using DBMS_DBFS_SFS.....	50-2
Overview .....	50-3
Security Model.....	50-4
Constants .....	50-5
<b>Summary of DBMS_DBFS_SFS Subprograms</b> .....	50-6
CREATEFILESYSTEM Procedure .....	50-7
CREATESTORE Procedure.....	50-9
DROPFILESYSTEM Procedures .....	50-10
INITFS Procedure.....	50-11
<b>51 DBMS_DB_VERSION</b>	
Using DBMS_DB_VERSION .....	51-2
Overview .....	51-3
Constants .....	51-4
Examples .....	51-5
<b>52 DBMS_DEBUG</b>	
Using DBMS_DEBUG.....	52-2
Overview .....	52-3
Constants .....	52-4
Variables .....	52-5
Exceptions .....	52-6
Operational Notes .....	52-8
<b>Data Structures</b> .....	52-15
BREAKPOINT_INFO Record Type.....	52-16
PROGRAM_INFO Record Type .....	52-17
RUNTIME_INFO Record Type.....	52-18
BACKTRACE_TABLE Table Type .....	52-19
BREAKPOINT_TABLE Table Type.....	52-20
INDEX_TABLE Table Type .....	52-21
VC2_TABLE Table Type .....	52-22
<b>Summary of DBMS_DEBUG Subprograms</b> .....	52-23
ATTACH_SESSION Procedure.....	52-25

CONTINUE Function .....	52-26
DEBUG_OFF Procedure.....	52-27
DEBUG_ON Procedure.....	52-28
DELETE_BREAKPOINT Function .....	52-29
DELETE_OER_BREAKPOINT Function .....	52-30
DETACH_SESSION Procedure.....	52-31
DISABLE_BREAKPOINT Function.....	52-32
ENABLE_BREAKPOINT Function .....	52-33
EXECUTE Procedure .....	52-34
GET_INDEXES Function.....	52-36
GET_MORE_SOURCE Procedure .....	52-37
GET_LINE_MAP Function .....	52-38
GET_RUNTIME_INFO Function.....	52-39
GET_TIMEOUT_BEHAVIOUR Function.....	52-40
GET_VALUE Function .....	52-41
INITIALIZE Function .....	52-43
PING Procedure .....	52-45
PRINT_BACKTRACE Procedure .....	52-46
PRINT_INSTANTIATIONS Procedure .....	52-47
PROBE_VERSION Procedure .....	52-48
SELF_CHECK Procedure.....	52-49
SET_BREAKPOINT Function.....	52-50
SET_OER_BREAKPOINT Function .....	52-51
SET_TIMEOUT Function .....	52-52
SET_TIMEOUT_BEHAVIOUR Procedure .....	52-53
SET_VALUE Function.....	52-54
SHOW_BREAKPOINTS Procedures.....	52-56
SHOW_FRAME_SOURCE Procedure .....	52-57
SHOW_SOURCE Procedures.....	52-58
SYNCHRONIZE Function .....	52-60
TARGET_PROGRAM_RUNNING Procedure .....	52-61

## 53 DBMS\_DDL

<b>Using DBMS_DDL .....</b>	<b>53-2</b>
Deprecated Subprograms .....	53-3
Security Model.....	53-4
Operational Notes .....	53-5
<b>Summary of DBMS_DDL Subprograms .....</b>	<b>53-6</b>
ALTER_COMPILE Procedure .....	53-7
ALTER_TABLE_NOT_REFERENCEABLE Procedure.....	53-8
ALTER_TABLE_REFERENCEABLE Procedure .....	53-9
CREATE_WRAPPED Procedures.....	53-10
IS_TRIGGER_FIRE_ONCE Function .....	53-12
SET_TRIGGER_FIRING_PROPERTY Procedures .....	53-13
WRAP Functions .....	53-15

<b>54</b>	<b>DBMS_DEFER</b>	
	Documentation of DBMS_DEFER.....	54-2
<b>55</b>	<b>DBMS_DEFER_QUERY</b>	
	Documentation of DBMS_DEFER_QUERY.....	55-2
<b>56</b>	<b>DBMS_DEFER_SYS</b>	
	Documentation of DBMS_DEFER_SYS.....	56-2
<b>57</b>	<b>DBMS_DESCRIBE</b>	
	Using DBMS_DESCRIBE.....	57-2
	Overview.....	57-3
	Security Model.....	57-4
	Types.....	57-5
	Exceptions.....	57-6
	Examples.....	57-7
	<b>Summary of DBMS_DESCRIBE Subprograms.....</b>	<b>57-10</b>
	DESCRIBE_PROCEDURE Procedure.....	57-11
<b>58</b>	<b>DBMS_DG</b>	
	Using DBMS_DG.....	58-2
	Security Model.....	58-3
	<b>Summary of the DBMS_DG Subprogram.....</b>	<b>58-4</b>
	INITIATE_FS_FAILOVER Procedure.....	58-5
<b>59</b>	<b>DBMS_DIMENSION</b>	
	Using DBMS_DIMENSION.....	59-2
	Security Model.....	59-3
	<b>Summary of DBMS_DIMENSION Subprograms.....</b>	<b>59-4</b>
	DESCRIBE_DIMENSION Procedure.....	59-5
	VALIDATE_DIMENSION Procedure.....	59-6
<b>60</b>	<b>DBMS_DST</b>	
	Using DBMS_DST.....	60-2
	Overview.....	60-3
	Security Model.....	60-4
	Views.....	60-5
	<b>Summary of DBMS_DST Subprograms.....</b>	<b>60-6</b>
	BEGIN_PREPARE Procedure.....	60-7
	BEGIN_UPGRADE Procedure.....	60-8
	CREATE_AFFECTED_TABLE Procedure.....	60-9
	CREATE_ERROR_TABLE Procedure.....	60-10
	CREATE_TRIGGER_TABLE Procedure.....	60-11
	END_PREPARE Procedure.....	60-12

END_UPGRADE Procedure.....	60-13
FIND_AFFECTED_TABLES Procedure.....	60-14
UPGRADE_DATABASE Procedure.....	60-15
UPGRADE_SCHEMA Procedure.....	60-17
UPGRADE_TABLE Procedure.....	60-19
<b>61 DBMS_DISTRIBUTED_TRUST_ADMIN</b>	
<b>Using DBMS_DISTRIBUTED_TRUST_ADMIN</b> .....	61-2
Overview .....	61-3
Security Model.....	61-4
Examples .....	61-5
<b>Summary of DBMS_DISTRIBUTED_TRUST_ADMIN Subprograms</b> .....	61-6
ALLOW_ALL Procedure .....	61-7
ALLOW_SERVER Procedure .....	61-8
DENY_ALL Procedure .....	61-9
DENY_SERVER Procedure.....	61-10
<b>62 DBMS_EDITIONS_UTILITIES</b>	
<b>Using DBMS_EDITIONS_UTILITIES</b> .....	62-2
Overview .....	62-3
Security Model.....	62-4
Exceptions .....	62-5
<b>Summary of DBMS_EDITIONS_UTILITIES Subprograms</b> .....	62-6
SET_EDITIONING_VIEWS_READ_ONLY Procedure.....	62-7
<b>63 DBMS_EPG</b>	
<b>Using DBMS_EPG</b> .....	63-2
Overview .....	63-3
Security Model.....	63-4
Exceptions .....	63-5
<b>Data Structures</b> .....	63-6
<b>Subprogram Groups</b> .....	63-7
Configuration Subprograms.....	63-8
Authorization Subprograms.....	63-9
<b>Summary of DBMS_EPG Subprograms</b> .....	63-10
AUTHORIZE_DAD Procedure.....	63-11
CREATE_DAD Procedure .....	63-12
DEAUTHORIZE_DAD Procedure .....	63-13
DELETE_DAD_ATTRIBUTE Procedure .....	63-14
DELETE_GLOBAL_ATTRIBUTE Procedure .....	63-15
DROP_DAD Procedure.....	63-16
GET_ALL_DAD_ATTRIBUTES Procedure.....	63-17
GET_ALL_DAD_MAPPINGS Procedure.....	63-18
GET_ALL_GLOBAL_ATTRIBUTES Procedure .....	63-19
GET_DAD_ATTRIBUTE Function .....	63-20
GET_DAD_LIST Procedure.....	63-21



GET_GLOBAL_ATTRIBUTE Function.....	63-22
MAP_DAD Procedure.....	63-23
SET_DAD_ATTRIBUTE Procedure.....	63-24
SET_GLOBAL_ATTRIBUTE Procedure .....	63-26
UNMAP_DAD Procedure .....	63-27
<b>64 DBMS_ERRLOG</b>	
Using DBMS_ERRLOG .....	64-2
Security Model.....	64-3
<b>Summary of DBMS_ERRLOG Subprograms</b> .....	64-4
CREATE_ERROR_LOG Procedure .....	64-5
<b>65 DBMS_EXPFIL</b>	
Using DBMS_EXPFIL.....	65-2
Security Model.....	65-3
<b>Summary of Expression Filter Subprograms</b> .....	65-4
ADD_ELEMENTARY_ATTRIBUTE Procedures .....	65-5
ADD_FUNCTIONS Procedure .....	65-8
ASSIGN_ATTRIBUTE_SET Procedure.....	65-10
BUILD_EXCEPTIONS_TABLE Procedure.....	65-12
CLEAR_EXPRSET_STATS Procedure.....	65-13
COPY_ATTRIBUTE_SET Procedure.....	65-14
CREATE_ATTRIBUTE_SET Procedure .....	65-15
DEFAULT_INDEX_PARAMETERS Procedure .....	65-17
DEFAULT_XPINDEX_PARAMETERS Procedure .....	65-19
DEFRAG_INDEX Procedure .....	65-21
DROP_ATTRIBUTE_SET Procedure.....	65-22
GET_EXPRSET_STATS Procedure .....	65-23
GRANT_PRIVILEGE Procedure.....	65-24
INDEX_PARAMETERS Procedure .....	65-25
MODIFY_OPERATOR_LIST Procedure.....	65-27
REVOKE_PRIVILEGE Procedure.....	65-28
SYNC_TEXT_INDEXES Procedure .....	65-29
UNASSIGN_ATTRIBUTE_SET Procedure .....	65-30
VALIDATE_EXPRESSIONS Procedure.....	65-31
XPINDEX_PARAMETERS Procedure .....	65-32
<b>66 DBMS_FGA</b>	
Using DBMS_FGA.....	66-2
Security Model.....	66-3
Operational Notes .....	66-4
<b>Summary of DBMS_FGA Subprograms</b> .....	66-5
ADD_POLICY Procedure .....	66-6
DISABLE_POLICY Procedure.....	66-11
DROP_POLICY Procedure .....	66-12
ENABLE_POLICY Procedure .....	66-13

---

## 67 DBMS\_FILE\_GROUP

Using DBMS_FILE_GROUP .....	67-2
Overview .....	67-3
Security Model.....	67-4
Constants .....	67-5
<b>Summary of DBMS_FILE_GROUP Subprograms .....</b>	<b>67-6</b>
ADD_FILE Procedure.....	67-7
ALTER_FILE Procedure .....	67-9
ALTER_FILE_GROUP Procedure .....	67-11
ALTER_VERSION Procedure .....	67-13
CREATE_FILE_GROUP Procedure.....	67-15
CREATE_VERSION Procedure.....	67-17
DROP_FILE_GROUP Procedure .....	67-18
DROP_VERSION Procedure .....	67-19
GRANT_OBJECT_PRIVILEGE Procedure .....	67-20
GRANT_SYSTEM_PRIVILEGE Procedure .....	67-21
PURGE_FILE_GROUP Procedure .....	67-22
REMOVE_FILE Procedure.....	67-23
REVOKE_OBJECT_PRIVILEGE Procedure .....	67-24
REVOKE_SYSTEM_PRIVILEGE Procedure .....	67-25

## 68 DBMS\_FILE\_TRANSFER

Using DBMS_FILE_TRANSFER.....	68-2
Operating Notes .....	68-3
<b>Summary of DBMS_FILE_TRANSFER Subprograms.....</b>	<b>68-4</b>
COPY_FILE Procedure .....	68-5
GET_FILE Procedure .....	68-7
PUT_FILE Procedure .....	68-9

## 69 DBMS\_FLASHBACK

Using DBMS_FLASHBACK .....	69-2
Overview .....	69-3
Security Model.....	69-4
Types .....	69-5
Exceptions .....	69-6
Operational Notes .....	69-7
Examples .....	69-8
<b>Summary of DBMS_FLASHBACK Subprograms.....</b>	<b>69-11</b>
DISABLE Procedure .....	69-12
ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure .....	69-13
ENABLE_AT_TIME Procedure.....	69-14
GET_SYSTEM_CHANGE_NUMBER Function.....	69-15
TRANSACTION_BACKOUT Procedures.....	69-16

## 70 DBMS\_FLASHBACK\_ARCHIVE

Using DBMS_FLASHBACK_ARCHIVE.....	70-2
-----------------------------------	------

Overview .....	70-3
Security Model.....	70-4
Examples .....	70-5
<b>Summary of DBMS_FLASHBACK_ARCHIVE Subprograms.....</b>	<b>70-7</b>
DISASSOCIATE_FBA Procedure .....	70-8
REASSOCIATE_FBA Procedure.....	70-9
<b>71 DBMS_FREQUENT_ITEMSET</b>	
<b>Summary of DBMS_FREQUENT_ITEMSET Subprograms.....</b>	<b>71-2</b>
FI_HORIZONTAL Function.....	71-3
FI_TRANSACTIONAL Function.....	71-5
<b>72 DBMS_HM</b>	
<b>Using DBMS_HM .....</b>	<b>72-2</b>
Security Model.....	72-3
<b>Summary of DBMS_HM Subprograms.....</b>	<b>72-4</b>
GET_RUN_REPORT Function .....	72-5
RUN_CHECK Procedure .....	72-6
<b>73 DBMS_HPROF</b>	
<b>Summary of DBMS_HPROF Subprograms.....</b>	<b>73-2</b>
ANALYZE Function .....	73-3
START_PROFILING Procedure.....	73-5
STOP_PROFILING Procedure .....	73-6
<b>74 DBMS_HS_PARALLEL</b>	
<b>Using DBMS_HS_PARALLEL.....</b>	<b>74-2</b>
<b>Summary of DBMS_HS_PARALLEL Subprograms .....</b>	<b>74-3</b>
CREATE_OR_REPLACE_VIEW .....	74-4
CREATE_TABLE_TEMPLATE .....	74-6
DROP_VIEW.....	74-7
LOAD_TABLE .....	74-8
<b>75 DBMS_HS_PASSTHROUGH</b>	
<b>Using DBMS_HS_PASSTHROUGH.....</b>	<b>75-2</b>
Overview .....	75-3
Operational Notes .....	75-4
<b>Summary of DBMS_HS_PASSTHROUGH Subprograms .....</b>	<b>75-5</b>
BIND_INOUT_VARIABLE Procedure .....	75-6
BIND_INOUT_VARIABLE_RAW Procedure.....	75-7
BIND_OUT_VARIABLE Procedure .....	75-8
BIND_OUT_VARIABLE_RAW Procedure .....	75-9
BIND_VARIABLE Procedure.....	75-10
BIND_VARIABLE_RAW Procedure .....	75-11
CLOSE_CURSOR Procedure .....	75-12

EXECUTE_IMMEDIATE Procedure .....	75-13
EXECUTE_NON_QUERY Function.....	75-14
FETCH_ROW Function.....	75-15
GET_VALUE Procedure.....	75-16
GET_VALUE_RAW Procedure.....	75-17
OPEN_CURSOR Function .....	75-18
PARSE Procedure.....	75-19
<b>76 DBMS_IOT</b>	
<b>Summary of DBMS_IOT Subprograms.....</b>	<b>76-2</b>
BUILD_CHAIN_ROWS_TABLE Procedure .....	76-3
BUILD_EXCEPTIONS_TABLE Procedure.....	76-4
<b>77 DBMS_JAVA</b>	
Documentation of DBMS_JAVA .....	77-2
<b>78 DBMS_JOB</b>	
<b>Using DBMS_JOB .....</b>	<b>78-2</b>
Security Model.....	78-3
Operational Notes .....	78-4
<b>Summary of DBMS_JOB Subprograms.....</b>	<b>78-6</b>
BROKEN Procedure.....	78-7
CHANGE Procedure .....	78-8
INSTANCE Procedure.....	78-9
INTERVAL Procedure.....	78-10
NEXT_DATE Procedure .....	78-11
REMOVE Procedure .....	78-12
RUN Procedure .....	78-13
SUBMIT Procedure .....	78-14
USER_EXPORT Procedures.....	78-16
WHAT Procedure.....	78-17
<b>79 DBMS_LDAP</b>	
Documentation of DBMS_LDAP .....	79-2
<b>80 DBMS_LDAP_UTL</b>	
Documentation of DBMS_LDAP_UTL.....	80-2
<b>81 DBMS_LIBCACHE</b>	
<b>Using DBMS_LIBCACHE .....</b>	<b>81-2</b>
Overview .....	81-3
Security Model.....	81-4
<b>Summary of DBMS_LIBCACHE Subprograms.....</b>	<b>81-5</b>
COMPILE_FROM_REMOTE Procedure .....	81-6

---

## 82 DBMS\_LOB

<b>Using DBMS_LOB</b> .....	82-2
Overview .....	82-3
Security Model.....	82-4
Constants .....	82-5
Datatypes.....	82-7
Operational Notes .....	82-8
Rules and Limits.....	82-11
Exceptions .....	82-15
<b>Summary of DBMS_LOB Subprograms</b> .....	82-16
APPEND Procedures .....	82-19
CLOSE Procedure .....	82-20
COMPARE Functions.....	82-21
CONVERTTOBLOB Procedure.....	82-23
CONVERTTOCLOB Procedure .....	82-26
COPY Procedures.....	82-29
COPY_DBFS_LINK Procedures.....	82-31
COPY_FROM_DBFS_LINK.....	82-32
CREATETEMPORARY Procedures .....	82-33
DBFS_LINK_GENERATE_PATH Functions .....	82-34
ERASE Procedures .....	82-35
FILECLOSE Procedure .....	82-37
FILECLOSEALL Procedure .....	82-38
FILEEXISTS Function .....	82-39
FILEGETNAME Procedure .....	82-40
FILEISOPEN Function.....	82-41
FILEOPEN Procedure.....	82-42
FRAGMENT_DELETE Procedure .....	82-43
FRAGMENT_INSERT Procedures .....	82-44
FRAGMENT_MOVE Procedure .....	82-45
FRAGMENT_REPLACE Procedures .....	82-46
FREETEMPORARY Procedures.....	82-48
GET_DBFS_LINK Functions .....	82-49
GET_DBFS_LINK_STATE Procedures .....	82-50
GETCONTENTTYPE Functions .....	82-51
GET_STORAGE_LIMIT Function.....	82-52
GETCHUNKSIZE Functions .....	82-53
GETLENGTH Functions .....	82-55
GETOPTIONS Functions .....	82-56
INSTR Functions .....	82-57
ISOPEN Functions .....	82-59
ISTEMPORARY Functions .....	82-60
LOADBLOBFROMFILE Procedure.....	82-61
LOADCLOBFROMFILE Procedure .....	82-63
LOADFROMFILE Procedure .....	82-67
MOVE_TO_DBFS_LINK Procedures .....	82-69
OPEN Procedures .....	82-70

READ Procedures .....	82-72
SET_DBFS_LINK Procedures.....	82-74
SETCONTENTTYPE Procedure.....	82-75
SETOPTIONS Procedures.....	82-76
SUBSTR Functions .....	82-77
TRIM Procedures.....	82-79
WRITE Procedures.....	82-81
WRITEAPPEND Procedures .....	82-83

### 83 DBMS\_LOCK

<b>Using DBMS_LOCK</b> .....	83-2
Overview .....	83-3
Security Model.....	83-4
Constants .....	83-5
Rules and Limits.....	83-6
Operational Notes .....	83-7
<b>Summary of DBMS_LOCK Subprograms</b> .....	83-8
ALLOCATE_UNIQUE Procedure.....	83-9
CONVERT Function .....	83-11
RELEASE Function .....	83-12
REQUEST Function.....	83-13
SLEEP Procedure.....	83-14

### 84 DBMS\_LOGMNR

<b>Using DBMS_LOGMNR</b> .....	84-2
Overview .....	84-3
Security Model.....	84-4
Constants .....	84-5
Views.....	84-7
Operational Notes .....	84-8
<b>Summary of DBMS_LOGMNR Subprograms</b> .....	84-9
ADD_LOGFILE Procedure .....	84-10
COLUMN_PRESENT Function .....	84-12
END_LOGMNR Procedure .....	84-14
MINE_VALUE Function .....	84-15
REMOVE_LOGFILE Procedure .....	84-17
START_LOGMNR Procedure .....	84-18

### 85 DBMS\_LOGMNR\_D

<b>Using DBMS_LOGMNR_D</b> .....	85-2
Overview .....	85-3
Security Model.....	85-4
<b>Summary of DBMS_LOGMNR_D Subprograms</b> .....	85-5
BUILD Procedure .....	85-6
SET_TABLESPACE Procedure.....	85-9

---

## 86 DBMS\_LOGSTDBY

Using DBMS_LOGSTDBY .....	86-2
Overview .....	86-3
Security Model .....	86-4
<b>Summary of DBMS_LOGSTDBY Subprograms .....</b>	<b>86-5</b>
APPLY_SET Procedure .....	86-7
APPLY_UNSET Procedure .....	86-10
BUILD Procedure .....	86-11
INSTANTIATE_TABLE Procedure .....	86-12
IS_APPLY_SERVER Function .....	86-14
MAP_PRIMARY_SCN Function .....	86-15
PREPARE_FOR_NEW_PRIMARY Procedure .....	86-16
PURGE_SESSION Procedure .....	86-18
REBUILD Procedure .....	86-19
SET_TABLESPACE Procedure .....	86-20
SKIP Procedure .....	86-21
SKIP_ERROR Procedure .....	86-29
SKIP_TRANSACTION Procedure .....	86-33
UNSKIP Procedure .....	86-35
UNSKIP_ERROR Procedure .....	86-37
UNSKIP_TRANSACTION Procedure .....	86-39

## 87 DBMS\_METADATA

Using DBMS_METADATA .....	87-2
Overview .....	87-3
Security Model .....	87-4
Rules and Limits .....	87-5
<b>Data Structures - Object and Table Types .....</b>	<b>87-6</b>
<b>Subprogram Groupings .....</b>	<b>87-8</b>
Subprograms for Retrieving Multiple Objects From the Database .....	87-9
Subprograms for Submitting XML to the Database .....	87-10
<b>Summary of All DBMS_METADATA Subprograms .....</b>	<b>87-11</b>
ADD_TRANSFORM Function .....	87-12
CLOSE Procedure .....	87-16
CONVERT Functions and Procedures .....	87-17
FETCH_xxx Functions and Procedures .....	87-19
GET_xxx Functions .....	87-22
GET_QUERY Function .....	87-26
OPEN Function .....	87-27
OPENW Function .....	87-33
PUT Function .....	87-34
SET_COUNT Procedure .....	87-36
SET_FILTER Procedure .....	87-37
SET_PARSE_ITEM Procedure .....	87-47
SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures .....	87-50

---

## 88 DBMS\_METADATA\_DIFF

Using DBMS_METADATA_DIFF .....	88-2
Overview .....	88-3
Security Model.....	88-4
<b>Browsing APIs for Fetching and Comparing Objects.....</b>	<b>88-5</b>
<b>Summary of DBMS_METADATA_DIFF Subprograms.....</b>	<b>88-7</b>
OPENC Function.....	88-8
ADD_DOCUMENT Procedure.....	88-9
FETCH_CLOB Functions and Procedures .....	88-10
CLOSE Procedure .....	88-11

## 89 DBMS\_MGD\_ID\_UTL

Using DBMS_MGD_ID_UTL.....	89-2
Security Model.....	89-3
Constants .....	89-4
Exceptions .....	89-5
<b>Summary of DBMS_MGD_ID_UTL Subprograms .....</b>	<b>89-6</b>
ADD_SCHEME Procedure .....	89-7
CREATE_CATEGORY Function.....	89-11
EPC_TO_ORACLE_SCHEME Function .....	89-12
GET_CATEGORY_ID Function .....	89-15
GET_COMPONENTS Function .....	89-16
GET_ENCODINGS Function .....	89-18
GET_JAVA_LOGGING_LEVEL Function .....	89-19
GET_PLSQL_LOGGING_LEVEL Function .....	89-20
GET_SCHEME_NAMES Function .....	89-21
GET_TDT_XML Function .....	89-22
GET_VALIDATOR Function.....	89-24
REFRESH_CATEGORY Function.....	89-28
REMOVE_CATEGORY Procedure.....	89-29
REMOVE_PROXY Procedure.....	89-30
REMOVE_SCHEME Procedure .....	89-31
SET_JAVA_LOGGING_LEVEL Procedure .....	89-32
SET_PLSQL_LOGGING_LEVEL Procedure .....	89-33
SET_PROXY Procedure .....	89-34
VALIDATE_SCHEME Function .....	89-35

## 90 DBMS\_MGWADM

Using DBMS_MGWADM .....	90-2
Deprecated Subprograms .....	90-3
Constants .....	90-4
<b>Data Structures.....</b>	<b>90-7</b>
SYS.MGW_MQSERIES_PROPERTIES Object Type.....	90-8
SYS.MGW_PROPERTIES Object Type.....	90-10
SYS.MGW_PROPERTY Object Type.....	90-12
SYS.MGW_TIBRV_PROPERTIES Object Type.....	90-13



<b>Summary of DBMS_MGWADM Subprograms</b> .....	90-14
ADD_SUBSCRIBER Procedure .....	90-16
ALTER_AGENT Procedures .....	90-20
ALTER_JOB Procedure .....	90-22
ALTER_MSGSYSTEM_LINK Procedure for TIB/Rendezvous .....	90-24
ALTER_MSGSYSTEM_LINK Procedure for WebSphere MQ .....	90-25
ALTER_PROPAGATION_SCHEDULE Procedure .....	90-26
ALTER_SUBSCRIBER Procedure .....	90-27
CLEANUP_GATEWAY Procedures .....	90-29
CREATE_AGENT Procedure .....	90-32
CREATE_JOB Procedure.....	90-34
CREATE_MSGSYSTEM_LINK Procedures for TIB/Rendezvous .....	90-38
CREATE_MSGSYSTEM_LINK Procedures for WebSphere MQ.....	90-39
DB_CONNECT_INFO Procedure.....	90-40
DISABLE_JOB Procedure.....	90-41
DISABLE_PROPAGATION_SCHEDULE Procedure .....	90-42
ENABLE_JOB Procedure .....	90-43
ENABLE_PROPAGATION_SCHEDULE Procedure .....	90-44
REGISTER_FOREIGN_QUEUE Procedure.....	90-45
REMOVE_AGENT Procedure.....	90-46
REMOVE_JOB Procedure .....	90-47
REMOVE_MSGSYSTEM_LINK Procedure.....	90-48
REMOVE_OPTION Procedure .....	90-49
REMOVE_SUBSCRIBER Procedure .....	90-51
RESET_JOB Procedure .....	90-52
RESET_SUBSCRIBER Procedure .....	90-53
SCHEDULE_PROPAGATION Procedure .....	90-54
SET_LOG_LEVEL Procedures .....	90-56
SET_OPTION Procedure.....	90-57
SHUTDOWN Procedures .....	90-59
STARTUP Procedures .....	90-60
UNREGISTER_FOREIGN_QUEUE Procedure .....	90-61
UNSCHEDULE_PROPAGATION Procedure .....	90-62

## 91 DBMS\_MGWMSG

<b>Using DBMS_MGWMSG</b> .....	91-2
Security Model.....	91-3
Constants .....	91-4
Types .....	91-6
<b>Summary of DBMS_MGWMSG Subprograms</b> .....	91-21
LCR_TO_XML Function.....	91-22
NVARARRAY_ADD Procedure.....	91-23
NVARARRAY_FIND_NAME Function.....	91-24
NVARARRAY_FIND_NAME_TYPE Function.....	91-25
NVARARRAY_GET Function.....	91-26
NVARARRAY_GET_BOOLEAN Function .....	91-27
NVARARRAY_GET_BYTE Function.....	91-28

NVARRAY_GET_DATE Function .....	91-29
NVARRAY_GET_DOUBLE Function.....	91-30
NVARRAY_GET_FLOAT Function .....	91-31
NVARRAY_GET_INTEGER Function.....	91-32
NVARRAY_GET_LONG Function.....	91-33
NVARRAY_GET_RAW Function.....	91-34
NVARRAY_GET_SHORT Function.....	91-35
NVARRAY_GET_TEXT Function.....	91-36
XML_TO_LCR Function.....	91-37

## 92 DBMS\_MONITOR

<b>Summary of DBMS_MONITOR Subprograms .....</b>	<b>92-2</b>
CLIENT_ID_STAT_DISABLE Procedure .....	92-3
CLIENT_ID_STAT_ENABLE Procedure.....	92-4
CLIENT_ID_TRACE_DISABLE Procedure.....	92-5
CLIENT_ID_TRACE_ENABLE Procedure .....	92-6
DATABASE_TRACE_DISABLE Procedure .....	92-7
DATABASE_TRACE_ENABLE Procedure.....	92-8
SERV_MOD_ACT_STAT_DISABLE Procedure.....	92-9
SERV_MOD_ACT_STAT_ENABLE Procedure.....	92-10
SERV_MOD_ACT_TRACE_DISABLE Procedure .....	92-12
SERV_MOD_ACT_TRACE_ENABLE Procedure .....	92-13
SESSION_TRACE_DISABLE Procedure .....	92-15
SESSION_TRACE_ENABLE Procedure .....	92-16

## 93 DBMS\_MVIEW

<b>Using DBMS_MVIEW .....</b>	<b>93-2</b>
Operational Notes .....	93-3
Security Model.....	93-4
Rules and Limits.....	93-5
<b>Summary of DBMS_MVIEW Subprograms .....</b>	<b>93-6</b>
BEGIN_TABLE_REORGANIZATION Procedure .....	93-7
END_TABLE_REORGANIZATION Procedure.....	93-8
ESTIMATE_MVIEW_SIZE Procedure .....	93-9
EXPLAIN_MVIEW Procedure .....	93-10
EXPLAIN_REWRITE Procedure.....	93-11
I_AM_A_REFRESH Function.....	93-13
PMARKER Function .....	93-14
PURGE_DIRECT_LOAD_LOG Procedure .....	93-15
PURGE_LOG Procedure .....	93-16
PURGE_MVIEW_FROM_LOG Procedure.....	93-17
REFRESH Procedures .....	93-18
REFRESH_ALL_MVIEWS Procedure.....	93-20
REFRESH_DEPENDENT Procedures.....	93-21
REGISTER_MVIEW Procedure.....	93-23
UNREGISTER_MVIEW Procedure .....	93-25

<b>94</b>	<b>DBMS_NETWORK_ACL_ADMIN</b>	
	Using DBMS_NETWORK_ACL_ADMIN .....	94-2
	Examples .....	94-3
	<b>Summary of DBMS_NETWORK_ACL_ADMIN Subprograms</b> .....	94-5
	ADD_PRIVILEGE Procedure .....	94-6
	ASSIGN_ACL Procedure .....	94-8
	ASSIGN_WALLET_ACL Procedure .....	94-10
	CHECK_PRIVILEGE Function .....	94-11
	CHECK_PRIVILEGE_ACLID Function .....	94-12
	CREATE_ACL Procedure .....	94-13
	DELETE_PRIVILEGE Procedure .....	94-14
	DROP_ACL Procedure .....	94-15
	UNASSIGN_ACL Procedure .....	94-16
	UNASSIGN_WALLET_ACL Procedure .....	94-17
<b>95</b>	<b>DBMS_NETWORK_ACL_UTILITY</b>	
	Using DBMS_NETWORK_ACL_UTILITY .....	95-2
	Examples .....	95-3
	<b>Summary of DBMS_NETWORK_ACL_UTILITY Subprograms</b> .....	95-4
	CONTAINS_HOST Function .....	95-5
	DOMAIN_LEVEL Function .....	95-6
	DOMAINS Function .....	95-7
	EQUALS_HOST Function .....	95-8
<b>96</b>	<b>DBMS_OBFUSCATION_TOOLKIT</b>	
	Using DBMS_OBFUSCATION_TOOLKIT .....	96-2
	Overview .....	96-3
	Security Model .....	96-4
	Operational Notes .....	96-5
	<b>Summary of DBMS_OBFUSCATION Subprograms</b> .....	96-7
	DES3DECRYPT Procedures and Functions .....	96-8
	DES3ENCRYPT Procedures and Functions .....	96-10
	DES3GETKEY Procedures and Functions .....	96-12
	DESDECRYPT Procedures and Functions .....	96-13
	DESENCRYPT Procedures and Functions .....	96-15
	DESGETKEY Procedures and Functions .....	96-17
	MD5 Procedures and Functions .....	96-18
<b>97</b>	<b>DBMS_ODCI</b>	
	<b>Summary of DBMS_ODCI Subprograms</b> .....	97-2
	ESTIMATE_CPU_UNITS Function .....	97-3
<b>98</b>	<b>DBMS_OFFLINE_OG</b>	
	Documentation of DBMS_OFFLINE_OG .....	98-2

---

## 99 DBMS\_OUTLN

Using DBMS_OUTLN.....	99-2
Overview .....	99-3
Security Model.....	99-4
<b>Summary of DBMS_OUTLN Subprograms .....</b>	<b>99-5</b>
CLEAR_USED Procedure .....	99-6
CREATE_OUTLINE Procedure .....	99-7
DROP_BY_CAT Procedure.....	99-8
DROP_UNUSED Procedure.....	99-9
EXACT_TEXT_SIGNATURES Procedure .....	99-10
UPDATE_BY_CAT Procedure .....	99-11
UPDATE_SIGNATURES Procedure.....	99-12

## 100 DBMS\_OUTPUT

Using DBMS_OUTPUT .....	100-2
Overview .....	100-3
Security Model.....	100-4
Operational Notes .....	100-5
Exceptions .....	100-6
Rules and Limits.....	100-7
Examples .....	100-8
<b>Data Structures.....</b>	<b>100-11</b>
CHARARR Table Type.....	100-12
DBMSOUTPUT_LINESARRAY Object Type .....	100-13
<b>Summary of DBMS_OUTPUT Subprograms .....</b>	<b>100-14</b>
DISABLE Procedure .....	100-15
ENABLE Procedure .....	100-16
GET_LINE Procedure .....	100-17
GET_LINES Procedure.....	100-18
NEW_LINE Procedure .....	100-19
PUT Procedure .....	100-20
PUT_LINE Procedure .....	100-21

## 101 DBMS\_PARALLEL\_EXECUTE

Using DBMS_PARALLEL_EXECUTE.....	101-2
Overview .....	101-3
Security Model.....	101-4
Constants .....	101-5
Views.....	101-6
Exceptions .....	101-7
Examples .....	101-8
<b>Summary of DBMS_PARALLEL_EXECUTE Subprograms.....</b>	<b>101-11</b>
ADM_DROP_CHUNKS Procedure.....	101-12
ADM_DROP_TASK Procedure.....	101-13
ADM_TASK_STATUS Procedure.....	101-14
ADM_STOP_TASK Procedure.....	101-15

CREATE_TASK Procedure.....	101-16
CREATE_CHUNKS_BY_NUMBER_COL Procedure.....	101-17
CREATE_CHUNKS_BY_ROWID Procedure.....	101-18
CREATE_CHUNKS_BY_SQL Procedure.....	101-19
DROP_TASK Procedure.....	101-20
DROP_CHUNKS Procedure.....	101-21
GENERATE_TASK_NAME Function.....	101-22
GET_NUMBER_COL_CHUNK Procedure.....	101-23
GET_ROWID_CHUNK Procedure.....	101-24
PURGE_PROCESSED_CHUNKS Procedure.....	101-25
RESUME_TASK Procedures.....	101-26
RUN_TASK Procedure.....	101-28
SET_CHUNK_STATUS Procedure.....	101-30
STOP_TASK Procedure.....	101-31
TASK_STATUS Procedure.....	101-32

## 102 DBMS\_PCLXUTIL

<b>Using DBMS_PCLXUTIL</b> .....	102-2
Overview .....	102-3
Security Model.....	102-4
Operational Notes .....	102-5
Rules and Limits.....	102-6
<b>Summary of DBMS_PCLXUTIL Subprograms</b> .....	102-7
BUILD_PART_INDEX Procedure .....	102-8

## 103 DBMS\_PIPE

<b>Using DBMS_PIPE</b> .....	103-2
Overview .....	103-3
Security Model.....	103-4
Constants .....	103-5
Operational Notes .....	103-6
Exceptions .....	103-8
Examples .....	103-9
<b>Summary of DBMS_PIPE Subprograms</b> .....	103-18
CREATE_PIPE Function .....	103-19
NEXT_ITEM_TYPE Function .....	103-21
PACK_MESSAGE Procedures .....	103-22
PURGE Procedure.....	103-24
RECEIVE_MESSAGE Function.....	103-25
RESET_BUFFER Procedure .....	103-27
REMOVE_PIPE Function .....	103-28
SEND_MESSAGE Function .....	103-29
UNIQUE_SESSION_NAME Function .....	103-31
UNPACK_MESSAGE Procedures .....	103-32

---

## 104 DBMS\_PREDICTIVE\_ANALYTICS

Using DBMS_PREDICTIVE_ANALYTICS .....	104-2
Overview .....	104-3
Security Model.....	104-4
<b>Summary of DBMS_PREDICTIVE_ANALYTICS Subprograms</b> .....	104-5
EXPLAIN Procedure.....	104-6
PREDICT Procedure .....	104-8
PROFILE Procedure.....	104-10

## 105 DBMS\_PREPROCESSOR

Using DBMS_PREPROCESSOR.....	105-2
Overview .....	105-3
Operating Notes .....	105-4
<b>Data Structures</b> .....	105-5
SOURCE_LINES_T Table Type.....	105-6
<b>Summary of DBMS_PREPROCESSOR Subprograms</b> .....	105-7
GET_POST_PROCESSED_SOURCE Functions.....	105-8
PRINT_POST_PROCESSED_SOURCE Procedures.....	105-10

## 106 DBMS\_PROFILER

Using DBMS_PROFILER .....	106-2
Overview .....	106-3
Security Model.....	106-5
Operational Notes .....	106-6
Exceptions .....	106-8
<b>Summary of DBMS_PROFILER Subprograms</b> .....	106-9
FLUSH_DATA Function and Procedure .....	106-10
GET_VERSION Procedure.....	106-11
INTERNAL_VERSION_CHECK Function.....	106-12
PAUSE_PROFILER Function and Procedure .....	106-13
RESUME_PROFILER Function and Procedure .....	106-14
START_PROFILER Functions and Procedures .....	106-15
STOP_PROFILER Function and Procedure .....	106-16

## 107 DBMS\_PROPAGATION\_ADM

Using DBMS_PROPAGATION_ADM .....	107-2
Overview .....	107-3
Security Model.....	107-4
<b>Summary of DBMS_PROPAGATION_ADM Subprograms</b> .....	107-5
ALTER_PROPAGATION Procedure .....	107-6
CREATE_PROPAGATION Procedure .....	107-8
DROP_PROPAGATION Procedure .....	107-11
START_PROPAGATION Procedure .....	107-13
STOP_PROPAGATION Procedure .....	107-14

---

## 108 DBMS\_RANDOM

Using DBMS_RANDOM.....	108-2
Deprecated Subprograms .....	108-3
Security Model.....	108-4
Operational Notes .....	108-5
<b>Summary of DBMS_RANDOM Subprograms .....</b>	<b>108-6</b>
INITIALIZE Procedure.....	108-7
NORMAL Function .....	108-8
RANDOM Procedure .....	108-9
SEED Procedures.....	108-10
STRING Function .....	108-11
TERMINATE Procedure .....	108-12
VALUE Functions .....	108-13

## 109 DBMS\_RECTIFIER\_DIFF

Documentation of DBMS_RECTIFIER_DIFF .....	109-2
--	-------

## 110 DBMS\_REDEFINITION

Using DBMS_REDEFINITION.....	110-2
Overview .....	110-3
Constants .....	110-4
Operational Notes .....	110-5
Rules and Limits.....	110-6
<b>Summary of DBMS_REDEFINITION Subprograms.....</b>	<b>110-7</b>
ABORT_REDEF_TABLE Procedure.....	110-8
CAN_REDEF_TABLE Procedure .....	110-9
COPY_TABLE_DEPENDENTS Procedure .....	110-10
FINISH_REDEF_TABLE Procedure .....	110-12
REGISTER_DEPENDENT_OBJECT Procedure.....	110-13
START_REDEF_TABLE Procedure.....	110-14
SYNC_INTERIM_TABLE Procedure .....	110-15
UNREGISTER_DEPENDENT_OBJECT Procedure .....	110-16

## 111 DBMS\_REFRESH

Documentation of DBMS_REFRESH.....	111-2
------------------------------------	-------

## 112 DBMS\_REPAIR

Using DBMS_REPAIR.....	112-2
Overview .....	112-3
Security Model.....	112-4
Constants .....	112-5
Operating Notes .....	112-6
Exceptions .....	112-7
Examples .....	112-8
<b>Summary of DBMS_REPAIR Subprograms .....</b>	<b>112-9</b>

ADMIN_TABLES Procedure.....	112-10
CHECK_OBJECT Procedure .....	112-11
DUMP_ORPHAN_KEYS Procedure.....	112-13
FIX_CORRUPT_BLOCKS Procedure .....	112-14
ONLINE_INDEX_CLEAN Function.....	112-15
REBUILD_FREELISTS Procedure.....	112-16
SEGMENT_FIX_STATUS Procedure .....	112-17
SKIP_CORRUPT_BLOCKS Procedure.....	112-18
<b>113 DBMS_REPCAT</b>	
Documentation of DBMS_REPCAT .....	113-2
<b>114 DBMS_REPCAT_ADMIN</b>	
Documentation of DBMS_REPCAT_ADMIN .....	114-2
<b>115 DBMS_REPCAT_INSTANTIATE</b>	
Documentation of DBMS_REPCAT_INSTANTIATE.....	115-2
<b>116 DBMS_REPCAT_RGT</b>	
Documentation of DBMS_REPCAT_RGT .....	116-2
<b>117 DBMS_REPUTIL</b>	
Documentation of DBMS_REPUTIL.....	117-2
<b>118 DBMS_RESCONFIG</b>	
Using DBMS_RESCONFIG .....	118-2
Overview .....	118-3
Summary of DBMS_RESCONFIG Subprograms.....	118-4
ADDREPOSITORYRESCONFIG Procedure.....	118-5
ADDRESCONFIG Procedure.....	118-6
APPENDRESCONFIG Procedure.....	118-7
DELETEREPOSITORYRESCONFIG Procedure.....	118-8
DELETERESCONFIG Procedures .....	118-9
GETLISTENERS Function.....	118-10
GETREPOSITORYRESCONFIG Function.....	118-11
GETREPOSITORYRESCONFIGPATHS Function .....	118-12
GETRESCONFIG Function.....	118-13
GETRESCONFIGPATHS Function .....	118-14
PATCHREPOSITORYRESCONFIGLIST Procedure.....	118-15
<b>119 DBMS_RESOURCE_MANAGER</b>	
Using DBMS_RESOURCE_MANAGER.....	119-2
Deprecated Subprograms .....	119-3
Security Model.....	119-4



Constants .....	119-5
<b>Summary of DBMS_RESOURCE_MANAGER Subprograms</b> .....	119-7
BEGIN_SQL_BLOCK Procedure .....	119-9
CALIBRATE_IO Procedure .....	119-10
CLEAR_PENDING_AREA Procedure .....	119-13
CREATE_CATEGORY Procedure .....	119-14
CREATE_CONSUMER_GROUP Procedure .....	119-15
CREATE_PENDING_AREA Procedure .....	119-16
CREATE_PLAN Procedure .....	119-18
CREATE_PLAN_DIRECTIVE Procedure .....	119-19
CREATE_SIMPLE_PLAN Procedure .....	119-23
DELETE_CATEGORY Procedure .....	119-25
DELETE_CONSUMER_GROUP Procedure .....	119-26
DELETE_PLAN Procedure .....	119-27
DELETE_PLAN_CASCADE Procedure .....	119-28
DELETE_PLAN_DIRECTIVE Procedure .....	119-29
END_SQL_BLOCK Procedure .....	119-30
SET_CONSUMER_GROUP_MAPPING Procedure .....	119-31
SET_CONSUMER_GROUP_MAPPING_PRI Procedure .....	119-32
SET_INITIAL_CONSUMER_GROUP Procedure .....	119-33
SUBMIT_PENDING_AREA Procedure .....	119-34
SWITCH_CONSUMER_GROUP_FOR_SESS Procedure .....	119-35
SWITCH_CONSUMER_GROUP_FOR_USER Procedure .....	119-36
SWITCH_PLAN Procedure .....	119-37
UPDATE_CATEGORY Procedure .....	119-38
UPDATE_CONSUMER_GROUP Procedure .....	119-39
UPDATE_PLAN Procedure .....	119-40
UPDATE_PLAN_DIRECTIVE Procedure .....	119-41
VALIDATE_PENDING_AREA Procedure .....	119-45

## 120 DBMS\_RESOURCE\_MANAGER\_PRIVS

<b>Summary of DBMS_RESOURCE_MANAGER_PRIVS Subprograms</b> .....	120-2
GRANT_SWITCH_CONSUMER_GROUP Procedure .....	120-3
GRANT_SYSTEM_PRIVILEGE Procedure .....	120-4
REVOKE_SWITCH_CONSUMER_GROUP Procedure .....	120-5
REVOKE_SYSTEM_PRIVILEGE Procedure .....	120-6

## 121 DBMS\_RESULT\_CACHE

<b>Using DBMS_RESULT_CACHE</b> .....	121-2
Constants .....	121-3
<b>Summary of DBMS_RESULT_CACHE Subprograms</b> .....	121-4
BYPASS Procedure .....	121-5
FLUSH Function & Procedure .....	121-7
INVALIDATE Functions & Procedures .....	121-8
INVALIDATE_OBJECT Functions & Procedures .....	121-9
MEMORY_REPORT Procedure .....	121-10
STATUS Function .....	121-11

---

## 122 DBMS\_RESUMABLE

Using DBMS_RESUMABLE .....	122-2
Operational Notes .....	122-3
Summary of DBMS_RESUMABLE Subprograms .....	122-4
ABORT Procedure .....	122-5
GET_SESSION_TIMEOUT Function .....	122-6
GET_TIMEOUT Function .....	122-7
SET_SESSION_TIMEOUT Procedure .....	122-8
SET_TIMEOUT Procedure .....	122-9
SPACE_ERROR_INFO Function .....	122-10

## 123 DBMS\_RLMGR

Using DBMS_RLMGR .....	123-2
Security Model .....	123-3
Summary of Rules Manager Subprograms .....	123-4
ADD_ELEMENTARY_ATTRIBUTE Procedures .....	123-6
ADD_EVENT Procedure .....	123-9
ADD_FUNCTIONS Procedure .....	123-11
ADD_RULE Procedure .....	123-12
CONDITION_REF Function .....	123-14
CONSUME_EVENT Function .....	123-15
CONSUME_PRIM_EVENTS Function .....	123-17
CREATE_CONDITIONS_TABLE Procedure .....	123-19
CREATE_EVENT_STRUCT Procedure .....	123-21
CREATE_EXPFIL_INDEXES Procedure .....	123-22
CREATE_INTERFACE Procedure .....	123-24
CREATE_RULE_CLASS Procedure .....	123-26
DELETE_RULE Procedure .....	123-30
DROP_CONDITIONS_TABLE Procedure .....	123-31
DROP_EVENT_STRUCT Procedure .....	123-32
DROP_EXPFIL_INDEXES Procedure .....	123-33
DROP_INTERFACE Procedure .....	123-34
DROP_RULE_CLASS Procedure .....	123-35
EXTEND_EVENT_STRUCT Procedure .....	123-36
GET_AGGREGATE_VALUE Function .....	123-37
GRANT_PRIVILEGE Procedure .....	123-39
PROCESS_RULES Procedure .....	123-40
PURGE_EVENTS Procedure .....	123-42
RESET_SESSION Procedure .....	123-43
REVOKE_PRIVILEGE Procedure .....	123-44
SYNC_TEXT_INDEXES Procedure .....	123-45

## 124 DBMS\_RLS

Using DBMS_RLS .....	124-2
Overview .....	124-3
Security Model .....	124-4

Operational Notes .....	124-5
<b>Summary of DBMS_RLS Subprograms .....</b>	<b>124-6</b>
ADD_GROUPED_POLICY Procedure .....	124-7
ADD_POLICY Procedure .....	124-9
ADD_POLICY_CONTEXT Procedure .....	124-13
CREATE_POLICY_GROUP Procedure .....	124-14
DELETE_POLICY_GROUP Procedure .....	124-15
DISABLE_GROUPED_POLICY Procedure .....	124-16
DROP_GROUPED_POLICY Procedure .....	124-17
DROP_POLICY Procedure .....	124-18
DROP_POLICY_CONTEXT Procedure .....	124-19
ENABLE_GROUPED_POLICY Procedure .....	124-20
ENABLE_POLICY Procedure .....	124-21
REFRESH_GROUPED_POLICY Procedure .....	124-22
REFRESH_POLICY Procedure .....	124-23
<b>125 DBMS_ROWID</b>	
<b>Using DBMS_ROWID .....</b>	<b>125-2</b>
Security Model .....	125-3
Types .....	125-4
Exceptions .....	125-6
Operational Notes .....	125-7
Examples .....	125-8
<b>Summary of DBMS_ROWID Subprograms .....</b>	<b>125-9</b>
ROWID_BLOCK_NUMBER Function .....	125-10
ROWID_CREATE Function .....	125-11
ROWID_INFO Procedure .....	125-12
ROWID_OBJECT Function .....	125-13
ROWID_RELATIVE_FNO Function .....	125-14
ROWID_ROW_NUMBER Function .....	125-15
ROWID_TO_ABSOLUTE_FNO Function .....	125-16
ROWID_TO_EXTENDED Function .....	125-17
ROWID_TO_RESTRICTED Function .....	125-19
ROWID_TYPE Function .....	125-20
ROWID_VERIFY Function .....	125-21
<b>126 DBMS_RULE</b>	
<b>Using DBMS_RULE .....</b>	<b>126-2</b>
Overview .....	126-3
Security Model .....	126-4
<b>Summary of DBMS_RULE Subprograms .....</b>	<b>126-5</b>
CLOSE_ITERATOR Procedure .....	126-6
EVALUATE Procedures .....	126-7
GET_NEXT_HIT Function .....	126-11

---

## 127 DBMS\_RULE\_ADM

<b>Using DBMS_RULE_ADM</b> .....	127-2
Overview .....	127-3
Security Model.....	127-4
<b>Summary of DBMS_RULE_ADM Subprograms</b> .....	127-5
ADD_RULE Procedure .....	127-6
ALTER_EVALUATION_CONTEXT Procedure.....	127-8
ALTER_RULE Procedure.....	127-11
CREATE_EVALUATION_CONTEXT Procedure.....	127-13
CREATE_RULE Procedure.....	127-15
CREATE_RULE_SET Procedure.....	127-17
DROP_EVALUATION_CONTEXT Procedure .....	127-18
DROP_RULE Procedure .....	127-19
DROP_RULE_SET Procedure .....	127-20
GRANT_OBJECT_PRIVILEGE Procedure .....	127-21
GRANT_SYSTEM_PRIVILEGE Procedure .....	127-23
REMOVE_RULE Procedure .....	127-25
REVOKE_OBJECT_PRIVILEGE Procedure .....	127-27
REVOKE_SYSTEM_PRIVILEGE Procedure .....	127-28

## 128 DBMS\_SCHEDULER

<b>Data Structures</b> .....	128-2
JOBARG Object Type.....	128-3
JOB Object Type .....	128-5
JOB_DEFINITION Object Type .....	128-6
JOBATTR Object Type.....	128-10
SCHEDULER\$_STEP_TYPE Object Type.....	128-12
SCHEDULER\$_EVENT_INFO Object Type .....	128-13
SCHEDULER_FILEWATCHER_RESULT Object Type .....	128-15
SCHEDULER_FILEWATCHER_REQUEST Object Type .....	128-16
<b>Using DBMS_SCHEDULER</b> .....	128-17
Security Model.....	128-18
Rules and Limits .....	128-19
Operational Notes .....	128-20
<b>Summary of DBMS_SCHEDULER Subprograms</b> .....	128-30
ADD_EVENT_QUEUE_SUBSCRIBER Procedure .....	128-34
ADD_GROUP_MEMBER Procedure .....	128-35
ADD_JOB_EMAIL_NOTIFICATION Procedure .....	128-36
ADD_WINDOW_GROUP_MEMBER Procedure .....	128-39
ALTER_CHAIN Procedure .....	128-40
ALTER_RUNNING_CHAIN Procedure .....	128-43
CLOSE_WINDOW Procedure.....	128-45
COPY_JOB Procedure .....	128-46
CREATE_CHAIN Procedure.....	128-47
CREATE_CREDENTIAL Procedure .....	128-49
CREATE_DATABASE_DESTINATION Procedure .....	128-50
CREATE_EVENT_SCHEDULE Procedure .....	128-51

---

CREATE_FILE_WATCHER Procedure .....	128-53
CREATE_GROUP Procedure .....	128-55
CREATE_JOB Procedure.....	128-57
CREATE_JOB_CLASS Procedure .....	128-64
CREATE_JOBS Procedure .....	128-66
CREATE_PROGRAM Procedure.....	128-67
CREATE_SCHEDULE Procedure.....	128-70
CREATE_WINDOW Procedure.....	128-71
CREATE_WINDOW_GROUP Procedure .....	128-74
DEFINE_ANYDATA_ARGUMENT Procedure.....	128-75
DEFINE_CHAIN_EVENT_STEP Procedure.....	128-76
DEFINE_CHAIN_RULE Procedure .....	128-77
DEFINE_CHAIN_STEP Procedure .....	128-80
DEFINE_METADATA_ARGUMENT Procedure .....	128-81
DEFINE_PROGRAM_ARGUMENT Procedure.....	128-83
DISABLE Procedure .....	128-85
DROP_AGENT_DESTINATION Procedure.....	128-88
DROP_CHAIN Procedure .....	128-89
DROP_CHAIN_RULE Procedure .....	128-90
DROP_CHAIN_STEP Procedure.....	128-91
DROP_CREDENTIAL Procedure.....	128-92
DROP_DATABASE_DESTINATION Procedure .....	128-93
DROP_FILE_WATCHER Procedure.....	128-94
DROP_GROUP Procedure.....	128-95
DROP_JOB Procedure .....	128-96
DROP_JOB_CLASS Procedure.....	128-97
DROP_PROGRAM Procedure .....	128-98
DROP_PROGRAM_ARGUMENT Procedure .....	128-99
DROP_SCHEDULE Procedure .....	128-100
DROP_WINDOW Procedure .....	128-101
DROP_WINDOW_GROUP Procedure.....	128-102
ENABLE Procedure .....	128-103
END_DETACHED_JOB_RUN Procedure.....	128-105
EVALUATE_CALENDAR_STRING Procedure .....	128-106
EVALUATE_RUNNING_CHAIN Procedure .....	128-108
GENERATE_JOB_NAME Function .....	128-109
GET_AGENT_INFO Function.....	128-110
GET_AGENT_VERSION Function.....	128-111
GET_ATTRIBUTE Procedure.....	128-112
GET_FILE Procedure .....	128-113
GET_SCHEDULER_ATTRIBUTE Procedure.....	128-115
OPEN_WINDOW Procedure .....	128-116
PURGE_LOG Procedure .....	128-118
PUT_FILE Procedure .....	128-119
REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure .....	128-120
REMOVE_GROUP_MEMBER Procedure .....	128-121
REMOVE_JOB_EMAIL_NOTIFICATION Procedure .....	128-122

REMOVE_WINDOW_GROUP_MEMBER Procedure .....	128-123
RESET_JOB_ARGUMENT_VALUE Procedure .....	128-124
RUN_CHAIN Procedure .....	128-125
RUN_JOB Procedure .....	128-127
SET_AGENT_REGISTRATION_PASS Procedure .....	128-129
SET_ATTRIBUTE Procedure .....	128-130
SET_ATTRIBUTE_NULL Procedure.....	128-145
SET_JOB_ANYDATA_VALUE Procedure.....	128-146
SET_JOB_ARGUMENT_VALUE Procedure.....	128-147
SET_JOB_ATTRIBUTES Procedure.....	128-149
SET_SCHEDULER_ATTRIBUTE Procedure .....	128-150
STOP_JOB Procedure .....	128-153

## 129 DBMS\_SERVER\_ALERT

<b>Using DBMS_SERVER_ALERT</b> .....	129-2
Security Model.....	129-3
Object Types.....	129-4
Relational Operators.....	129-5
Supported Metrics.....	129-6
<b>Summary of DBMS_SERVER_ALERT Subprograms</b> .....	129-12
EXPAND_MESSAGE Function.....	129-13
GET_THRESHOLD Procedure .....	129-14
SET_THRESHOLD Procedure .....	129-15

## 130 DBMS\_SERVICE

<b>Using DBMS_SERVICE</b> .....	130-2
Overview .....	130-3
Security Model.....	130-4
Deprecated Subprograms .....	130-5
Constants .....	130-6
Exceptions .....	130-8
<b>Summary of DBMS_SERVICE Subprograms</b> .....	130-9
CREATE_SERVICE Procedure.....	130-10
DELETE_SERVICE Procedure .....	130-12
DISCONNECT_SESSION Procedure.....	130-13
MODIFY_SERVICE Procedure .....	130-14
START_SERVICE Procedure.....	130-16
STOP_SERVICE Procedure.....	130-17

## 131 DBMS\_SESSION

<b>Using DBMS_SESSION</b> .....	131-2
Security Model.....	131-3
Operational Notes .....	131-4
<b>Data Structures</b> .....	131-5
INTEGER_ARRAY Table Type.....	131-6
LNAME_ARRAY Table Type.....	131-7

<b>Summary of DBMS_SESSION Subprograms</b> .....	131-8
CLEAR_ALL_CONTEXT Procedure.....	131-9
CLEAR_CONTEXT Procedure.....	131-10
CLEAR_IDENTIFIER Procedure .....	131-11
CLOSE_DATABASE_LINK Procedure.....	131-12
FREE_UNUSED_USER_MEMORY Procedure.....	131-13
GET_PACKAGE_MEMORY_UTILIZATION Procedure .....	131-15
IS_ROLE_ENABLED Function .....	131-16
IS_SESSION_ALIVE Function.....	131-17
LIST_CONTEXT Procedures .....	131-18
MODIFY_PACKAGE_STATE Procedure.....	131-19
SESSION_TRACE_DISABLE Procedure .....	131-23
SESSION_TRACE_ENABLE Procedure .....	131-24
RESET_PACKAGE Procedure .....	131-25
SET_CONTEXT Procedure .....	131-27
SET_EDITION_DEFERRED Procedure .....	131-29
SET_IDENTIFIER Procedure.....	131-30
SET-NLS Procedure.....	131-31
SET_ROLE Procedure.....	131-32
SET_SQL_TRACE Procedure .....	131-33
SWITCH_CURRENT_CONSUMER_GROUP Procedure .....	131-34
UNIQUE_SESSION_ID Function .....	131-36

## 132 DBMS\_SHARED\_POOL

<b>Using DBMS_SHARED_POOL</b> .....	132-2
Overview .....	132-3
Operational Notes .....	132-4
<b>Summary of DBMS_SHARED_POOL Subprograms</b> .....	132-5
ABORTED_REQUEST_THRESHOLD Procedure.....	132-6
KEEP Procedure .....	132-7
MARKHOT Procedure .....	132-9
PURGE Procedure.....	132-10
SIZES Procedure.....	132-12
UNKEEP Procedure.....	132-13
UNMARKHOT Procedure.....	132-14

## 133 DBMS\_SPACE

<b>Using DBMS_SPACE</b> .....	133-2
Security Model.....	133-3
<b>Data Structures</b> .....	133-4
CREATE_TABLE_COST_COLINFO Object Type.....	133-5
ASA_RECO_ROW Record Type.....	133-6
ASA_RECO_ROW_TB Table Type.....	133-7
<b>Summary of DBMS_SPACE Subprograms</b> .....	133-8
ASA_RECOMMENDATIONS Function.....	133-9
CREATE_INDEX_COST Procedure .....	133-10
CREATE_TABLE_COST Procedures .....	133-11

FREE_BLOCKS Procedure.....	133-13
ISDATAFILEDROPPABLE_NAME Procedure.....	133-15
OBJECT_DEPENDENT_SEGMENTS Function.....	133-16
OBJECT_GROWTH_TREND Function.....	133-18
SPACE_USAGE Procedures.....	133-20
UNUSED_SPACE Procedure.....	133-23

## 134 DBMS\_SPACE\_ADMIN

<b>Using DBMS_SPACE_ADMIN</b> .....	134-2
Security Model.....	134-3
Constants.....	134-4
Operational Notes.....	134-6
<b>Summary of DBMS_SPACE_ADMIN Subprograms</b> .....	134-7
ASSM_SEGMENT_VERIFY Procedure.....	134-8
ASSM_TABLESPACE_VERIFY Procedure.....	134-10
DROP_EMPTY_SEGMENTS Procedure.....	134-11
MATERIALIZED_DEFERRED_SEGMENTS Procedure.....	134-12
SEGMENT_CORRUPT Procedure.....	134-13
SEGMENT_DROP_CORRUPT Procedure.....	134-14
SEGMENT_DUMP Procedure.....	134-15
SEGMENT_VERIFY Procedure.....	134-16
TABLESPACE_FIX_BITMAPS Procedure.....	134-17
TABLESPACE_FIX_SEGMENT_STATES Procedure.....	134-18
TABLESPACE_MIGRATE_FROM_LOCAL Procedure.....	134-19
TABLESPACE_MIGRATE_TO_LOCAL Procedure.....	134-20
TABLESPACE_REBUILD_BITMAPS Procedure.....	134-21
TABLESPACE_REBUILD_QUOTAS Procedure.....	134-22
TABLESPACE_RELOCATE_BITMAPS Procedure.....	134-23
TABLESPACE_VERIFY Procedure.....	134-24

## 135 DBMS\_SPM

<b>Using DBMS_SPM</b> .....	135-2
Overview.....	135-3
Security Model.....	135-4
Constants.....	135-5
Examples.....	135-6
<b>Data Structures</b> .....	135-7
NAMELIST Table Type.....	135-8
<b>Summary of DBMS_SPM Subprograms</b> .....	135-9
ALTER_SQL_PLAN_BASELINE Function.....	135-10
CONFIGURE Procedure.....	135-12
CREATE_STGTAB_BASELINE Procedure.....	135-13
DROP_SQL_PLAN_BASELINE Function.....	135-14
EVOLVE_SQL_PLAN_BASELINE Function.....	135-15
LOAD_PLANS_FROM_CURSOR_CACHE Functions.....	135-17
LOAD_PLANS_FROM_SQLSET Function.....	135-19
MIGRATE_STORED_OUTLINE Functions.....	135-20



PACK_STGTAB_BASELINE Function .....	135-22
UNPACK_STGTAB_BASELINE Function .....	135-23

## 136 DBMS\_SQL

<b>Using DBMS_SQL</b> .....	136-2
Overview .....	136-3
Security Model.....	136-4
Constants .....	136-5
Exceptions .....	136-6
Operational Notes .....	136-7
Examples .....	136-12
<b>Data Structures</b> .....	136-25
DESC_REC Record Type.....	136-26
DESC_REC2 Record Type.....	136-27
DESC_REC3 Record Type.....	136-28
BFILE_TABLE Table Type .....	136-29
BINARY_DOUBLE_TABLE Table Type.....	136-30
BINARY_FLOAT_TABLE Table Type .....	136-31
BLOB_TABLE Table Type.....	136-32
CLOB_TABLE Table Type .....	136-33
DATE_TABLE Table Type.....	136-34
DESC_TAB Table Type .....	136-35
DESC_TAB2 Table Type .....	136-36
DESC_TAB3 Table Type .....	136-37
INTERVAL_DAY_TO_SECOND_TABLE Table Type .....	136-38
INTERVAL_YEAR_TO_MONTH_TABLE Table Type.....	136-39
NUMBER_TABLE Table Type .....	136-40
TIME_TABLE Table Type .....	136-41
TIME_WITH_TIME_ZONE_TABLE Table Type .....	136-42
TIMESTAMP_TABLE Table Type .....	136-43
TIMESTAMP_WITH_LTZ_TABLE Table Type .....	136-44
TIMESTAMP_WITH_TIME_ZONE_TABLE Table Type .....	136-45
UROWID_TABLE Table Type.....	136-46
VARCHAR2_TABLE Table Type .....	136-47
VARCHAR2A Table Type .....	136-48
VARCHAR2S Table Type .....	136-49
<b>Summary of DBMS_SQL Subprograms</b> .....	136-50
BIND_ARRAY Procedures .....	136-52
BIND_VARIABLE Procedures .....	136-55
CLOSE_CURSOR Procedure .....	136-57
COLUMN_VALUE Procedure.....	136-58
COLUMN_VALUE_LONG Procedure.....	136-61
DEFINE_ARRAY Procedure .....	136-62
DEFINE_COLUMN Procedures .....	136-65
DEFINE_COLUMN_CHAR Procedure.....	136-67
DEFINE_COLUMN_LONG Procedure .....	136-68
DEFINE_COLUMN_RAW Procedure .....	136-69

DEFINE_COLUMN_ROWID Procedure.....	136-70
DESCRIBE_COLUMNS Procedure .....	136-71
DESCRIBE_COLUMNS2 Procedure .....	136-72
DESCRIBE_COLUMNS3 Procedure .....	136-73
EXECUTE Function.....	136-75
EXECUTE_AND_FETCH Function.....	136-76
FETCH_ROWS Function.....	136-77
IS_OPEN Function .....	136-78
LAST_ERROR_POSITION Function.....	136-79
LAST_ROW_COUNT Function .....	136-80
LAST_ROW_ID Function.....	136-81
LAST_SQL_FUNCTION_CODE Function.....	136-82
OPEN_CURSOR Function .....	136-83
PARSE Procedures .....	136-84
TO_CURSOR_NUMBER Function .....	136-87
TO_REFCURSOR Function .....	136-89
VARIABLE_VALUE Procedures .....	136-91

## 137 DBMS\_SQLDIAG

<b>Using DBMS_SQLDIAG</b> .....	137-2
Overview .....	137-3
Constants .....	137-5
Examples .....	137-8
<b>Summary of DBMS_SQLDIAG Subprograms</b> .....	137-9
ACCEPT_SQL_PATCH Function & Procedure.....	137-11
ALTER_SQL_PATCH Procedure .....	137-13
CANCEL_DIAGNOSIS_TASK Procedure .....	137-14
CREATE_DIAGNOSIS_TASK Functions .....	137-15
CREATE_STGTAB_SQLPATCH Procedure .....	137-17
DROP_DIAGNOSIS_TASK Procedure .....	137-18
DROP_SQL_PATCH Procedure .....	137-19
EXECUTE_DIAGNOSIS_TASK Procedure .....	137-20
EXPLAIN_SQL_TESTCASE Function.....	137-21
EXPORT_SQL_TESTCASE Procedures .....	137-22
EXPORT_SQL_TESTCASE_DIR_BY_INC Function.....	137-25
EXPORT_SQL_TESTCASE_DIR_BY_TXT Function.....	137-26
GET_FIX_CONTROL Function.....	137-27
GET_SQL Function .....	137-28
IMPORT_SQL_TESTCASE Procedures .....	137-29
INCIDENTID_2_SQL Procedure .....	137-31
INTERRUPT_DIAGNOSIS_TASK Procedure .....	137-32
LOAD_SQLSET_FROM_TCB Function.....	137-33
PACK_STGTAB_SQLPATCH Procedure.....	137-34
REPORT_DIAGNOSIS_TASK Function.....	137-35
RESET_DIAGNOSIS_TASK Procedure .....	137-36
RESUME_DIAGNOSIS_TASK Procedure.....	137-37
SET_DIAGNOSIS_TASK_PARAMETER Procedure .....	137-38

UNPACK_STGTAB_SQLPATCH Procedure .....	137-39
<b>138 DBMS_SQLPA</b>	
<b>Using DBMS_SQLPA</b> .....	138-2
Overview .....	138-3
Security Model.....	138-4
<b>Summary of DBMS_SQLPA Subprograms</b> .....	138-5
CANCEL_ANALYSIS_TASK Procedure.....	138-6
CREATE_ANALYSIS_TASK Functions.....	138-7
DROP_ANALYSIS_TASK Procedure .....	138-10
EXECUTE_ANALYSIS_TASK Function & Procedure .....	138-11
INTERRUPT_ANALYSIS_TASK Procedure.....	138-14
REPORT_ANALYSIS_TASK Function .....	138-15
RESET_ANALYSIS_TASK Procedure.....	138-17
RESUME_ANALYSIS_TASK Procedure .....	138-18
SET_ANALYSIS_TASK_PARAMETER Procedures.....	138-19
SET_ANALYSIS_DEFAULT_PARAMETER Procedures.....	138-21
<b>139 DBMS_SQLTUNE</b>	
<b>Using DBMS_SQLTUNE</b> .....	139-2
Overview .....	139-3
Security Model.....	139-6
<b>Data Structures</b> .....	139-7
SQLSET_ROW Object Type.....	139-8
<b>Subprogram Groups</b> .....	139-10
SQL Tuning Advisor Subprograms.....	139-11
SQL Profile Subprograms .....	139-12
SQL Tuning Set Subprograms.....	139-13
Real-time SQL Monitoring Subprograms.....	139-14
SQL Performance Reporting Subprograms.....	139-15
<b>Summary of DBMS_SQLTUNE Subprograms</b> .....	139-16
ACCEPT_SQL_PROFILE Procedure and Function .....	139-19
ADD_SQLSET_REFERENCE Function .....	139-22
ALTER_SQL_PROFILE Procedure.....	139-23
CANCEL_TUNING_TASK Procedure .....	139-25
CAPTURE_CURSOR_CACHE_SQLSET Procedure.....	139-26
CREATE_SQL_PLAN_BASELINE Procedure.....	139-28
CREATE_SQLSET Procedure and Function .....	139-29
CREATE_STGTAB_SQLPROF Procedure.....	139-30
CREATE_STGTAB_SQLSET Procedure .....	139-31
CREATE_TUNING_TASK Functions .....	139-33
DELETE_SQLSET Procedure .....	139-37
DROP_SQL_PROFILE Procedure.....	139-38
DROP_SQLSET Procedure .....	139-39
DROP_TUNING_TASK Procedure.....	139-40
EXECUTE_TUNING_TASK Function & Procedure .....	139-41
IMPLEMENT_TUNING_TASK Function .....	139-42

INTERRUPT_TUNING_TASK Procedure .....	139-43
LOAD_SQLSET Procedure .....	139-44
PACK_STGTAB_SQLPROF Procedure .....	139-48
PACK_STGTAB_SQLSET Procedure .....	139-49
REMAP_STGTAB_SQLPROF Procedure .....	139-51
REMAP_STGTAB_SQLSET Procedure .....	139-52
REMOVE_SQLSET_REFERENCE Procedure .....	139-53
REPORT_AUTO_TUNING_TASK Function .....	139-54
REPORT_SQL_DETAIL Function .....	139-56
REPORT_SQL_MONITOR Function .....	139-59
REPORT_SQL_MONITOR_LIST Function .....	139-64
REPORT_TUNING_TASK Function .....	139-66
RESET_TUNING_TASK Procedure .....	139-68
RESUME_TUNING_TASK Procedure .....	139-69
SCRIPT_TUNING_TASK Function .....	139-70
SELECT_CURSOR_CACHE Function .....	139-72
SELECT_SQL_TRACE Function .....	139-76
SELECT_SQLPA_TASK Function .....	139-78
SELECT_SQLSET Function .....	139-80
SELECT_WORKLOAD_REPOSITORY Functions .....	139-82
SET_TUNING_TASK_PARAMETER Procedures .....	139-84
SQLTEXT_TO_SIGNATURE Function .....	139-86
UNPACK_STGTAB_SQLPROF Procedure .....	139-87
UNPACK_STGTAB_SQLSET Procedure .....	139-88
UPDATE_SQLSET Procedures .....	139-90

## 140 DBMS\_STAT\_FUNCS

<b>Summary of DBMS_STAT_FUNCS Subprograms</b> .....	140-2
EXPONENTIAL_DIST_FIT Procedure .....	140-3
NORMAL_DIST_FIT Procedure .....	140-4
POISSON_DIST_FIT Procedure .....	140-5
SUMMARY Procedure .....	140-6
UNIFORM_DIST_FIT Procedure .....	140-7
WEIBULL_DIST_FIT Procedure .....	140-8

## 141 DBMS\_STATS

<b>Using DBMS_STATS</b> .....	141-2
Overview .....	141-3
Types .....	141-4
Constants .....	141-5
Operational Notes .....	141-6
Deprecated Subprograms .....	141-12
Examples .....	141-13
<b>Summary of DBMS_STATS Subprograms</b> .....	141-15
ALTER_STATS_HISTORY_RETENTION Procedure .....	141-20
CONVERT_RAW_VALUE Procedures .....	141-21
CONVERT_RAW_VALUE_NVARCHAR Procedure .....	141-22

CONVERT_RAW_VALUE_ROWID Procedure.....	141-23
COPY_TABLE_STATS Procedure .....	141-24
CREATE_EXTENDED_STATS Function.....	141-26
CREATE_STAT_TABLE Procedure .....	141-28
DELETE_COLUMN_STATS Procedure .....	141-29
DELETE_DATABASE_PREFS Procedure .....	141-31
DELETE_DATABASE_STATS Procedure .....	141-34
DELETE_DICTIONARY_STATS Procedure .....	141-35
DELETE_FIXED_OBJECTS_STATS Procedure .....	141-36
DELETE_INDEX_STATS Procedure .....	141-37
DELETE_PENDING_STATS Procedure .....	141-38
DELETE_SCHEMA_PREFS Procedure.....	141-39
DELETE_SCHEMA_STATS Procedure .....	141-42
DELETE_SYSTEM_STATS Procedure .....	141-43
DELETE_TABLE_PREFS Procedure .....	141-44
DELETE_TABLE_STATS Procedure .....	141-47
DIFF_TABLE_STATS_IN_HISTORY Function.....	141-49
DIFF_TABLE_STATS_IN_PENDING Function .....	141-50
DIFF_TABLE_STATS_IN_STATTAB Function .....	141-51
DROP_EXTENDED_STATS Procedure .....	141-52
DROP_STAT_TABLE Procedure .....	141-53
EXPORT_COLUMN_STATS Procedure.....	141-54
EXPORT_DATABASE_PREFS Procedure.....	141-55
EXPORT_DATABASE_STATS Procedure.....	141-56
EXPORT_DICTIONARY_STATS Procedure .....	141-57
EXPORT_FIXED_OBJECTS_STATS Procedure .....	141-58
EXPORT_INDEX_STATS Procedure.....	141-59
EXPORT_PENDING_STATS Procedure .....	141-60
EXPORT_SCHEMA_PREFS Procedure .....	141-61
EXPORT_SCHEMA_STATS Procedure.....	141-62
EXPORT_SYSTEM_STATS Procedure.....	141-63
EXPORT_TABLE_PREFS Procedure.....	141-64
EXPORT_TABLE_STATS Procedure.....	141-65
FLUSH_DATABASE_MONITORING_INFO Procedure.....	141-66
GATHER_DATABASE_STATS Procedures.....	141-67
GATHER_DICTIONARY_STATS Procedure .....	141-71
GATHER_FIXED_OBJECTS_STATS Procedure.....	141-75
GATHER_INDEX_STATS Procedure .....	141-76
GATHER_SCHEMA_STATS Procedures.....	141-78
GATHER_SYSTEM_STATS Procedure.....	141-82
GATHER_TABLE_STATS Procedure .....	141-84
GENERATE_STATS Procedure .....	141-88
GET_COLUMN_STATS Procedures.....	141-89
GET_INDEX_STATS Procedures.....	141-91
GET_PARAM Function.....	141-94
GET_PREFS Function .....	141-95
GET_STATS_HISTORY_AVAILABILITY Function .....	141-98

GET_STATS_HISTORY_RETENTION Function .....	141-99
GET_SYSTEM_STATS Procedure.....	141-100
GET_TABLE_STATS Procedure .....	141-102
IMPORT_COLUMN_STATS Procedure.....	141-104
IMPORT_DATABASE_PREFS Procedure .....	141-106
IMPORT_DATABASE_STATS Procedure.....	141-107
IMPORT_DICTIONARY_STATS Procedure .....	141-108
IMPORT_FIXED_OBJECTS_STATS Procedure .....	141-110
IMPORT_INDEX_STATS Procedure.....	141-111
IMPORT_SCHEMA_PREFS Procedure .....	141-112
IMPORT_SCHEMA_STATS Procedure.....	141-113
IMPORT_SYSTEM_STATS Procedure.....	141-115
IMPORT_TABLE_PREFS Procedure.....	141-116
IMPORT_TABLE_STATS Procedure.....	141-117
LOCK_PARTITION_STATS Procedure.....	141-119
LOCK_SCHEMA_STATS Procedure .....	141-120
LOCK_TABLE_STATS Procedure .....	141-121
MERGE_COL_USAGE Procedure.....	141-122
PREPARE_COLUMN_VALUES Procedures.....	141-123
PREPARE_COLUMN_VALUES_NVARCHAR2 Procedure.....	141-125
PREPARE_COLUMN_VALUES_ROWID Procedure .....	141-127
PUBLISH_PENDING_STATS Procedure .....	141-129
PURGE_STATS Procedure.....	141-130
RESET_GLOBAL_PREF_DEFAULTS Procedure.....	141-131
RESET_PARAM_DEFAULTS Procedure .....	141-132
RESTORE_DATABASE_STATS Procedure .....	141-133
RESTORE_DICTIONARY_STATS Procedure .....	141-134
RESTORE_FIXED_OBJECTS_STATS Procedure.....	141-135
RESTORE_SCHEMA_STATS Procedure.....	141-136
RESTORE_SYSTEM_STATS Procedure.....	141-137
RESTORE_TABLE_STATS Procedure .....	141-138
SEED_COL_USAGE Procedure .....	141-139
SET_COLUMN_STATS Procedures.....	141-140
SET_DATABASE_PREFS Procedure.....	141-142
SET_GLOBAL_PREFS Procedure .....	141-145
SET_INDEX_STATS Procedures.....	141-148
SET_PARAM Procedure .....	141-151
SET_SCHEMA_PREFS Procedure .....	141-153
SET_SYSTEM_STATS Procedure.....	141-154
SET_TABLE_PREFS Procedure.....	141-156
SET_TABLE_STATS Procedure .....	141-159
SHOW_EXTENDED_STATS_NAME Function.....	141-161
UNLOCK_PARTITION_STATS Procedure .....	141-162
UNLOCK_SCHEMA_STATS Procedure.....	141-163
UNLOCK_TABLE_STATS Procedure.....	141-164
UPGRADE_STAT_TABLE Procedure.....	141-165

---

## 142 DBMS\_STORAGE\_MAP

Using DBMS_STORAGE_MAP .....	142-2
Overview .....	142-3
Operational Notes .....	142-4
<b>Summary of DBMS_STORAGE_MAP Subprograms .....</b>	<b>142-5</b>
DROP_ALL Function.....	142-6
DROP_ELEMENT Function .....	142-7
DROP_FILE Function .....	142-8
LOCK_MAP Procedure.....	142-9
MAP_ALL Function.....	142-10
MAP_ELEMENT Function .....	142-11
MAP_FILE Function .....	142-12
MAP_OBJECT Function.....	142-13
RESTORE Function.....	142-14
SAVE Function .....	142-15
UNLOCK_MAP Procedure .....	142-16

## 143 DBMS\_STREAMS

Using DBMS_STREAMS.....	143-2
Overview .....	143-3
Security Model.....	143-4
<b>Summary of DBMS_STREAMS Subprograms .....</b>	<b>143-5</b>
COMPATIBLE_11_2 Function.....	143-6
COMPATIBLE_11_1 Function.....	143-7
COMPATIBLE_10_2 Function.....	143-8
COMPATIBLE_10_1 Function.....	143-9
COMPATIBLE_9_2 Function.....	143-10
CONVERT_ANYDATA_TO_LCR_DDL Function .....	143-11
CONVERT_ANYDATA_TO_LCR_ROW Function.....	143-12
CONVERT_LCR_TO_XML Function.....	143-13
CONVERT_XML_TO_LCR Function.....	143-14
GET_INFORMATION Function .....	143-15
GET_STREAMS_NAME Function.....	143-16
GET_STREAMS_TYPE Function .....	143-17
GET_TAG Function .....	143-18
MAX_COMPATIBLE Function .....	143-19
SET_TAG Procedure.....	143-20

## 144 DBMS\_STREAMS\_ADM

Using DBMS_STREAMS_ADM .....	144-2
Overview .....	144-3
Deprecated Subprograms .....	144-4
Security Model.....	144-5
Operational Notes .....	144-9
<b>Summary of DBMS_STREAMS_ADM Subprograms.....</b>	<b>144-23</b>
ADD_COLUMN Procedure .....	144-27

ADD_GLOBAL_PROPAGATION_RULES Procedure.....	144-30
ADD_GLOBAL_RULES Procedure.....	144-34
ADD_MESSAGE_PROPAGATION_RULE Procedure .....	144-39
ADD_MESSAGE_RULE Procedure .....	144-43
ADD_SCHEMA_PROPAGATION_RULES Procedure.....	144-46
ADD_SCHEMA_RULES Procedure.....	144-50
ADD_SUBSET_PROPAGATION_RULES Procedure.....	144-55
ADD_SUBSET_RULES Procedure.....	144-59
ADD_TABLE_PROPAGATION_RULES Procedure .....	144-65
ADD_TABLE_RULES Procedure .....	144-70
CLEANUP_INSTANTIATION_SETUP Procedure .....	144-76
DELETE_COLUMN Procedure .....	144-80
GET_MESSAGE_TRACKING Function .....	144-83
GET_SCN_MAPPING Procedure.....	144-84
GET_TAG Function .....	144-86
KEEP_COLUMNS Procedure.....	144-87
MAINTAIN_CHANGE_TABLE Procedure.....	144-90
MAINTAIN_GLOBAL Procedure .....	144-97
MAINTAIN_SCHEMAS Procedure .....	144-101
MAINTAIN_SIMPLE_TABLESPACE Procedure .....	144-106
MAINTAIN_SIMPLE_TTS Procedure .....	144-111
MAINTAIN_TABLES Procedure.....	144-114
MAINTAIN_TABLESPACES Procedure.....	144-119
MAINTAIN_TTS Procedure.....	144-126
MERGE_STREAMS Procedure .....	144-130
MERGE_STREAMS_JOB Procedure .....	144-133
POST_INSTANTIATION_SETUP Procedure.....	144-136
PRE_INSTANTIATION_SETUP Procedure.....	144-141
PURGE_SOURCE_CATALOG Procedure .....	144-146
RECOVER_OPERATION Procedure .....	144-147
REMOVE_QUEUE Procedure.....	144-149
REMOVE_RULE Procedure .....	144-150
REMOVE_STREAMS_CONFIGURATION Procedure .....	144-152
RENAME_COLUMN Procedure .....	144-154
RENAME_SCHEMA Procedure .....	144-157
RENAME_TABLE Procedure.....	144-159
SET_MESSAGE_NOTIFICATION Procedure .....	144-161
SET_MESSAGE_TRACKING Procedure.....	144-166
SET_RULE_TRANSFORM_FUNCTION Procedure.....	144-167
SET_TAG Procedure.....	144-171
SET_UP_QUEUE Procedure.....	144-172
SPLIT_STREAMS Procedure .....	144-175

## 145 DBMS\_STREAMS\_ADVISOR\_ADM

Using DBMS_STREAMS_ADVISOR_ADM .....	145-2
Overview .....	145-3
Security Model.....	145-4



Constants .....	145-5
Views.....	145-6
Operational Notes .....	145-7
<b>Summary of DBMS_STREAMS_ADVISOR_ADM Subprograms</b> .....	145-9
ANALYZE_CURRENT_PERFORMANCE Procedure .....	145-10
<b>146  DBMS_STREAMS_AUTH</b>	
Using DBMS_STREAMS_AUTH .....	146-2
Overview .....	146-3
Security Model.....	146-4
<b>Summary of DBMS_STREAMS_AUTH Subprograms</b> .....	146-5
GRANT_ADMIN_PRIVILEGE Procedure .....	146-6
GRANT_REMOTE_ADMIN_ACCESS Procedure.....	146-9
REVOKE_ADMIN_PRIVILEGE Procedure .....	146-10
REVOKE_REMOTE_ADMIN_ACCESS Procedure .....	146-12
<b>147  DBMS_STREAMS_HANDLER_ADM</b>	
Using DBMS_STREAMS_HANDLER_ADM .....	147-2
Overview .....	147-3
Security Model.....	147-4
Views.....	147-5
Operational Notes .....	147-6
<b>Summary of DBMS_STREAMS_HANDLER_ADM Subprograms</b> .....	147-10
ADD_STMT_TO_HANDLER Procedure .....	147-11
CREATE_STMT_HANDLER Procedure .....	147-12
DROP_STMT_HANDLER Procedure.....	147-13
REMOVE_STMT_FROM_HANDLER Procedure.....	147-14
<b>148  DBMS_STREAMS_MESSAGING</b>	
Using DBMS_STREAMS_MESSAGING.....	148-2
Overview .....	148-3
Security Model.....	148-4
<b>Summary of DBMS_STREAMS_MESSAGING Subprograms</b> .....	148-5
DEQUEUE Procedure.....	148-6
ENQUEUE Procedure .....	148-8
<b>149  DBMS_STREAMS_TABLESPACE_ADM</b>	
Using DBMS_STREAMS_TABLESPACE_ADM .....	149-2
Overview .....	149-3
Security Model.....	149-4
<b>Data Structures</b> .....	149-5
DIRECTORY_OBJECT_SET Table Type.....	149-6
FILE Record Type.....	149-7
FILE_SET Table Type .....	149-8
TABLESPACE_SET Table Type .....	149-9
<b>Summary of DBMS_STREAMS_TABLESPACE_ADM Subprograms</b> .....	149-10

ATTACH_SIMPLE_TABLESPACE Procedure.....	149-11
ATTACH_TABLESPACES Procedure .....	149-13
CLONE_SIMPLE_TABLESPACE Procedure.....	149-18
CLONE_TABLESPACES Procedure .....	149-20
DETACH_SIMPLE_TABLESPACE Procedure.....	149-24
DETACH_TABLESPACES Procedure .....	149-26
PULL_SIMPLE_TABLESPACE Procedure.....	149-30
PULL_TABLESPACES Procedure .....	149-32
<b>150 DBMS_TDB</b>	
<b>Using DBMS_TDB</b> .....	150-2
Overview .....	150-3
Security Model.....	150-4
Constants .....	150-5
Views.....	150-6
Operational Notes .....	150-7
<b>Summary of DBMS_TDB Subprograms</b> .....	150-8
CHECK_DB Function .....	150-9
CHECK_EXTERNAL Function .....	150-11
<b>151 DBMS_TRACE</b>	
<b>Using DBMS_TRACE</b> .....	151-2
Overview .....	151-3
Security Model.....	151-4
Constants .....	151-5
Restrictions .....	151-6
Operational Notes .....	151-7
<b>Summary of DBMS_TRACE Subprograms</b> .....	151-10
CLEAR_PLSQL_TRACE Procedure .....	151-11
GET_PLSQL_TRACE_LEVEL Function .....	151-12
PLSQL_TRACE_VERSION Procedure .....	151-13
SET_PLSQL_TRACE Procedure .....	151-14
<b>152 DBMS_TRANSACTION</b>	
<b>Using DBMS_TRANSACTION</b> .....	152-2
Security Model.....	152-3
<b>Summary of DBMS_TRANSACTION Subprograms</b> .....	152-4
ADVISE_COMMIT Procedure .....	152-5
ADVISE_NOHING Procedure .....	152-6
ADVISE_ROLLBACK Procedure .....	152-7
COMMIT Procedure .....	152-8
COMMIT_COMMENT Procedure .....	152-9
COMMIT_FORCE Procedure.....	152-10
LOCAL_TRANSACTION_ID Function.....	152-11
PURGE_LOST_DB_ENTRY Procedure.....	152-12
PURGE_MIXED Procedure .....	152-14

READ_ONLY Procedure.....	152-15
READ_WRITE Procedure .....	152-16
ROLLBACK Procedure .....	152-17
ROLLBACK_FORCE Procedure .....	152-18
ROLLBACK_SAVEPOINT Procedure .....	152-19
SAVEPOINT Procedure .....	152-20
STEP_ID Function .....	152-21
USE_ROLLBACK_SEGMENT Procedure .....	152-22
<b>153 DBMS_TRANSFORM</b>	
<b>Summary of DBMS_TRANSFORM Subprograms</b> .....	153-2
CREATE_TRANSFORMATION Procedure.....	153-3
DROP_TRANSFORMATION Procedure .....	153-5
MODIFY_TRANSFORMATION Procedure .....	153-6
<b>154 DBMS_TTS</b>	
<b>Using DBMS_TTS</b> .....	154-2
Security Model.....	154-3
Exceptions .....	154-4
Operational Notes .....	154-5
<b>Summary of DBMS_TTS Subprograms</b> .....	154-6
DOWNGRADE Procedure.....	154-7
TRANSPORT_SET_CHECK Procedure.....	154-8
<b>155 DBMS_TYPES</b>	
<b>Using DBMS_TYPES</b> .....	155-2
Constants .....	155-3
Exceptions .....	155-4
<b>156 DBMS_UTILITY</b>	
<b>Using DBMS_UTILITY</b> .....	156-2
Security Model.....	156-3
Constants .....	156-4
Exceptions .....	156-5
<b>Data Structures</b> .....	156-6
INSTANCE_RECORD Record Type .....	156-7
DBLINK_ARRAY TABLE Type .....	156-8
INDEX_TABLE_TYPE Table Type .....	156-9
INSTANCE_TABLE Table Type .....	156-10
LNAME_ARRAY Table Type.....	156-11
NAME_ARRAY Table Type .....	156-12
NUMBER_ARRAY Table Type.....	156-13
UNCL_ARRAY Table Type .....	156-14
<b>Summary of DBMS_UTILITY Subprograms</b> .....	156-15
ACTIVE_INSTANCES Procedure .....	156-17
ANALYZE_DATABASE Procedure .....	156-18

ANALYZE_PART_OBJECT Procedure .....	156-19
ANALYZE_SCHEMA Procedure .....	156-20
CANONICALIZE Procedure.....	156-21
COMMA_TO_TABLE Procedures.....	156-22
COMPILE_SCHEMA Procedure .....	156-23
CREATE_ALTER_TYPE_ERROR_TABLE Procedure.....	156-24
CURRENT_INSTANCE Function.....	156-25
DATA_BLOCK_ADDRESS_BLOCK Function .....	156-26
DATA_BLOCK_ADDRESS_FILE Function .....	156-27
DB_VERSION Procedure .....	156-28
EXEC_DDL_STATEMENT Procedure.....	156-29
FORMAT_CALL_STACK Function .....	156-30
FORMAT_ERROR_BACKTRACE Function .....	156-31
FORMAT_ERROR_STACK Function.....	156-34
GET_CPU_TIME Function.....	156-35
GET_DEPENDENCY Procedure .....	156-36
GET_ENDIANNNESS Function .....	156-37
GET_HASH_VALUE Function .....	156-38
GET_PARAMETER_VALUE Function.....	156-39
GET_SQL_HASH Function .....	156-41
GET_TIME Function.....	156-42
GET_TZ_TRANSITIONS Procedure.....	156-43
INVALIDATE Procedure.....	156-44
IS_BIT_SET Function .....	156-47
IS_CLUSTER_DATABASE Function.....	156-48
MAKE_DATA_BLOCK_ADDRESS Function.....	156-49
NAME_RESOLVE Procedure.....	156-50
NAME_TOKENIZE Procedure .....	156-52
OLD_CURRENT_SCHEMA Function .....	156-53
OLD_CURRENT_USER Function .....	156-54
PORT_STRING Function .....	156-55
SQLID_TO_SQLHASH Function.....	156-56
TABLE_TO_COMMA Procedures.....	156-57
VALIDATE Procedure.....	156-58
WAIT_ON_PENDING_DML Function .....	156-60

## 157 DBMS\_WARNING

Using DBMS_WARNING .....	157-2
Security Model.....	157-3
<b>Summary of DBMS_WARNING Subprograms</b> .....	157-4
ADD_WARNING_SETTING_CAT Procedure.....	157-5
ADD_WARNING_SETTING_NUM Procedure.....	157-6
GET_CATEGORY Function.....	157-7
GET_WARNING_SETTING_CAT Function.....	157-8
GET_WARNING_SETTING_NUM Function.....	157-9
GET_WARNING_SETTING_STRING Function.....	157-10
SET_WARNING_SETTING_STRING Procedure.....	157-11

<b>158</b>	<b>DBMS_WM</b>	
	Documentation of DBMS_WM .....	158-2
<b>159</b>	<b>DBMS_WORKLOAD_CAPTURE</b>	
	Using DBMS_WORKLOAD_CAPTURE.....	159-2
	Overview .....	159-3
	Security Model.....	159-4
	<b>Summary of DBMS_WORKLOAD_CAPTURE Subprograms</b> .....	159-5
	ADD_FILTER Procedures .....	159-6
	DELETE_CAPTURE_INFO Procedure .....	159-7
	DELETE_FILTER Procedure.....	159-8
	EXPORT_AWR Procedure.....	159-9
	FINISH_CAPTURE Procedure.....	159-10
	GET_CAPTURE_INFO Function.....	159-11
	IMPORT_AWR Function .....	159-12
	REPORT Function .....	159-13
	START_CAPTURE Procedure.....	159-14
<b>160</b>	<b>DBMS_WORKLOAD_REPLAY</b>	
	Using DBMS_WORKLOAD_REPLAY .....	160-2
	Security Model.....	160-3
	<b>Summary of DBMS_WORKLOAD_REPLAY Subprograms</b> .....	160-4
	ADD_FILTER Procedure.....	160-6
	CALIBRATE Function .....	160-7
	CANCEL_REPLAY Procedure.....	160-8
	COMPARE_PERIOD_REPORT Procedure .....	160-9
	COMPARE_SQLSET_REPORT Function .....	160-10
	CREATE_FILTER_SET Procedure .....	160-11
	DELETE_FILTER Procedure.....	160-12
	DELETE_REPLAY_INFO Procedure .....	160-13
	EXPORT_AWR Procedure.....	160-14
	GET_DIVERGING_STATEMENT Function .....	160-15
	GET_REPLAY_INFO Function .....	160-16
	GET_REPLAY_TIMEOUT Procedure .....	160-17
	IMPORT_AWR Function .....	160-18
	INITIALIZE_REPLAY Procedure.....	160-19
	IS_REPLAY_PAUSED Function .....	160-20
	PAUSE_REPLAY Procedure.....	160-21
	POPULATE_DIVERGENCE Procedure .....	160-22
	PREPARE_REPLAY Procedure.....	160-23
	PROCESS_CAPTURE Procedure.....	160-27
	REMAP_CONNECTION Procedure.....	160-28
	REPORT Function .....	160-29
	RESUME_REPLAY Procedure .....	160-30
	REUSE_REPLAY_FILTER_SET Procedure .....	160-31
	SET_ADVANCED_PARAMETER Procedure .....	160-32

SET_REPLAY_TIMEOUT Procedure .....	160-33
START_REPLAY Procedure .....	160-34
USE_FILTER_SET Procedure .....	160-35

## 161 DBMS\_WORKLOAD\_REPOSITORY

<b>Using DBMS_WORKLOAD_REPOSITORY</b> .....	161-2
Examples .....	161-3
<b>Data Structures</b> .....	161-4
AWR_BASELINE_METRIC_TYPE Object Type .....	161-5
AWR_BASELINE_METRIC_TYPE_TABLE Table Type .....	161-6
AWRRPT_INSTANCE_LIST_TYPE Table Type .....	161-7
<b>Summary of DBMS_WORKLOAD_REPOSITORY Subprograms</b> .....	161-8
ADD_COLORED_SQL Procedure .....	161-10
ASH_GLOBAL_REPORT_HTML Function .....	161-11
ASH_GLOBAL_REPORT_TEXT Function .....	161-13
ASH_REPORT_HTML Function .....	161-15
ASH_REPORT_TEXT Function .....	161-18
AWR_DIFF_REPORT_HTML Function .....	161-21
AWR_DIFF_REPORT_TEXT Function .....	161-22
AWR_GLOBAL_DIFF_REPORT_HTML Functions .....	161-23
AWR_GLOBAL_DIFF_REPORT_TEXT Functions .....	161-24
AWR_GLOBAL_REPORT_HTML Functions .....	161-25
AWR_GLOBAL_REPORT_TEXT Functions .....	161-26
AWR_REPORT_HTML Function .....	161-27
AWR_REPORT_TEXT Function .....	161-28
AWR_SQL_REPORT_HTML Function .....	161-29
AWR_SQL_REPORT_TEXT Function .....	161-30
CREATE_BASELINE Functions & Procedures .....	161-31
CREATE_BASELINE_TEMPLATE Procedures .....	161-33
CREATE_SNAPSHOT Function and Procedure .....	161-34
DROP_BASELINE Procedure .....	161-35
DROP_BASELINE_TEMPLATE Procedure .....	161-36
DROP_SNAPSHOT_RANGE Procedure .....	161-37
MODIFY_SNAPSHOT_SETTINGS Procedures .....	161-38
MODIFY_BASELINE_WINDOW_SIZE Procedure .....	161-40
REMOVE_COLORED_SQL Procedure .....	161-41
RENAME_BASELINE Procedure .....	161-42
SELECT_BASELINE_METRICS Function .....	161-43

## 162 DBMS\_XA

<b>Using DBMS_XA</b> .....	162-2
Overview .....	162-3
Security Model .....	162-4
Constants .....	162-5
Operational Notes .....	162-7
<b>Data Structures</b> .....	162-8
DBMS_XA_XID Object Type .....	162-9

DBMS_XA_XID_ARRAY Table Type .....	162-10
<b>Summary of DBMS_XA Subprograms .....</b>	<b>162-11</b>
DIST_TXN_SYNC Procedure .....	162-12
XA_COMMIT Function.....	162-13
XA_END Function .....	162-14
XA_FORGET Function .....	162-15
XA_GETLASTOER Function.....	162-16
XA_PREPARE Function .....	162-17
XA_RECOVER Function .....	162-18
XA_ROLLBACK Function .....	162-19
XA_SETTIMEOUT Function .....	162-20
XA_START Function .....	162-21

## 163 DBMS\_XDB

<b>Using DBMS_XDB.....</b>	<b>163-2</b>
Overview .....	163-3
Deprecated Subprograms .....	163-4
Security Model.....	163-5
Constants .....	163-6
<b>Summary of DBMS_XDB Subprograms.....</b>	<b>163-7</b>
ACLCHECKPRIVILEGES Function .....	163-10
ADDHTTPEXPIREMAPPING Procedure .....	163-11
ADDMIMEMAPPING Procedure .....	163-12
ADDSCHEMALOCMAPPING Procedure.....	163-13
ADDSERVLET Procedure.....	163-14
ADDSERVLETMAPPING Procedure .....	163-15
ADDSERVLETSECROLE Procedure.....	163-16
ADDXMLEXTENSION Procedure .....	163-17
APPENDPATH Procedure .....	163-18
APPENDRESOURCEMETADATA Procedure.....	163-19
CFG_GET Function.....	163-20
CFG_REFRESH Procedure .....	163-21
CFG_UPDATE Procedure.....	163-22
CHANGEOWNER Procedure.....	163-23
CHANGEPRIVILEGES Function.....	163-24
CHECKPRIVILEGES Function .....	163-25
CREATEFOLDER Function.....	163-26
CREATEOIDPATH Function .....	163-27
CREATERESOURCE Functions.....	163-28
DELETEHTTPEXPIREMAPPING Procedure .....	163-30
DELETEMIMEMAPPING Procedure .....	163-31
DELETERESOURCE Procedure.....	163-32
DELETERESOURCEMETADATA Procedures .....	163-33
DELETESCHEMALOCMAPPING Procedure.....	163-34
DELETESERVLET Procedure.....	163-35
DELETESERVLETMAPPING Procedure .....	163-36
DELETESERVLETSECROLE Procedure.....	163-37

DELETXMLEXTENSION Procedure .....	163-38
EXISTSRESOURCE Function .....	163-39
GETACLDOCUMENT Function .....	163-40
GETCHILDRESPATHS Function .....	163-41
GETCONTENTBLOB Function.....	163-42
GETCONTENTCLOB Function .....	163-43
GETCONTENTVARCHAR2 Function .....	163-44
GETCONTENTXMLREF Function.....	163-45
GETCONTENTXMLTYPE Function .....	163-46
GETFTPPORT Function .....	163-47
GETHTTPPORT Function.....	163-48
GETLISTENERENDPOINT Procedure.....	163-49
GETLOCKTOKEN Procedure .....	163-50
GETPRIVILEGES Function.....	163-51
GETRESOID Function .....	163-52
GETXDB_TABLESPACE Function.....	163-53
HASBLOBCONTENT Function .....	163-54
HASCHARCONTENT Function .....	163-55
HASXMLCONTENT Function.....	163-56
HASXMLREFERENCE Function.....	163-57
ISFOLDER Function .....	163-58
LINK Procedures.....	163-59
LOCKRESOURCE Function .....	163-60
MOVEXDB_TABLESPACE Procedure .....	163-61
PROCESSLINKS Procedure.....	163-62
PURGERESOURCEMETADATA Procedure.....	163-63
REBUILDHIERARCHICALINDEX Procedure.....	163-64
RENAMERESOURCE Procedure .....	163-65
SETACL Procedure .....	163-66
SETCONTENT Procedures.....	163-67
SETFTPPORT Procedure.....	163-69
SETHHTTPPORT Procedure .....	163-70
SETLISTENERENDPOINT Procedure.....	163-71
SETLISTENERLOCALACCESS Procedure.....	163-72
SPLITPATH Procedure .....	163-73
TOUCHRESOURCE Procedure .....	163-74
UPDATERESOURCEMETADATA Procedures .....	163-75
UNLOCKRESOURCE Function.....	163-77

## 164 DBMS\_XDB\_ADMIN

Using DBMS_XDB_ADMIN.....	164-2
Deprecated Subprograms .....	164-3
Security Model.....	164-4
Summary of DBMS_XDB_ADMIN Subprograms .....	164-5
CREATEREPOSITORYXMLINDEX Procedure.....	164-6
DROPREPOSITORYXMLINDEX Procedure .....	164-7
MOVEXDB_TABLESPACE Procedure .....	164-8



	REBUILDHIERARCHICALINDEX Procedure.....	164-9
	XMLINDEXADDPATH Procedure .....	164-10
	XMLINDEXREMOVEPATH Procedure .....	164-11
<b>165</b>	<b>DBMS_XDB_VERSION</b>	
	Using DBMS_XDB_VERSION .....	165-2
	Security Model.....	165-3
	<b>Summary of DBMS_XDB_VERSION Subprograms</b> .....	165-4
	CHECKIN Function.....	165-5
	CHECKOUT Procedure .....	165-6
	GETCONTENTSBLOBBYRESID Function.....	165-7
	GETCONTENTSCLOBBYRESID Function .....	165-8
	GETCONTENTSEXMLBYRESID Function .....	165-9
	GETPREDECESSORS Function.....	165-10
	GETPREDSBYRESID Function.....	165-11
	GETRESOURCEBYRESID Function.....	165-12
	GETSUCCESSORS Function.....	165-13
	GETSUCCSBYRESID Function .....	165-14
	MAKEVERSIONED Function .....	165-15
	UNCHECKOUT Function .....	165-16
<b>166</b>	<b>DBMS_XDBRESOURCE</b>	
	Using DBMS_XDBRESOURCE .....	166-2
	Overview .....	166-3
	Security Model.....	166-4
	<b>Summary of DBMS_XDBRESOURCE Subprograms</b> .....	166-5
	FREERESOURCE Procedure .....	166-8
	GETACL Function.....	166-9
	GETACLDOCFROMRES Function .....	166-10
	GETAUTHOR Function .....	166-11
	GETCHARACTERSET Function.....	166-12
	GETCOMMENT Function .....	166-13
	GETCONTENTBLOB Function.....	166-14
	GETCONTENTCLOB Function .....	166-15
	GETCONTENTREF Function.....	166-16
	GETCONTENTTYPE Function .....	166-17
	GETCONTENTXML Function .....	166-18
	GETCONTENTVARCHAR2 Function .....	166-19
	GETCREATIONDATE Function.....	166-20
	GETCREATOR Function.....	166-21
	GETCUSTOMMETADATA Function .....	166-22
	GETDISPLAYNAME Function .....	166-23
	GETLANGUAGE Function .....	166-24
	GETLASTMODIFIER Function.....	166-25
	GETMODIFICATIONDATE Function.....	166-26
	GETOWNER Function .....	166-27
	GETREFCOUNT Function.....	166-28

GETVERSIONID Function.....	166-29
HASACLCHANGED Function.....	166-30
HASAUTHORCHANGED Function .....	166-31
HASCHANGED Function .....	166-32
HASCHARACTERSETCHANGED Function.....	166-33
HASCOMMENTCHANGED Function .....	166-34
HASCONTENTCHANGED Function .....	166-35
HASCONTENTTYPECHANGED Function .....	166-36
HASCREATIONDATECHANGED Function.....	166-37
HASCREATORCHANGED Function.....	166-38
HASCUSTOMMETADATACHANGED Function .....	166-39
HASDISPLAYNAMECHANGED Function .....	166-40
HASLANGUAGECHANGED Function.....	166-41
HASLASTMODIFIERCHANGED Function.....	166-42
HASMODIFICATIONDATECHANGED Function.....	166-43
HASOWNERCHANGED Function.....	166-44
HASREFCOUNTCHANGED Function .....	166-45
HASVERSIONIDCHANGED Function.....	166-46
ISFOLDER Function .....	166-47
ISNULL Function .....	166-48
MAKEDOCUMENT Function.....	166-49
SAVE Procedure.....	166-50
SETACL Procedure .....	166-51
SETAUTHOR Procedure.....	166-52
SETCHARACTERSET Procedure .....	166-53
SETCOMMENT Procedure.....	166-54
SETCONTENT Procedures.....	166-55
SETCONTENTTYPE Procedure.....	166-56
SETCUSTOMMETADATA Procedure .....	166-57
SETDISPLAYNAME Procedure.....	166-58
SETLANGUAGE Procedure.....	166-59
SETOWNER Procedure.....	166-60

## 167 DBMS\_XDBT

Using DBMS_XDBT .....	167-2
Overview .....	167-3
Security Model.....	167-4
Operational Notes .....	167-5
Summary of DBMS_XDBT Subprograms .....	167-7
CONFIGUREAUTOSYNC Procedure.....	167-8
CREATEDATASTOREPREF Procedure .....	167-9
CREATEFILTERPREF Procedure .....	167-10
CREATEINDEX Procedure.....	167-11
CREATELEXERPREF Procedure .....	167-12
CREATEPREFERENCES Procedure .....	167-13
CREATESECTIONGROUPPREF Procedure.....	167-14
CREATESTOPLISTPREF Procedure .....	167-15

CREATESTORAGEPREF Procedure.....	167-16
CREATEWORLDLISTPREF Procedure.....	167-17
DROPPREFERENCES Procedure.....	167-18
<b>168 DBMS_XDBZ</b>	
<b>Using DBMS_XDBZ</b> .....	168-2
Security Model.....	168-3
Constants.....	168-4
<b>Summary of DBMS_XDBZ Subprograms</b> .....	168-5
DISABLE_HIERARCHY Procedure.....	168-6
ENABLE_HIERARCHY Procedure.....	168-7
GET_ACLOID Function.....	168-8
GET_USERID Function.....	168-9
IS_HIERARCHY_ENABLED Function.....	168-10
PURGELDAPCACHE Function.....	168-11
<b>169 DBMS_XEVENT</b>	
<b>Using DBMS_XEVENT</b> .....	169-2
Security Model.....	169-3
Constants.....	169-4
<b>Subprogram Groups</b> .....	169-5
XDBEvent Type Subprograms.....	169-6
XDBRepositoryEvent Type Subprograms.....	169-7
XDBHandlerList Type Subprograms.....	169-8
XDBHandler Type Subprograms.....	169-9
XDBPath Type Subprograms.....	169-10
XDBLink Type Subprograms.....	169-11
<b>Summary of DBMS_XEVENT Subprograms</b> .....	169-12
CLEAR Procedure.....	169-15
GETAPPLICATIONDATA Function.....	169-16
GETCHILDROID Function.....	169-17
GETCURRENTUSER Function.....	169-18
GETEVENT Function.....	169-19
GETFIRST Function.....	169-20
GETHANDLERLIST Function.....	169-21
GETINTERFACE Function.....	169-22
GETLANGUAGE Function.....	169-23
GETLINK Function.....	169-24
GETLINKNAME Function.....	169-25
GETLOCK Function.....	169-26
GETLANGUAGE Function.....	169-27
GETNAME Function.....	169-28
GETNEXT Function.....	169-29
GETOLDRESOURCE Function.....	169-30
GETOPENACCESSMODE Function.....	169-31
GETOPENDENYMODE Function.....	169-32
GETOUTPUTSTREAM Function.....	169-33

GETPARAMETER Function.....	169-34
GETPARENT Function.....	169-35
GETPARENTNAME Function.....	169-36
GETPARENTOID Function.....	169-37
GETPARENTPATH Function.....	169-38
GETPATH Function.....	169-39
GETRESOURCE Function.....	169-40
GETSCHEMA Function.....	169-41
GETSOURCE Function.....	169-42
GETUPDATEBYTECOUNT Function.....	169-43
GETUPDATEBYTEOFFSET Function.....	169-44
GETXDBEVENT Function.....	169-45
ISNULL Functions.....	169-46
REMOVE Procedure.....	169-48
SETRENDERPATH Procedure.....	169-49
SETRENDERSTREAM Procedure.....	169-50

## 170 DBMS\_XMLDOM

Using DBMS_XMLDOM.....	170-3
Overview.....	170-4
Security Model.....	170-6
Constants.....	170-7
Types.....	170-8
Exceptions.....	170-9
<b>Subprogram Groups.....</b>	<b>170-10</b>
DOMNode Subprograms.....	170-11
DOMAttr Subprograms.....	170-13
DOMCDATASection Subprograms.....	170-14
DOMCharacterData Subprograms.....	170-15
DOMComment Subprograms.....	170-16
DOMDocument Subprograms.....	170-17
DOMDocumentFragment Subprograms.....	170-19
DOMDocumentType Subprograms.....	170-20
DOMElement Subprograms.....	170-21
DOMEntity Subprograms.....	170-22
DOMEntityReference Subprograms.....	170-23
DOMImplementation Subprograms.....	170-24
DOMNamedNodeMap Subprograms.....	170-25
DOMNodeList Subprograms.....	170-26
DOMNotation Subprograms.....	170-27
DOMProcessingInstruction Subprograms.....	170-28
DOMText Subprograms.....	170-29
<b>Summary of DBMS_XMLDOM Subprograms.....</b>	<b>170-30</b>
ADOPTNODE Function.....	170-41
APPENDCHILD Function.....	170-42
APPENDDATA Procedure.....	170-43
CLONENODE Function.....	170-44

---

CREATEATTRIBUTE Functions.....	170-45
CREATECDATASECTION Function.....	170-46
CREATECOMMENT Function.....	170-47
CREATEDOCUMENT Function.....	170-48
CREATEDOCUMENTFRAGMENT Function.....	170-49
CREATEELEMENT Functions.....	170-50
CREATEENTITYREFERENCE Function.....	170-51
CREATEPROCESSINGINSTRUCTION Function.....	170-52
CREATETEXTNODE Function.....	170-53
DELETEDATA Procedure.....	170-54
FINDENTITY Function.....	170-55
FINDNOTATION Function.....	170-56
FREEDOCFRAG Procedure.....	170-57
FREEDOCUMENT Procedure.....	170-58
FREEELEMENT Procedure.....	170-59
FREENODE Procedure.....	170-60
FREENODELIST Procedure.....	170-61
GETATTRIBUTE Functions.....	170-62
GETATTRIBUTENODE Functions.....	170-63
GETATTRIBUTES Function.....	170-64
GETCHARSET Function.....	170-65
GETCHILDNODES Function.....	170-66
GETCHILDRENBYTAGNAME Functions.....	170-67
GETDATA Functions.....	170-68
GETDOCTYPE Function.....	170-69
GETDOCUMENTELEMENT Function.....	170-70
GETELEMENTSBYTAGNAME Functions.....	170-71
GETENTITIES Function.....	170-72
GETEXPANDEDNAME Procedure and Functions.....	170-73
GETFIRSTCHILD Function.....	170-74
GETIMPLEMENTATION Function.....	170-75
GETLASTCHILD Function.....	170-76
GETLENGTH Functions.....	170-77
GETLOCALNAME Procedure and Functions.....	170-78
GETNAME Functions.....	170-79
GETNAMEDITEM Function.....	170-80
GETNAMESPACE Procedure and Functions.....	170-81
GETNEXTSIBLING Function.....	170-82
GETNODETYPE Function.....	170-83
GETNODENAME Function.....	170-84
GETNODEVALUE Function.....	170-85
GETNODEVALUEASBINARYSTREAM Function & Procedure.....	170-86
GETNODEVALUEASCHARACTERSTREAM Function & Procedure.....	170-87
GETNOTATIONNAME Function.....	170-88
GETNOTATIONS Function.....	170-89
GETTARGET Function.....	170-90
GETOWNERDOCUMENT Function.....	170-91

GETOWNERELEMENT Function .....	170-92
GETPARENTNODE Function.....	170-93
GETPREFIX Function .....	170-94
GETPREVIOUSIBLING Function.....	170-95
GETPUBLICID Functions .....	170-96
GETQUALIFIEDNAME Functions .....	170-97
GETSCHEMANODE Function .....	170-98
GETSPECIFIED Function.....	170-99
GETSTANDALONE Function .....	170-100
GETSYSTEMID Functions .....	170-101
GETTAGNAME Function.....	170-102
GETVALUE Function .....	170-103
GETVERSION Function .....	170-104
GETXMLTYPE Function .....	170-105
HASATTRIBUTE Functions .....	170-106
HASATTRIBUTES Function.....	170-107
HASCHILDNODES Function .....	170-108
HASFEATURE Function .....	170-109
IMPORTNODE Function .....	170-110
INSERTBEFORE Function .....	170-111
INSERTDATA Procedure .....	170-112
ISNULL Functions .....	170-113
ITEM Functions .....	170-116
MAKEATTR Function.....	170-117
MAKECDATASECTION Function .....	170-118
MAKECHARACTERDATA Function .....	170-119
MAKECOMMENT Function.....	170-120
MAKEDOCUMENT Function.....	170-121
MAKEDOCUMENTFRAGMENT Function .....	170-122
MAKEDOCUMENTTYPE Function.....	170-123
MAKEELEMENT Function .....	170-124
MAKEENTITY Function .....	170-125
MAKEENTITYREFERENCE Function.....	170-126
MAKENODE Functions .....	170-127
MAKENOTATION Function .....	170-130
MAKEPROCESSINGINSTRUCTION Function .....	170-131
MAKETEXT Function.....	170-132
NEWDOMDOCUMENT Functions .....	170-133
NORMALIZE Procedure.....	170-134
REMOVEATTRIBUTE Procedures.....	170-135
REMOVEATTRIBUTENODE Function .....	170-136
REMOVECHILD Function.....	170-137
REMOVENAMEDITEM Function.....	170-138
REPLACECHILD Function.....	170-139
REPLACEDATA Procedure .....	170-140
RESOLVENAMESPACEPREFIX Function .....	170-141
SETATTRIBUTE Procedures .....	170-142

SETATTRIBUTENODE Functions.....	170-143
SETCHARSET Procedure.....	170-144
SETDATA Procedures.....	170-145
SETDOCTYPE Procedure.....	170-146
SETNAMEDITEM Function.....	170-147
SETNODEVALUE Procedure.....	170-148
SETNODEVALUEASBINARYSTREAM Function & Procedure.....	170-149
SETNODEVALUEASCHARACTERSTREAM Function & Procedure.....	170-150
SETPREFIX Procedure.....	170-151
SETSTANDALONE Procedure.....	170-152
SETVALUE Procedure.....	170-153
SETVERSION Procedure.....	170-154
SPLITTEXT Function.....	170-155
SUBSTRINGDATA Function.....	170-156
USEBINARYSTREAM Function.....	170-157
WRITETOBUFFER Procedures.....	170-158
WRITETOCLOB Procedures.....	170-159
WRITETOFILE Procedures.....	170-160

## 171 DBMS\_XMLGEN

Using DBMS_XMLGEN.....	171-2
Security Model.....	171-3
<b>Summary of DBMS_XMLGEN Subprograms.....</b>	<b>171-4</b>
CLOSECONTEXT Procedure.....	171-5
CONVERT Functions.....	171-6
GETNUMROWSPROCESSED Function.....	171-7
GETXML Functions.....	171-8
GETXMLTYPE Functions.....	171-9
NEWCONTEXT Functions.....	171-10
NEWCONTEXTFROMHIERARCHY Function.....	171-11
RESTARTQUERY Procedure.....	171-12
SETCONVERTSPECIALCHARS Procedure.....	171-13
SETMAXROWS Procedure.....	171-14
SETNULLHANDLING Procedure.....	171-15
SETROWSETTAG Procedure.....	171-16
SETROWTAG Procedure.....	171-17
SETSKIPROWS Procedure.....	171-18
USEITEMTAGSFORCOLL Procedure.....	171-19
USENULLATTRIBUTEINDICATOR Procedure.....	171-20

## 172 DBMS\_XMLINDEX

Using DBMS_XMLINDEX.....	172-2
Overview.....	172-3
Security Model.....	172-4
<b>Summary of DBMS_XMLINDEX Subprograms.....</b>	<b>172-5</b>
CREATEDATEINDEX Procedure.....	172-6
CREATENUMBERINDEX Procedure.....	172-7

DROPPARAMETER Procedure .....	172-8
MODIFYPARAMETER Procedure .....	172-9
REGISTERPARAMETER Procedure .....	172-10
SYNCINDEX Procedure .....	172-11

## 173 DBMS\_XMLPARSER

Using DBMS_XMLPARSER .....	173-2
Security Model .....	173-3
Summary of DBMS_XMLPARSER Subprograms .....	173-4
FREEPARSER .....	173-5
GETDOCTYPE .....	173-6
GETDOCUMENT .....	173-7
GETRELEASEVERSION .....	173-8
GETVALIDATIONMODE .....	173-9
NEWPARSER .....	173-10
PARSE .....	173-11
PARSEBUFFER .....	173-12
PARSECLOB .....	173-13
PARSEDTD .....	173-14
PARSEDTDBUFFER .....	173-15
PARSEDTDCLOB .....	173-16
SETBASEDIR .....	173-17
SETDOCTYPE .....	173-18
SETERRORLOG .....	173-19
SETPRESERVEWHITESPACE .....	173-20
SETVALIDATIONMODE .....	173-21
SHOWWARNINGS .....	173-22

## 174 DBMS\_XMLQUERY

Using DBMS_XMLQUERY .....	174-2
Security Model .....	174-3
Constants .....	174-4
Types .....	174-5
Summary of DBMS_XMLQUERY Subprograms .....	174-6
CLOSECONTEXT .....	174-8
GETDTD .....	174-9
GETEXCEPTIONCONTENT .....	174-10
GETNUMROWSPROCESSED .....	174-11
GETVERSION .....	174-12
GETXML .....	174-13
NEWCONTEXT .....	174-14
PROPAGATEORIGINALEXCEPTION .....	174-15
REMOVEXSLTPARAM .....	174-16
SETBINDVALUE .....	174-17
SETCOLLIDATTRNAME .....	174-18
SETDATAHEADER .....	174-19
SETDATEFORMAT .....	174-20



SETENCODINGTAG .....	174-21
SETERRORTAG .....	174-22
SETMAXROWS .....	174-23
SETMETAHEADER.....	174-24
SETRAISEEXCEPTION .....	174-25
SETRAISENOROWSEXCEPTION.....	174-26
SETROWIDATTRNAME .....	174-27
SETROWIDATTRVALUE.....	174-28
SETROWSETTAG .....	174-29
SETROWTAG .....	174-30
SETSKIPROWS.....	174-31
SETSQLTOXMLNAMEESCAPING .....	174-32
SETSTYLESHEETHEADER.....	174-33
SETTAGCASE.....	174-34
SETXSLT .....	174-35
SETXSLTPARAM.....	174-36
USENULLATTRIBUTEINDICATOR.....	174-37
USETYPEFORCOLLELEMTAG .....	174-38

## 175 DBMS\_XMLSAVE

<b>Using DBMS_XMLSAVE.....</b>	<b>175-2</b>
Security Model.....	175-3
Constants .....	175-4
Types .....	175-5
<b>Summary of DBMS_XMLSAVE Subprograms .....</b>	<b>175-6</b>
CLEARKEYCOLUMNLIST .....	175-7
CLEARUPDATECOLUMNLIST.....	175-8
CLOSECONTEXT .....	175-9
DELETEXML .....	175-10
GETEXCEPTIONCONTENT.....	175-11
INSERTXML .....	175-12
NEWCONTEXT.....	175-13
PROPAGATEORIGINALEXCEPTION .....	175-14
REMOVEXSLTPARAM.....	175-15
SETBATCHSIZE.....	175-16
SETCOMMITBATCH .....	175-17
SETDATEFORMAT .....	175-18
SETIGNORECASE .....	175-19
SETKEYCOLUMN .....	175-20
SETPRESERVEWHITESPACE .....	175-21
SETROWTAG .....	175-22
SETSQLTOXMLNAMEESCAPING .....	175-23
SETUPDATECOLUMN .....	175-24
SETXSLT .....	175-25
SETXSLTPARAM .....	175-26
UPDATEXML .....	175-27

---

## 176 DBMS\_XMLSCHEMA

<b>Using DBMS_XMLSCHEMA</b> .....	176-2
Overview .....	176-3
Security Model.....	176-4
Constants.....	176-5
Views.....	176-7
Operational Notes .....	176-8
<b>Summary of DBMS_XMLSCHEMA Subprograms</b> .....	176-9
COMPILESCHEMA Procedure .....	176-10
COPYEVOLVE Procedure .....	176-11
DELETESCHEMA Procedure.....	176-13
GENERATESCHEMA Function .....	176-14
GENERATESCHEMAS Function .....	176-15
INPLACEEVOLVE Procedure .....	176-16
PURGESCHEMA Procedure .....	176-18
REGISTERSCHEMA Procedures .....	176-19
REGISTERURI Procedure .....	176-23

## 177 DBMS\_XMLSTORE

<b>Using DBMS_XMLSTORE</b> .....	177-2
Security Model.....	177-3
Types .....	177-4
<b>Summary of DBMS_XMLSTORE Subprograms</b> .....	177-5
CLEARKEYCOLUMNLIST .....	177-6
CLEARUPDATECOLUMNLIST.....	177-7
CLOSECONTEXT .....	177-8
DELETEXML .....	177-9
INSERTXML .....	177-10
NEWCONTEXT.....	177-11
SETKEYCOLUMN .....	177-12
SETROWTAG .....	177-13
SETUPDATECOLUMN .....	177-14
UPDATEXML .....	177-15

## 178 DBMS\_XMLTRANSLATIONS

<b>Using DBMS_XMLTRANSLATIONS</b> .....	178-2
Security Model.....	178-3
<b>Summary of DBMS_XMLTRANSLATIONS Subprograms</b> .....	178-4
DISABLETRANSLATION Procedure .....	178-5
ENABLETRANSLATION Procedure.....	178-6
EXTRACTXLIFF Function & Procedure .....	178-7
GETBASEDOCUMENT Function.....	178-10
MERGEXLIFF Functions .....	178-11
SETSOURCELANG Function.....	178-14
TRANSLATEXML Function .....	178-16
UPDATETRANSLATION Function.....	178-17

---

## 179 DBMS\_XPLAN

Using DBMS_XPLAN.....	179-2
Overview .....	179-3
Security Model.....	179-4
Examples .....	179-5
<b>Summary of DBMS_XPLAN Subprograms .....</b>	<b>179-9</b>
DISPLAY Function.....	179-10
DISPLAY_AWR Function .....	179-13
DISPLAY_CURSOR Function .....	179-16
DISPLAY_PLAN Function .....	179-19
DISPLAY_SQL_PLAN_BASELINE Function .....	179-22
DISPLAY_SQLSET Function .....	179-24

## 180 DBMS\_XSLPROCESSOR

Using DBMS_XSLPROCESSOR.....	180-2
Overview .....	180-3
Security Model.....	180-4
<b>Summary of DBMS_XSLPROCESSOR Subprograms.....</b>	<b>180-5</b>
CLOB2FILE Procedure .....	180-6
FREEPROCESSOR Procedure .....	180-7
FREESTYLESHEET Procedure.....	180-8
NEWPROCESSOR Function.....	180-9
NEWSTYLESHEET Functions.....	180-10
PROCESSXSL Functions and Procedures.....	180-11
READ2CLOB Function.....	180-13
REMOVEPARAM Procedure .....	180-14
RESETPARAMS Procedure .....	180-15
SELECTNODES Function .....	180-16
SELECTSINGLENODE Function.....	180-17
SETERRORLOG Procedure .....	180-18
SETPARAM Procedure .....	180-19
SHOWWARNINGS Procedure .....	180-20
TRANSFORMNODE Function .....	180-21
VALUEOF Function and Procedure.....	180-22

## 181 DEBUG\_EXTPROC

Using DEBUG_EXTPROC .....	181-2
Security Model.....	181-3
Operational Notes .....	181-4
Rules and Limits.....	181-5
<b>Summary of DEBUG_EXTPROC Subprograms .....</b>	<b>181-6</b>
STARTUP_EXTPROC_AGENT Procedure .....	181-7

## 182 HTF

Using HTF.....	182-2
Deprecated Subprograms .....	182-3

Operational Notes .....	182-4
Rules and Limits.....	182-5
Examples .....	182-6
<b>Summary of Tags .....</b>	<b>182-7</b>
<b>Summary of HTF Subprograms .....</b>	<b>182-10</b>
ADDRESS Function .....	182-16
ANCHOR Function.....	182-17
ANCHOR2 Function.....	182-18
APPLETCLOSE Function.....	182-19
APPLETOPEN Function .....	182-20
AREA Function.....	182-21
BASE Function.....	182-22
BASEFONT Function.....	182-23
BGSOUND Function.....	182-24
BIG Function .....	182-25
BLOCKQUOTECLOSE Function .....	182-26
BLOCKQUOTEOPEN Function.....	182-27
BODYCLOSE Function.....	182-28
BODYOPEN Function .....	182-29
BOLD Function.....	182-30
BR Function.....	182-31
CENTER Function.....	182-32
CENTERCLOSE Function.....	182-33
CENTEROPEN Function .....	182-34
CITE Function.....	182-35
CODE Function .....	182-36
COMMENT Function .....	182-37
DFN Function .....	182-38
DIRLISTCLOSE Function.....	182-39
DIRLISTOPEN Function .....	182-40
DIV Function.....	182-41
DLISTCLOSE Function.....	182-42
DLISTDEF Function.....	182-43
DLISTOPEN Function .....	182-44
DLISTTERM Function .....	182-45
EM Function.....	182-46
EMPHASIS Function .....	182-47
ESCAPE_SC Function.....	182-48
ESCAPE_URL Function .....	182-49
FONTCLOSE Function.....	182-50
FONTOPEN Function .....	182-51
FORMAT_CELL Function .....	182-52
FORMCHECKBOX Function.....	182-53
FORMCLOSE Function .....	182-54
FORMFILE Function.....	182-55
FORMHIDDEN Function .....	182-56
FORMIMAGE Function .....	182-57

FORMOPEN Function.....	182-58
FORMPASSWORD Function .....	182-59
FORMRADIO Function.....	182-60
FORMRESET Function .....	182-61
FORMSELECTCLOSE Function .....	182-62
FORMSELECTOPEN Function .....	182-63
FORMSELECTOPTION Function.....	182-64
FORMSUBMIT Function .....	182-65
FORMTEXT Function .....	182-66
FORMTEXTAREA Function.....	182-67
FORMTEXTAREA2 Function.....	182-68
FORMTEXTAREACLOSE Function.....	182-69
FORMTEXTAREAOPEN Function .....	182-70
FORMTEXTAREAOPEN2 Function .....	182-71
FRAME Function.....	182-72
FRAMESETCLOSE Function.....	182-73
FRAMESETOPEN Function .....	182-74
HEADCLOSE Function .....	182-75
HEADER Function.....	182-76
HEADOPEN Function.....	182-77
HR Function.....	182-78
HTMLCLOSE Function .....	182-79
HTMLOPEN Function.....	182-80
IMG Function.....	182-81
IMG2 Function.....	182-82
ISINDEX Function.....	182-83
ITALIC Function .....	182-84
KBD Function.....	182-85
KEYBOARD Function .....	182-86
LINE Function .....	182-87
LINKREL Function .....	182-88
LINKREV Function.....	182-89
LISTHEADER Function.....	182-90
LISTINGCLOSE Function.....	182-91
LISTINGOPEN Function.....	182-92
LISTITEM Function.....	182-93
MAILTO Function.....	182-94
MAPCLOSE Function.....	182-95
MAPOPEN Function .....	182-96
MENULISTCLOSE Function .....	182-97
MENULISTOPEN Function.....	182-98
META Function .....	182-99
NL Function .....	182-100
NOBR Function .....	182-101
NOFRAMESCLOSE Function .....	182-102
NOFRAMESOPEN Function.....	182-103
OLISTCLOSE Function .....	182-104

OLISTOPEN Function .....	182-105
PARA Function.....	182-106
PARAGRAPH Function .....	182-107
PARAM Function.....	182-108
PLAINTEXT Function .....	182-109
PRECLOSE Function .....	182-110
PREOPEN Function .....	182-111
PRINT Functions .....	182-112
PRN Functions .....	182-113
S Function.....	182-114
SAMPLE Function.....	182-115
SCRIPT Function .....	182-116
SMALL Function .....	182-117
STRIKE Function .....	182-118
STRONG Function .....	182-119
STYLE Function .....	182-120
SUB Function .....	182-121
SUP Function .....	182-122
TABLECAPTION Function .....	182-123
TABLECLOSE Function .....	182-124
TABLEDATA Function .....	182-125
TABLEHEADER Function .....	182-126
TABLEOPEN Function.....	182-127
TABLEROWCLOSE Function .....	182-128
TABLEROWOPEN Function.....	182-129
TELETYPE Function .....	182-130
TITLE Function.....	182-131
ULISTCLOSE Function.....	182-132
ULISTOPEN Function .....	182-133
UNDERLINE Function.....	182-134
VARIABLE Function .....	182-135
WBR Function.....	182-136

## 183 HTP

<b>Using HTP</b> .....	183-2
Operational Notes .....	183-3
Rules and Limits.....	183-4
Examples .....	183-5
<b>Summary of Tags</b> .....	183-6
<b>Summary of HTP Subprograms</b> .....	183-9
ADDRESS Procedure .....	183-15
ANCHOR Procedure .....	183-16
ANCHOR2 Procedure .....	183-17
APPLETCLOSE Procedure .....	183-18
APPLETOPEN Procedure .....	183-19
AREA Procedure .....	183-20
BASE Procedure .....	183-21

---

BASEFONT Procedure .....	183-22
BGSOUND Procedure .....	183-23
BIG Procedure.....	183-24
BLOCKQUOTECLOSE Procedure .....	183-25
BLOCKQUOTEOPEN Procedure .....	183-26
BODYCLOSE Procedure .....	183-27
BODYOPEN Procedure.....	183-28
BOLD Procedure .....	183-29
BR Procedure .....	183-30
CENTER Procedure .....	183-31
CENTERCLOSE Procedure .....	183-32
CENTEROPEN Procedure.....	183-33
CITE Procedure .....	183-34
CODE Procedure.....	183-35
COMMENT Procedure.....	183-36
DFN Procedure.....	183-37
DIRLISTCLOSE Procedure.....	183-38
DIRLISTOPEN Procedure.....	183-39
DIV Procedure .....	183-40
DLISTCLOSE Procedure .....	183-41
DLISTDEF Procedure .....	183-42
DLISTOPEN Procedure.....	183-43
DLISTTERM Procedure.....	183-44
EM Procedure .....	183-45
EMPHASIS Procedure.....	183-46
ESCAPE_SC Procedure .....	183-47
FONTCLOSE Procedure .....	183-48
FONTOPEN Procedure.....	183-49
FORMCHECKBOX Procedure .....	183-50
FORMCLOSE Procedure.....	183-51
FORMOPEN Procedure .....	183-52
FORMFILE Procedure .....	183-53
FORMHIDDEN Procedure.....	183-54
FORMIMAGE Procedure.....	183-55
FORMPASSWORD Procedure .....	183-56
FORMRADIO Procedure .....	183-57
FORMRESET Procedure.....	183-58
FORMSELECTCLOSE Procedure.....	183-59
FORMSELECTOPEN Procedure.....	183-60
FORMSELECTIONOPTION Procedure .....	183-61
FORMSUBMIT Procedure .....	183-62
FORMTEXT Procedure.....	183-63
FORMTEXTAREA Procedure .....	183-64
FORMTEXTAREA2 Procedure .....	183-65
FORMTEXTAREACLOSE Procedure .....	183-66
FORMTEXTAREAOPEN Procedure.....	183-67
FORMTEXTAREAOPEN2 Procedure .....	183-68

FRAME Procedure .....	183-69
FRAMESETCLOSE Procedure .....	183-70
FRAMESETOPEN Procedure.....	183-71
HEADCLOSE Procedure.....	183-72
HEADER Procedure .....	183-73
HEADOPEN Procedure .....	183-74
HR Procedure .....	183-75
HTMLCLOSE Procedure .....	183-76
HTMLOPEN Procedure .....	183-77
IMG Procedure .....	183-78
IMG2 Procedure .....	183-79
ISINDEX Procedure .....	183-80
ITALIC Procedure .....	183-81
KBD Procedure .....	183-82
KEYBOARD Procedure .....	183-83
LINE Procedure .....	183-84
LINKREL Procedure.....	183-85
LINKREV Procedure .....	183-86
LISTHEADER Procedure .....	183-87
LISTINGCLOSE Procedure .....	183-88
LISTINGOPEN Procedure .....	183-89
LISTITEM Procedure .....	183-90
MAILTO Procedure .....	183-91
MAPCLOSE Procedure .....	183-92
MAPOPEN Procedure.....	183-93
MENULISTCLOSE Procedure .....	183-94
MENULISTOPEN Procedure .....	183-95
META Procedure.....	183-96
NL Procedure.....	183-97
NOBR Procedure.....	183-98
NOFRAMESCLOSE Procedure.....	183-99
NOFRAMESOPEN Procedure .....	183-100
OLISTCLOSE Procedure .....	183-101
OLISTOPEN Procedure.....	183-102
PARA Procedure .....	183-103
PARAGRAPH Procedure.....	183-104
PARAM Procedure .....	183-105
PLAINTEXT Procedure.....	183-106
PRECLOSE Procedure.....	183-107
PREOPEN Procedure.....	183-108
PRINT Procedures.....	183-109
PRINTS Procedure .....	183-110
PRN Procedures .....	183-111
PS Procedure .....	183-112
S Procedure .....	183-113
SAMPLE Procedure .....	183-114
SCRIPT Procedure.....	183-115



SMALL Procedure.....	183-116
STRIKE Procedure.....	183-117
STRONG Procedure.....	183-118
STYLE Procedure .....	183-119
SUB Procedure.....	183-120
SUP Procedure.....	183-121
TABLECAPTION Procedure .....	183-122
TABLECLOSE Procedure.....	183-123
TABLEDATA Procedure.....	183-124
TABLEHEADER Procedure .....	183-125
TABLEOPEN Procedure .....	183-126
TABLEROWCLOSE Procedure.....	183-127
TABLEROWOPEN Procedure .....	183-128
TELETYPE Procedure.....	183-129
TITLE Procedure .....	183-130
ULISTCLOSE Procedure .....	183-131
ULISTOPEN Procedure.....	183-132
UNDERLINE Procedure .....	183-133
VARIABLE Procedure.....	183-134
WBR Procedure .....	183-135
<b>184 ORD_DICOM</b>	
Documentation of ORD_DICOM.....	184-2
<b>185 ORD_DICOM_ADMIN</b>	
Documentation of ORD_DICOM_ADMIN.....	185-2
<b>186 OWA_CACHE</b>	
Using OWA_CACHE .....	186-2
Constants.....	186-3
<b>Summary of OWA_CACHE Subprograms</b> .....	186-4
DISABLE Procedure .....	186-5
GET_ETAG Function.....	186-6
GET_LEVEL Function .....	186-7
SET_CACHE Procedure.....	186-8
SET_EXPIRES Procedure .....	186-9
SET_NOT_MODIFIED Procedure.....	186-10
SET_SURROGATE_CONTROL Procedure.....	186-11
<b>187 OWA_COOKIE</b>	
Using OWA_COOKIE .....	187-2
Overview .....	187-3
Types.....	187-4
Rules and Limits.....	187-5
<b>Summary of OWA_COOKIE Subprograms</b> .....	187-6
GET Function.....	187-7

GET_ALL Procedure.....	187-8
REMOVE Procedure .....	187-9
SEND procedure.....	187-10
<b>188 OWA_CUSTOM</b>	
Using OWA_CUSTOM .....	188-2
Constants .....	188-3
<b>Summary of OWA_CUSTOM Subprograms</b> .....	188-4
AUTHORIZE Function.....	188-5
<b>189 OWA_IMAGE</b>	
Using OWA_IMAGE .....	189-2
Overview .....	189-3
Types .....	189-4
Variables .....	189-5
Examples .....	189-6
<b>Summary of OWA_IMAGE Subprograms</b> .....	189-7
GET_X Function .....	189-8
GET_Y Function .....	189-9
<b>190 OWA_OPT_LOCK</b>	
Using OWA_OPT_LOCK.....	190-2
Overview .....	190-3
Types .....	190-4
<b>Summary of OWA_OPT_LOCK Subprograms</b> .....	190-5
CHECKSUM Functions.....	190-6
GET_ROWID Function.....	190-7
STORE_VALUES Procedure.....	190-8
VERIFY_VALUES Function.....	190-9
<b>191 OWA_PATTERN</b>	
Using OWA_PATTERN .....	191-2
Types .....	191-3
Operational Notes .....	191-4
<b>Summary of OWA_PATTERN Subprograms</b> .....	191-6
AMATCH Function .....	191-7
CHANGE Functions and Procedures .....	191-9
GETPAT Procedure.....	191-11
MATCH Function .....	191-12
<b>192 OWA_SEC</b>	
Using OWA_SEC .....	192-2
Operational Notes .....	192-3
<b>Summary of OWA_SEC Subprograms</b> .....	192-4
GET_CLIENT_HOSTNAME Function .....	192-5

GET_CLIENT_IP Function .....	192-6
GET_PASSWORD Function .....	192-7
GET_USER_ID Function .....	192-8
SET_AUTHORIZATION Procedure .....	192-9
SET_PROTECTION_REALM Procedure.....	192-10
<b>193 OWA_TEXT</b>	
Using OWA_TEXT .....	193-2
Types .....	193-3
<b>Summary of OWA_TEXT Subprograms</b> .....	193-4
ADD2MULTI Procedure .....	193-5
NEW_ROW_LIST Function and Procedure .....	193-6
PRINT_MULTI Procedure .....	193-7
PRINT_ROW_LIST Procedure .....	193-8
STREAM2MULTI Procedure.....	193-9
<b>194 OWA_UTIL</b>	
Using OWA_UTIL .....	194-2
Overview .....	194-3
Types .....	194-4
<b>Summary of OWA_UTIL Subprograms</b> .....	194-5
BIND_VARIABLES Function .....	194-6
CALENDARPRINT Procedures .....	194-7
CELLSPRINT Procedures .....	194-8
CHOOSE_DATE Procedure .....	194-10
GET_CGI_ENV Function .....	194-11
GET_OWA_SERVICE_PATH Function.....	194-12
GET_PROCEDURE Function .....	194-13
HTTP_HEADER_CLOSE Procedure.....	194-14
LISTPRINT Procedure.....	194-15
MIME_HEADER Procedure .....	194-16
PRINT_CGI_ENV Procedure .....	194-17
REDIRECT_URL Procedure .....	194-18
SHOWPAGE Procedure.....	194-19
SHOWSOURCE Procedure.....	194-20
SIGNATURE procedure.....	194-21
STATUS_LINE Procedure.....	194-22
TABLEPRINT Function.....	194-23
TODATE Function .....	194-26
WHO_CALLED_ME Procedure .....	194-27
<b>195 SDO_CS</b>	
Documentation of SDO_CS .....	195-2
<b>196 SDO_CSW_PROCESS</b>	
Documentation of SDO_CSW_PROCESS .....	196-2

---

<b>197</b>	<b>SDO_GCDR</b>	
	Documentation of SDO_GCDR .....	197-2
<b>198</b>	<b>SDO_GEOM</b>	
	Documentation of SDO_GEOM .....	198-2
<b>199</b>	<b>SDO_GEOR</b>	
	Documentation of SDO_GEOR.....	199-2
<b>200</b>	<b>SDO_GEOR_ADMIN</b>	
	Documentation of SDO_GEOR_ADMIN .....	200-2
<b>201</b>	<b>SDO_GEOR_UTL</b>	
	Documentation of SDO_GEOR_UTL .....	201-2
<b>202</b>	<b>SDO_LRS</b>	
	Documentation of SDO_LRS.....	202-2
<b>203</b>	<b>SDO_MIGRATE</b>	
	Documentation of SDO_MIGRATE.....	203-2
<b>204</b>	<b>SDO_NET</b>	
	Documentation of SDO_NET .....	204-2
<b>205</b>	<b>SDO_NET_MEM</b>	
	Documentation of SDO_NET_MEM.....	205-2
<b>206</b>	<b>SDO_OLS</b>	
	Documentation of SDO_OLS .....	206-2
<b>207</b>	<b>SDO_PC_PKG</b>	
	Documentation of SDO_PC_PKG .....	207-2
<b>208</b>	<b>SDO_SAM</b>	
	Documentation of SDO_SAM.....	208-2
<b>209</b>	<b>SDO_TIN_PKG</b>	
	Documentation of SDO_TIN_PKG .....	209-2
<b>210</b>	<b>SDO_TOPO</b>	
	Documentation of SDO_TOPO.....	210-2

<b>211</b>	<b>SDO_TOPO_MAP</b>	
	Documentation of SDO_TOPO_MAP .....	211-2
<b>212</b>	<b>SDO_TUNE</b>	
	Documentation of SDO_TUNE .....	212-2
<b>213</b>	<b>SDO_UTIL</b>	
	Documentation of SDO_UTIL .....	213-2
<b>214</b>	<b>SDO_WFS_LOCK</b>	
	Documentation of SDO_WFS_LOCK .....	214-2
<b>215</b>	<b>SDO_WFS_PROCESS</b>	
	Documentation of SDO_WFS_PROCESS .....	215-2
<b>216</b>	<b>SEM_APIS</b>	
	Documentation of SEM_APIS .....	216-2
<b>217</b>	<b>SEM_PERF</b>	
	Documentation of SEM_PERF .....	217-2
<b>218</b>	<b>SEM_RDFCTX</b>	
	Documentation of SEM_RDFCTX .....	218-2
<b>219</b>	<b>SEM_RDFSFA</b>	
	Documentation of SEM_RDFSFA .....	219-2
<b>220</b>	<b>UTL_COLL</b>	
	Summary of UTL_COLL Subprograms .....	220-2
	IS_LOCATOR Function .....	220-3
<b>221</b>	<b>UTL_COMPRESS</b>	
	Using UTL_COMPRESS .....	221-2
	Constants .....	221-3
	Exceptions .....	221-4
	Operational Notes .....	221-5
	Summary of UTL_COMPRESS Subprograms .....	221-6
	ISOPEN Function .....	221-7
	LZ_COMPRESS Functions and Procedures .....	221-8
	LZ_COMPRESS_ADD Procedure .....	221-10
	LZ_COMPRESS_CLOSE .....	221-11
	LZ_COMPRESS_OPEN .....	221-12
	LZ_UNCOMPRESS Functions and Procedures .....	221-13

LZ_UNCOMPRESS_EXTRACT Procedure.....	221-14
LZ_UNCOMPRESS_OPEN Function.....	221-15
LZ_UNCOMPRESS_CLOSE Procedure .....	221-16

## 222 UTL\_ENCODE

<b>Summary of UTL_ENCODE Subprograms.....</b>	<b>222-2</b>
BASE64_DECODE Function.....	222-3
BASE64_ENCODE Function.....	222-4
MIMEHEADER_DECODE Function .....	222-5
MIMEHEADER_ENCODE Function .....	222-6
QUOTED_PRINTABLE_DECODE Function.....	222-7
QUOTED_PRINTABLE_ENCODE Function.....	222-8
TEXT_DECODE Function.....	222-9
TEXT_ENCODE Function.....	222-10
UUDECODE Function.....	222-11
UUENCODE Function .....	222-12

## 223 UTL\_FILE

<b>Using UTL_FILE .....</b>	<b>223-2</b>
Security Model.....	223-3
Operational Notes .....	223-4
Rules and Limits.....	223-5
Exceptions .....	223-6
Examples .....	223-7
<b>Data Structures.....</b>	<b>223-9</b>
FILETYPE Record Type.....	223-10
<b>Summary of UTL_FILE Subprograms.....</b>	<b>223-11</b>
FCLOSE Procedure .....	223-13
FCLOSE_ALL Procedure .....	223-14
FCOPY Procedure .....	223-15
FFLUSH Procedure .....	223-16
FGETATTR Procedure.....	223-17
FGETPOS Function .....	223-18
FOPEN Function .....	223-19
FOPEN_NCHAR Function .....	223-21
FREMOVE Procedure .....	223-22
FRENAME Procedure .....	223-23
FSEEK Procedure .....	223-24
GET_LINE Procedure .....	223-25
GET_LINE_NCHAR Procedure.....	223-26
GET_RAW Procedure.....	223-27
IS_OPEN Function .....	223-28
NEW_LINE Procedure .....	223-29
PUT Procedure .....	223-30
PUT_LINE Procedure .....	223-31
PUT_LINE_NCHAR Procedure.....	223-32
PUT_NCHAR Procedure .....	223-33

PUTF Procedure .....	223-34
PUTF_NCHAR Procedure .....	223-36
PUT_RAW Procedure.....	223-37
<b>224 UTL_HTTP</b>	
<b>Using UTL_HTTP</b> .....	224-2
Overview .....	224-3
Security Model.....	224-4
Constants .....	224-5
Datatypes.....	224-9
Operational Notes .....	224-13
Exceptions .....	224-19
Examples .....	224-21
<b>Subprogram Groups</b> .....	224-25
Simple HTTP Fetches in a Single Call Subprograms .....	224-26
Session Settings Subprograms.....	224-27
HTTP Requests Subprograms .....	224-28
HTTP Request Contexts Subprograms .....	224-29
HTTP Responses Subprograms.....	224-30
HTTP Cookies Subprograms.....	224-31
HTTP Persistent Connections Subprograms.....	224-32
Error Conditions Subprograms.....	224-33
<b>Summary of UTL_HTTP Subprograms</b> .....	224-34
ADD_COOKIES Procedure .....	224-38
BEGIN_REQUEST Function.....	224-39
CLEAR_COOKIES Procedure .....	224-41
CLOSE_PERSISTENT_CONN Procedure .....	224-42
CLOSE_PERSISTENT_CONNS Procedure .....	224-43
CREATE_REQUEST_CONTEXT Function .....	224-45
DESTROY_REQUEST_CONTEXT Procedure .....	224-47
END_REQUEST Procedure .....	224-48
END_RESPONSE Procedure .....	224-49
GET_AUTHENTICATION Procedure.....	224-50
GET_BODY_CHARSET Procedure .....	224-51
GET_COOKIE_COUNT Function .....	224-52
GET_COOKIE_SUPPORT Procedure .....	224-53
GET_COOKIES Function .....	224-54
GET_DETAILED_EXCP_SUPPORT Procedure .....	224-55
GET_DETAILED_SQLCODE Function .....	224-56
GET_DETAILED_SQLERRM Function .....	224-57
GET_FOLLOW_REDIRECT Procedure .....	224-58
GET_HEADER Procedure .....	224-59
GET_HEADER_BY_NAME Procedure.....	224-60
GET_HEADER_COUNT Function .....	224-61
GET_PERSISTENT_CONN_COUNT Function.....	224-62
GET_PERSISTENT_CONN_SUPPORT Procedure.....	224-63
GET_PERSISTENT_CONNS Procedure .....	224-64

GET_PROXY Procedure.....	224-65
GET_RESPONSE Function .....	224-66
GET_RESPONSE_ERROR_CHECK Procedure .....	224-67
GET_TRANSFER_TIMEOUT Procedure.....	224-68
READ_LINE Procedure.....	224-69
READ_RAW Procedure .....	224-70
READ_TEXT Procedure .....	224-71
REQUEST Function.....	224-73
REQUEST_PIECES Function .....	224-75
SET_AUTHENTICATION Procedure.....	224-78
SET_AUTHENTICATION_FROM_WALLET Procedure .....	224-79
SET_BODY_CHARSET Procedures .....	224-81
SET_COOKIE_SUPPORT Procedures.....	224-83
SET_DETAILED_EXCP_SUPPORT Procedure .....	224-85
SET_FOLLOW_REDIRECT Procedures .....	224-86
SET_HEADER Procedure .....	224-87
SET_PERSISTENT_CONN_SUPPORT Procedure.....	224-88
SET_PROXY Procedure .....	224-90
SET_RESPONSE_ERROR_CHECK Procedure.....	224-91
SET_TRANSFER_TIMEOUT Procedure.....	224-92
SET_WALLET Procedure.....	224-93
WRITE_LINE Procedure .....	224-95
WRITE_RAW Procedure.....	224-96
WRITE_TEXT Procedure.....	224-97

## 225 UTL\_I18N

Using UTL_I18N.....	225-2
Overview .....	225-3
Security Model.....	225-4
Constants .....	225-5
Summary of UTL_I18N Subprograms.....	225-7
ESCAPE_REFERENCE Function .....	225-9
GET_COMMON_TIME_ZONES Function .....	225-10
GET_DEFAULT_CHARSET Function .....	225-11
GET_DEFAULT_ISO_CURRENCY Function.....	225-12
GET_DEFAULT_LINGUISTIC_SORT Function .....	225-13
GET_LOCAL_LANGUAGES Function .....	225-14
GET_LOCAL_LINGUISTIC_SORTS Function .....	225-15
GET_LOCAL_TERRITORIES Function .....	225-16
GET_LOCAL_TIME_ZONES Function .....	225-17
GET_TRANSLATION Function.....	225-19
MAP_CHARSET Function.....	225-20
MAP_FROM_SHORT_LANGUAGE Function .....	225-22
MAP_LANGUAGE_FROM_ISO Function.....	225-23
MAP_LOCALE_TO_ISO Function .....	225-24
MAP_TERRITORY_FROM_ISO Function.....	225-25
MAP_TO_SHORT_LANGUAGE Function.....	225-26



RAW_TO_CHAR Functions.....	225-27
RAW_TO_NCHAR Functions.....	225-29
STRING_TO_RAW Function.....	225-31
TRANSLITERATE Function.....	225-32
UNESCAPE_REFERENCE Function.....	225-34
<b>226 UTL_INADDR</b>	
Using UTL_INADDR .....	226-2
Security Model.....	226-3
Exceptions .....	226-4
Examples .....	226-5
Summary of UTL_INADDR Subprograms .....	226-6
GET_HOST_ADDRESS Function .....	226-7
GET_HOST_NAME Function .....	226-8
<b>227 UTL_IDENT</b>	
Using UTL_IDENT .....	227-2
Overview .....	227-3
Security Model.....	227-4
Constants .....	227-5
<b>228 UTL_LMS</b>	
Using UTL_LMS.....	228-2
Security Model.....	228-3
Summary of UTL_LMS Subprograms.....	228-4
FORMAT_MESSAGE Function.....	228-5
GET_MESSAGE Function .....	228-6
<b>229 UTL_MAIL</b>	
Using UTL_MAIL.....	229-2
Security Model.....	229-3
Operational Notes .....	229-4
Rules and Limits.....	229-5
Summary of UTL_MAIL Subprograms .....	229-6
SEND Procedure .....	229-7
SEND_ATTACH_RAW Procedure .....	229-8
SEND_ATTACH_VARCHAR2 Procedure.....	229-9
<b>230 UTL_MATCH</b>	
Using UTL_MATCH .....	230-2
Overview .....	230-3
Security Model.....	230-4
Summary of UTL_MATCH Subprograms .....	230-5
EDIT_DISTANCE Function .....	230-6
EDIT_DISTANCE_SIMILARITY Function.....	230-7

JARO_WINKLER Function.....	230-8
JARO_WINKLER_SIMILARITY Function .....	230-9

## 231 UTL\_NLA

<b>Using UTL_NLA</b> .....	231-2
Overview .....	231-3
Rules and Limits.....	231-4
Security Model.....	231-5
<b>Subprogram Groups</b> .....	231-6
BLAS Level 1 (Vector-Vector Operations) Subprograms .....	231-7
BLAS Level 2 (Matrix-Vector Operations) Subprograms.....	231-8
BLAS Level 3 (Matrix-Matrix Operations) Subprograms .....	231-10
LAPACK Driver Routines (Linear Equations) Subprograms.....	231-11
LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms.....	231-12
<b>Summary of UTL_NLA Subprograms</b> .....	231-13
BLAS_ASUM Functions .....	231-18
BLAS_AXPY Procedures.....	231-19
BLAS_COPY Procedures.....	231-20
BLAS_DOT Functions .....	231-21
BLAS_GBMV Procedures .....	231-22
BLAS_GEMM Procedures.....	231-25
BLAS_GEMV Procedures .....	231-27
BLAS_GER Procedures .....	231-29
BLAS_IAMAX Functions .....	231-31
BLAS_NRM2 Functions .....	231-32
BLAS_ROT Procedures .....	231-33
BLAS_ROTG Procedures .....	231-34
BLAS_SCAL Procedures .....	231-35
BLAS_SPMV Procedures .....	231-36
BLAS_SPR Procedures .....	231-38
BLAS_SPR2 Procedures .....	231-40
BLAS_SBMV Procedures .....	231-42
BLAS_SWAP Procedures .....	231-44
BLAS_SYMM Procedures .....	231-45
BLAS_SYMV Procedures .....	231-47
BLAS_SYR Procedures .....	231-49
BLAS_SYR2 Procedures .....	231-51
BLAS_SYR2K Procedures .....	231-53
BLAS_SYRK Procedures .....	231-56
BLAS_TBMV Procedures .....	231-58
BLAS_TBSV Procedures.....	231-60
BLAS_TPMV Procedures .....	231-62
BLAS_TPSV Procedures.....	231-64
BLAS_TRMM Procedures .....	231-66
BLAS_TRMV Procedures.....	231-69
BLAS_TRSM Procedures.....	231-71
BLAS_TRSV Procedures.....	231-74

LAPACK_GBSV Procedures .....	231-76
LAPACK_GEES Procedures.....	231-78
LAPACK_GELS Procedures.....	231-80
LAPACK_GESDD Procedures .....	231-82
LAPACK_GESV Procedures .....	231-85
LAPACK_GESVD Procedures .....	231-87
LAPACK_GEEV Procedures .....	231-90
LAPACK_GTSV Procedures .....	231-93
LAPACK_PBSV Procedures .....	231-95
LAPACK_POSV Procedures .....	231-97
LAPACK_PPSV Procedures .....	231-99
LAPACK_PTSV Procedures .....	231-101
LAPACK_SBEV Procedures.....	231-103
LAPACK_SBEVD Procedures.....	231-105
LAPACK_SPEV Procedures .....	231-107
LAPACK_SPEVD Procedures.....	231-109
LAPACK_SPSV Procedures .....	231-111
LAPACK_STEV Procedures .....	231-113
LAPACK_STEVD Procedures.....	231-115
LAPACK_SYEV Procedures.....	231-117
LAPACK_SYEVD Procedures.....	231-119
LAPACK_SYSV Procedures .....	231-121

## 232 UTL\_RAW

Using UTL_RAW .....	232-2
Overview .....	232-3
Operational Notes .....	232-4
Summary of UTL_RAW Subprograms.....	232-5
BIT_AND Function .....	232-7
BIT_COMPLEMENT Function.....	232-8
BIT_OR Function.....	232-9
BIT_XOR Function .....	232-10
CAST_FROM_BINARY_DOUBLE Function .....	232-11
CAST_FROM_BINARY_FLOAT Function.....	232-13
CAST_FROM_BINARY_INTEGER Function .....	232-14
CAST_FROM_NUMBER Function.....	232-15
CAST_TO_BINARY_DOUBLE Function.....	232-16
CAST_TO_BINARY_FLOAT Function .....	232-18
CAST_TO_BINARY_INTEGER Function.....	232-20
CAST_TO_NUMBER Function .....	232-21
CAST_TO_NVARCHAR2 Function.....	232-22
CAST_TO_RAW Function.....	232-23
CAST_TO_VARCHAR2 Function .....	232-24
COMPARE Function.....	232-25
CONCAT Function .....	232-26
CONVERT Function .....	232-27
COPIES Function.....	232-28

LENGTH Function .....	232-29
OVERLAY Function.....	232-30
REVERSE Function .....	232-32
SUBSTR Function .....	232-33
TRANSLATE Function.....	232-35
TRANSLITERATE Function.....	232-37
XRANGE Function.....	232-39
<b>233 UTL_RECOMP</b>	
Using UTL_RECOMP .....	233-2
Overview .....	233-3
Operational Notes .....	233-4
Examples .....	233-5
<b>Summary of UTL_RECOMP Subprograms</b> .....	233-6
RECOMP_PARALLEL Procedure.....	233-7
RECOMP_SERIAL Procedure .....	233-8
<b>234 UTL_REF</b>	
Using UTL_REF .....	234-2
Overview .....	234-3
Security Model.....	234-4
Types .....	234-5
Exceptions .....	234-6
<b>Summary of UTL_REF Subprograms</b> .....	234-7
DELETE_OBJECT Procedure .....	234-8
LOCK_OBJECT Procedure .....	234-10
SELECT_OBJECT Procedure .....	234-11
UPDATE_OBJECT Procedure .....	234-12
<b>235 UTL_SMTP</b>	
Using UTL_SMTP .....	235-2
Overview .....	235-3
Security Model.....	235-4
Constants .....	235-5
Types .....	235-6
Reply Codes .....	235-8
Exceptions .....	235-10
Rules and Limits.....	235-11
Examples .....	235-12
<b>Summary of UTL_SMTP Subprograms</b> .....	235-13
AUTH Function and Procedure .....	235-14
CLOSE_CONNECTION Procedure .....	235-16
CLOSE_DATA Function and Procedure .....	235-17
COMMAND Function and Procedure .....	235-18
COMMAND_REPLIES Function .....	235-19
DATA Function and Procedure .....	235-20

EHLO Function and Procedure.....	235-21
HELO Function and Procedure.....	235-22
HELP Function .....	235-23
MAIL Function and Procedure .....	235-24
NOOP Function and Procedure .....	235-25
OPEN_CONNECTION Functions.....	235-26
OPEN_DATA Function and Procedure.....	235-28
QUIT Function and Procedure.....	235-29
RCPT Function.....	235-30
RSET Function and Procedure .....	235-31
STARTTLS Function and Procedure .....	235-32
VERFY Function .....	235-33
WRITE_DATA Procedure.....	235-34
WRITE_RAW_DATA Procedure.....	235-35

## 236 UTL\_SPADV

<b>Using UTL_SPADV</b> .....	236-2
Overview .....	236-3
Security Model.....	236-4
Operational Notes .....	236-5
<b>Summary of UTL_SPADV Subprograms</b> .....	236-14
ALTER_MONITORING Procedure.....	236-15
COLLECT_STATS Procedure.....	236-17
IS_MONITORING Function.....	236-19
SHOW_STATS Procedure.....	236-20
START_MONITORING Procedure .....	236-23
STOP_MONITORING Procedure.....	236-26

## 237 UTL\_TCP

<b>Using UTL_TCP</b> .....	237-2
Overview .....	237-3
Security Model.....	237-4
Types.....	237-5
Exceptions .....	237-7
Rules and Limits.....	237-8
Examples .....	237-9
<b>Summary of UTL_TCP Subprograms</b> .....	237-11
AVAILABLE Function.....	237-12
CLOSE_ALL_CONNECTIONS Procedure .....	237-14
CLOSE_CONNECTION Procedure .....	237-15
FLUSH Procedure .....	237-16
GET_LINE Function .....	237-17
GET_LINE_NCHAR Function .....	237-18
GET_RAW Function .....	237-19
GET_TEXT Function.....	237-20
GET_TEXT_NCHAR Function.....	237-21
OPEN_CONNECTION Function .....	237-22

READ_LINE Function .....	237-25
READ_RAW Function.....	237-27
READ_TEXT Function.....	237-28
SECURE_CONNECTION Procedure.....	237-30
WRITE_LINE Function .....	237-31
WRITE_RAW Function .....	237-32
WRITE_TEXT Function .....	237-33
<b>238 UTL_URL</b>	
Using UTL_URL .....	238-2
Overview .....	238-3
Exceptions .....	238-4
Examples .....	238-5
<b>Summary of UTL_URL Subprograms</b> .....	238-6
ESCAPE Function.....	238-7
UNESCAPE Function .....	238-9
<b>239 WPG_DOCLOAD</b>	
Using WPG_DOCLOAD .....	239-2
Constants.....	239-3
<b>Summary of WPG_DOCLOAD Subprograms</b> .....	239-4
DOWNLOAD_FILE Procedures.....	239-5
<b>240 ANYDATA TYPE</b>	
Using ANYDATA TYPE .....	240-2
Restrictions.....	240-3
Operational Notes .....	240-4
<b>Summary of ANYDATA Subprograms</b> .....	240-6
BEGINCREATE Static Procedure .....	240-7
ENDCREATE Member Procedure.....	240-8
GET* Member Functions.....	240-9
GETTYPE Member Function .....	240-12
GETTYPENAME Member Function .....	240-13
PIECEWISE Member Procedure .....	240-14
SET* Member Procedures .....	240-15
<b>241 ANYDATASET TYPE</b>	
<b>Construction</b> .....	241-2
<b>Summary of ANYDATASET TYPE Subprograms</b> .....	241-3
ADDINSTANCE Member Procedure .....	241-4
BEGINCREATE Static Procedure .....	241-5
ENDCREATE Member Procedure.....	241-6
GET* Member Functions.....	241-7
GETCOUNT Member Function .....	241-10
GETINSTANCE Member Function .....	241-11
GETTYPE Member Function .....	241-12

GETTYPE_NAME Member Function .....	241-13
PIECEWISE Member Procedure .....	241-14
SET* Member Procedures .....	241-15
<b>242 ANYTYPE TYPE</b>	
<b>Summary of ANYTYPE Subprograms</b> .....	242-2
BEGINCREATE Static Procedure .....	242-3
SETINFO Member Procedure .....	242-4
ADDATTR Member Procedure.....	242-6
ENDCREATE Member Procedure.....	242-7
GETPERSISTENT Static Function.....	242-8
GETINFO Member Function .....	242-9
GETATTRELEMINFO Member Function .....	242-10
<b>243 Oracle Streams AQ TYPEs</b>	
<b>Summary of Types</b> .....	243-2
AQ\$_AGENT Type .....	243-3
AQ\$_AGENT_LIST_T Type .....	243-4
AQ\$_DESCRIPTOR Type .....	243-5
AQ\$_NTFN_DESCRIPTOR Type.....	243-6
AQ\$_NTFN_MSGID_ARRAY Type .....	243-7
AQ\$_POST_INFO Type .....	243-8
AQ\$_POST_INFO_LIST Type.....	243-9
AQ\$_PURGE_OPTIONS_T Type .....	243-10
AQ\$_RECIPIENT_LIST_T Type .....	243-11
AQ\$_REG_INFO Type .....	243-12
AQ\$_REG_INFO_LIST Type .....	243-15
AQ\$_SUBSCRIBER_LIST_T Type .....	243-16
DEQUEUE_OPTIONS_T Type .....	243-17
ENQUEUE_OPTIONS_T Type .....	243-20
SYS.MSG_PROP_T Type.....	243-21
MESSAGE_PROPERTIES_T Type.....	243-24
MESSAGE_PROPERTIES_ARRAY_T Type.....	243-28
MSGID_ARRAY_T Type.....	243-29
<b>244 DBFS Content Interface Types</b>	
<b>Using Content Types</b> .....	244-2
Overview .....	244-3
Security Model.....	244-4
<b>Data Structures</b> .....	244-5
DBMS_DBFS_CONTENT_CONTEXT_T Object Type .....	244-6
DBMS_DBFS_CONTENT_LIST_ITEM_T Object Type .....	244-7
DBMS_DBFS_CONTENT_PROPERTY_T Object Type.....	244-8
DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type.....	244-9
DBMS_DBFS_CONTENT_PROPERTIES_T Table Type .....	244-10
DBMS_DBFS_CONTENT_RAW_T Table Type .....	244-11

---

## 245 Database URI TYPEs

<b>Summary of URITYPE Supertype Subprograms</b> .....	245-2
GETBLOB .....	245-3
GETCLOB .....	245-4
GETCONTENTTYPE .....	245-5
GETEXTERNALURL .....	245-6
GETURL .....	245-7
GETXML .....	245-8
<b>Summary of HTTPURITYPE Subtype Subprograms</b> .....	245-9
CREATEURI .....	245-10
GETBLOB .....	245-11
GETCLOB .....	245-12
GETCONTENTTYPE .....	245-13
GETEXTERNALURL .....	245-14
GETURL .....	245-15
GETXML .....	245-16
HTTPURITYPE .....	245-17
<b>Summary of DBURITYPE Subtype Subprograms</b> .....	245-18
CREATEURI .....	245-19
DBURITYPE .....	245-20
GETBLOB .....	245-21
GETCLOB .....	245-22
GETCONTENTTYPE .....	245-23
GETEXTERNALURL .....	245-24
GETURL .....	245-25
GETXML .....	245-26
<b>Summary of XDBURITYPE Subtype Subprograms</b> .....	245-27
CREATEURI .....	245-28
GETBLOB .....	245-29
GETCLOB .....	245-30
GETCONTENTTYPE .....	245-31
GETEXTERNALURL .....	245-32
GETURL .....	245-33
GETXML .....	245-34
XDBURITYPE .....	245-35
<b>Summary of URIFACTORY Package Subprograms</b> .....	245-36
GETURI .....	245-37
ESCAPEURI .....	245-38
UNESCAPEURI .....	245-39
REGISTERURLHANDLER .....	245-40
UNREGISTERURLHANDLER .....	245-41

## 246 JMS Types

<b>Using JMS Types</b> .....	246-2
Overview .....	246-3
Java Versus PL/SQL Data Types .....	246-4
More on Bytes, Stream and Map Messages .....	246-5



Upcasting and Downcasting Between General and Specific Messages .....	246-9
JMS Types Error Reporting.....	246-10
Oracle JMS Type Constants .....	246-11
CONVERT_JMS_SELECTOR .....	246-13
<b>Summary of JMS Types</b> .....	246-15
SYS.AQ\$_JMS_MESSAGE Type .....	246-16
SYS.AQ\$_JMS_TEXT_MESSAGE Type .....	246-22
SYS.AQ\$_JMS_BYTES_MESSAGE Type .....	246-26
SYS.AQ\$_JMS_MAP_MESSAGE Type .....	246-35
SYS.AQ\$_JMS_STREAM_MESSAGE Type.....	246-45
SYS.AQ\$_JMS_OBJECT_MESSAGE Type.....	246-55
SYS.AQ\$_JMS_NAMESARRAY Type .....	246-56
SYS.AQ\$_JMS_VALUE Type .....	246-57
SYS.AQ\$_JMS_EXCEPTION Type .....	246-58
<b>247 Expression Filter Types</b>	
Using Expression Filter Types .....	247-2
Security Model.....	247-3
<b>Summary of Expression Filter Types</b> .....	247-4
EXF\$ATTRIBUTE .....	247-5
EXF\$ATTRIBUTE_LIST.....	247-6
EXF\$INDEXOPER.....	247-7
EXF\$TABLE_ALIAS .....	247-9
EXF\$TEXT .....	247-10
EXF\$XPATH_TAG.....	247-11
EXF\$XPATH_TAGS.....	247-13
<b>248 Logical Change Record TYPES</b>	
Using Logical Change Record Types .....	248-2
Overview .....	248-3
Security Model.....	248-4
<b>Summary of Logical Change Record Types</b> .....	248-5
LCR\$_DDL_RECORD Type .....	248-6
LCR\$_ROW_RECORD Type .....	248-15
<b>Common Subprograms for LCR\$_DDL_RECORD and LCR\$_ROW_RECORD</b> .....	248-36
LCR\$_ROW_LIST Type.....	248-47
LCR\$_ROW_UNIT Type.....	248-48
<b>249 Oracle Multimedia ORDAudio TYPE</b>	
Documentation of ORDAudio.....	249-2
<b>250 Oracle Multimedia ORDDicom TYPE</b>	
Documentation of ORDDicom.....	250-2

<b>251</b>	<b>Oracle Multimedia ORDDoc TYPE</b>	
	Documentation of ORDDoc.....	251-2
<b>252</b>	<b>Oracle Multimedia ORDImage TYPE</b>	
	Documentation of ORDImage.....	252-2
<b>253</b>	<b>Oracle Multimedia SQL/MM Still Image TYPES</b>	
	Documentation of SQL/MM Still Image .....	253-2
<b>254</b>	<b>Oracle Multimedia ORDVideo TYPE</b>	
	Documentation of ORDVideo .....	254-2
<b>255</b>	<b>MGD_ID Package Types</b>	
	Using MGD_ID Package Object Types.....	255-2
	Security Model.....	255-3
	Summary of Types.....	255-4
	MGD_ID_COMPONENT Object Type .....	255-5
	MGD_ID_COMPONENT_VARRAY Object Type .....	255-6
	MGD_ID Object Type .....	255-7
	Summary of MGD_ID Subprograms .....	255-8
	MGD_ID Constructor Function .....	255-9
	FORMAT Function.....	255-13
	GET_COMPONENT Function .....	255-14
	TO_STRING Function .....	255-16
	TRANSLATE Function.....	255-17
<b>256</b>	<b>Rule TYPES</b>	
	Using Rule Types.....	256-2
	Overview .....	256-3
	Security Model.....	256-4
	Summary of Rule Types .....	256-5
	RE\$ATTRIBUTE_VALUE Type .....	256-7
	RE\$ATTRIBUTE_VALUE_LIST Type.....	256-8
	RE\$COLUMN_VALUE Type .....	256-9
	RE\$COLUMN_VALUE_LIST Type.....	256-10
	RE\$NAME_ARRAY Type.....	256-11
	RE\$NV_ARRAY Type .....	256-12
	RE\$NV_LIST Type .....	256-13
	RE\$NV_NODE Type .....	256-15
	RE\$RULE_HIT Type.....	256-16
	RE\$RULE_HIT_LIST Type .....	256-17
	RE\$TABLE_ALIAS Type.....	256-18
	RE\$TABLE_ALIAS_LIST Type .....	256-19
	RE\$TABLE_VALUE Type.....	256-20
	RE\$TABLE_VALUE_LIST Type .....	256-21

RE\$VARIABLE_TYPE Type .....	256-22
RE\$VARIABLE_TYPE_LIST Type.....	256-24
RE\$VARIABLE_VALUE Type .....	256-25
RE\$VARIABLE_VALUE_LIST Type.....	256-26
<b>257 Rules Manager Types</b>	
Using Rules Manager Types.....	257-2
Security Model.....	257-3
Summary of Rule Manager Types.....	257-4
RLM\$EVENTIDS Object Type .....	257-5
<b>258 UTL Streams Types</b>	
Summary of UTL Binary Streams Types .....	258-2
UTL_BINARYINPUTSTREAM Type.....	258-3
UTL_BINARYOUTPUTSTREAM Type.....	258-4
UTL_CHARACTERINPUTSTREAM Type .....	258-5
UTL_CHARACTEROUTPUTSTREAM Type.....	258-6
<b>259 XMLTYPE</b>	
Summary of XMLType Subprograms .....	259-2
CREATENONSCHEMABASEDXML .....	259-4
CREATESCHEMABASEDXML .....	259-5
CREATEXML.....	259-6
EXISTSNODE.....	259-8
EXTRACT .....	259-9
GETBLOBVAL.....	259-10
GETCLOBVAL .....	259-11
GETNAMESPACE .....	259-12
GETNUMBERVAL .....	259-13
GETROOTELEMENT .....	259-14
GETSCHEMAURL.....	259-15
GETSTRINGVAL .....	259-16
ISFRAGMENT .....	259-17
ISSCHEMABASED .....	259-18
ISSCHEMAVALID.....	259-19
ISSCHEMAVALIDATED.....	259-20
SCHEMAVALIDATE .....	259-21
SETSCHEMAVALIDATED .....	259-22
TOOBJECT .....	259-23
TRANSFORM .....	259-24
XMLTYPE.....	259-25

## Index



---

---

# Preface

This Preface contains these topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

## Audience

*Oracle Database PL/SQL Packages and Types Reference* is intended for programmers, systems analysts, project managers, and others interested in developing database applications. This manual assumes a working knowledge of application programming and familiarity with SQL to access information in relational database systems. Some sections also assume a knowledge of basic object-oriented programming.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Related Documents

For more information, see the following documents:

- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database PL/SQL Language Reference*

Many of the examples in this book use the sample schemas, which are installed by default when you select the Basic Installation option with an Oracle Database installation. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

[https://oraclestore.oracle.com/OA\\_HTML/ibeCZzpHome.jsp](https://oraclestore.oracle.com/OA_HTML/ibeCZzpHome.jsp)

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

<http://otn.oracle.com/membership/>

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

<http://otn.oracle.com/documentation/>

## Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- [Conventions in Text](#)
- [Conventions in Code Examples](#)

### Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
<b>Bold</b>	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an <b>index-organized table</b> .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle Database Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> reside on the same disk.
UPPERCASE monospace (fixed-width) font	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.
lowercase monospace (fixed-width) font	Lowercase monospace typeface indicates executable programs, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names and connect identifiers, user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values.  <i>Note:</i> Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter sqlplus to start SQL*Plus. The password is specified in the orapwd file. Back up the datafiles and control files in the /disk1/oracle/dbs directory. The department_id, department_name, and location_id columns are in the hr.departments table. Set the QUERY_REWRITE_ENABLED initialization parameter to true. The JReplUtil class implements these methods.

Convention	Meaning	Example
<i>lowercase italic monospace (fixed-width) font</i>	Lowercase italic monospace font represents placeholders or variables.	You can specify the <i>parallel_clause</i> .  Run <i>old_release</i> .SQL where <i>old_release</i> refers to the release you installed prior to upgrading.

### Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL\*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Anything enclosed in brackets is optional.	DECIMAL ( <i>digits</i> [ , <i>precision</i> ])
{ }	Braces are used for grouping items.	{ENABLE   DISABLE}
	A vertical bar represents a choice of two options.	{ENABLE   DISABLE} [COMPRESS   NOCOMPRESS]
...	Ellipsis points mean repetition in syntax descriptions.  In addition, ellipsis points can mean an omission in code examples or text.	CREATE TABLE ... AS <i>subquery</i> ;  SELECT <i>col1</i> , <i>col2</i> , ... , <i>coln</i> FROM employees;
Other symbols	You must use symbols other than brackets ([ ]), braces ({}), vertical bars ( ), and ellipsis points (...) exactly as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	enter <i>password</i>  DB_NAME = <i>database_name</i>
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. Because these terms are not case sensitive, you can use them in either UPPERCASE or lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;
lowercase	Lowercase typeface indicates user-defined programmatic elements, such as names of tables, columns, or files.  <b>Note:</b> Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;





---

---

# What's New in PL/SQL Packages and Types Reference?

The following sections describe the new documentation for Oracle Database 11g Release 1 (11.1) and Release 2 (11.2):

- [New Packages](#)
- [Updated Packages](#)
- [New Types](#)
- [Updated Types](#)

---

---

**Note:** Functionality introduced with Oracle Database 11g Release 2 (11.2.0.2) is flagged within the manual, and also by means of \* in the list of packages that follows.

---

---

## New Packages

- [APEX\\_CUSTOM\\_AUTH](#)
- [APEX\\_APPLICATION](#)
- [APEX\\_ITEM](#)
- [APEX\\_UTIL](#)
- [DBMS\\_ADDM](#)
- [DBMS\\_DBFS\\_HS](#)
- [DBMS\\_ASSERT](#)
- [DBMS\\_AUTO\\_SQLTUNE](#)
- [DBMS\\_AUTO\\_TASK\\_ADMIN](#)
- [DBMS\\_AUTO\\_TASK\\_IMMEDIATE](#)
- [DBMS\\_AUDIT\\_MGMT](#)
- [DBMS\\_AW\\_STATS](#)
- [DBMS\\_COMPARISON](#)
- [DBMS\\_COMPRESSION](#)
- [DBMS\\_CONNECTION\\_POOL](#)
- [DBMS\\_CSX\\_ADMIN](#)

- 
- DBMS\_CUBE
  - DBMS\_CUBE\_ADVISE
  - DBMS\_DBFS\_CONTENT
  - DBMS\_DBFS\_CONTENT\_SPI
  - DBMS\_DBFS\_HS
  - DBMS\_DBFS\_SFS
  - DBMS\_DG
  - DBMS\_DST
  - DBMS\_HM
  - DBMS\_HPROF
  - DBMS\_HS\_PARALLEL
  - DBMS\_METADATA\_DIFF
  - DBMS\_MGD\_ID\_UTL
  - DBMS\_NETWORK\_ACL\_ADMIN
  - DBMS\_NETWORK\_ACL\_UTILITY
  - DBMS\_PARALLEL\_EXECUTE
  - DBMS\_RESCONFIG
  - DBMS\_RESULT\_CACHE
  - DBMS\_SPM
  - DBMS\_SQLDIAG
  - DBMS\_STREAMS\_ADVISOR\_ADM
  - DBMS\_STREAMS\_HANDLER\_ADM
  - DBMS\_WORKLOAD\_CAPTURE \*
  - DBMS\_WORKLOAD\_REPLAY \*
  - DBMS\_XA
  - DBMS\_XDB\_ADMIN
  - DBMS\_XDBRESOURCE
  - DBMS\_XEVENT
  - DBMS\_XMLINDEX
  - DBMS\_XMLTRANSLATIONS
  - DBMS\_XPLAN
  - SDO\_CSW\_PROCESS
  - SDO\_GEOR\_ADMIN
  - SDO\_OLS
  - SDO\_PC\_PKG
  - SDO\_TIN\_PKG
  - SDO\_WFS\_LOCK

- 
- SDO\_WFS\_PROCESS
  - SEM\_APIS
  - SEM\_PERF
  - SEM\_RDFCTX
  - SEM\_RDFSA
  - UTL\_MATCH
  - UTL\_SPADV

## Updated Packages

- DBMS\_ADVISOR
- DBMS\_APPLY\_ADM
- DBMS\_AQ
- DBMS\_AQADM
- DBMS\_CAPTURE\_ADM \*
- DBMS\_CDC\_PUBLISH
- DBMS\_CDC\_SUBSCRIBE
- DBMS\_CQ\_NOTIFICATION
- DBMS\_DATA\_MINING \*
- DBMS\_DATA\_MINING\_TRANSFORM
- DBMS\_DATAPUMP \*
- DBMS\_EXPFIL
- DBMS\_FLASHBACK
- DBMS\_HS\_PASSTHROUGH
- DBMS\_LOB
- DBMS\_LOGSTDBY
- DBMS\_METADATA
- DBMS\_MGWADM
- DBMS\_MVIEW
- DBMS\_OUTLN
- DBMS\_PARALLEL\_EXECUTE
- DBMS\_PREDICTIVE\_ANALYTICS
- DBMS\_PROPAGATION\_ADM
- DBMS\_RESOURCE\_MANAGER \*
- DBMS\_RLMGR
- DBMS\_RULE\_ADM
- DBMS\_SCHEDULER \*
- DBMS\_SERVER\_ALERT

- 
- DBMS\_SERVICE \*
  - DBMS\_SESSION
  - DBMS\_SPACE
  - DBMS\_SPACE\_ADMIN \*
  - DBMS\_SQL
  - DBMS\_SQLPA
  - DBMS\_SQLTUNE
  - DBMS\_STATS
  - DBMS\_STREAMS
  - DBMS\_STREAMS\_ADM
  - DBMS\_STREAMS\_TABLESPACE\_ADM
  - DBMS\_TRACE
  - DBMS\_UTILITY
  - DBMS\_WORKLOAD\_REPOSITORY
  - DBMS\_XDB
  - DBMS\_XMLINDEX
  - DBMS\_XMLSCHEMA
  - DBMS\_XPLAN
  - ORD\_DICOM
  - UTL\_HTTP
  - UTL\_IDENT
  - UTL\_INADDR
  - UTL\_RECOMP
  - UTL\_SMTTP
  - UTL\_SPADV
  - UTL\_TCP

## **New Types**

- DBFS Content Interface Types
- MGD\_ID Package Types
- UTL Streams Types

## **Updated Types**

- Oracle Streams AQ TYPES
- Logical Change Record TYPES

---

---

# Introduction to Oracle Supplied PL/SQL Packages & Types

Oracle supplies many PL/SQL packages with the Oracle server to extend database functionality and provide PL/SQL access to SQL features. You can use the supplied packages when creating your applications or for ideas in creating your own stored procedures.

This manual covers the packages provided with the Oracle database server. Packages supplied with other products, such as Oracle Developer or the Oracle Application Server, are not covered.

Note that not every package or type described in this manual or elsewhere in the Oracle Database Documentation Library is installed by default. In such cases, the documentation states this and explains how to install the object. Run this query as a suitably privileged user:

```
SELECT DISTINCT Owner, Object_Type, Object_Name FROM DBA_Objects_AE
WHERE Owner IN (
  'SYS', 'OUTLN', 'SYSTEM', 'CTXSYS', 'DBSNMP',
  'LOGSTDBY_ADMINISTRATOR', 'ORDSYS',
  'ORDPLUGINS', 'OEM_MONITOR', 'MDSYS', 'LBACSYS',
  'DMSYS', 'WMSYS', 'OLAPDBA', 'OLAPSVR', 'OLAP_USER',
  'OLAPSYS', 'EXFSYS', 'SYSMAN', 'MDDATA',
  'SI_INFORMTN_SCHEMA', 'XDB', 'ODM')
AND Object_Type IN ('PACKAGE', 'TYPE')
ORDER BY Owner, Object_Type, Object_Name
```

This lists every Oracle-supplied package and type that is currently installed in the database. Note that it lists a number of objects not mentioned in the Oracle Database Documentation Library. This is deliberate. Some of the Oracle-supplied packages and types are intended to be used only by other Oracle-supplied components. Any package or type that is not described in the Oracle Database Documentation Library is not supported for direct customer use.

This chapter contains the following topics:

- [Package Overview](#)
- [Summary of Oracle Supplied PL/SQL Packages and Types](#)

**See Also:** *Oracle Database Advanced Application Developer's Guide*  
for information on how to create your own packages

---

## Package Overview

A *package* is an encapsulated collection of related program objects stored together in the database. Program objects are procedures, functions, variables, constants, cursors, and exceptions.

## Package Components

PL/SQL packages have two parts: the specification and the body, although sometimes the body is unnecessary. The specification is the interface to your application; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the specification.

Unlike subprograms, packages cannot be called, parameterized, or nested. However, the formats of a package and a subprogram are similar:

```
CREATE PACKAGE name AS -- specification (visible part)
    -- public type and item declarations
    -- subprogram specifications
END [name];
```

```
CREATE PACKAGE BODY name AS -- body (hidden part)
    -- private type and item declarations
    -- subprogram bodies
[BEGIN
    -- initialization statements]
END [name];
```

The specification holds public declarations that are visible to your application. The body holds implementation details and private declarations that are hidden from your application. You can debug, enhance, or replace a package body without changing the specification. You can change a package body without recompiling calling programs because the implementation details in the body are hidden from your application.

## Using Oracle Supplied Packages

Most Oracle supplied packages are automatically installed when the database is created. Certain packages are not installed automatically. Special installation instructions for these packages are documented in the individual chapters.

To call a PL/SQL function from SQL, you must either own the function or have EXECUTE privileges on the function. To select from a view defined with a PL/SQL function, you must have SELECT privileges on the view. No separate EXECUTE privileges are needed to select from the view. Instructions on special requirements for packages are documented in the individual chapters.



## Creating New Packages

To create packages and store them permanently in an Oracle database, use the `CREATE PACKAGE` and `CREATE PACKAGE BODY` statements. You can execute these statements interactively from SQL\*Plus or Enterprise Manager.

To create a new package, do the following:

1. Create the package specification with the `CREATE PACKAGE` statement.

You can declare program objects in the package specification. Such objects are called *public* objects. Public objects can be referenced outside the package, as well as by other objects in the package.

---

---

**Note:** It is often more convenient to add the `OR REPLACE` clause in the `CREATE PACKAGE` statement. But note that `CREATE PACKAGE` warns you if you are about to overwrite an existing package with the same name while `CREATE OR REPLACE` just overwrites it with no warning.

---

---

2. Create the package body with the `CREATE PACKAGE BODY` statement.

You can declare and define program objects in the package body.

- You must define public objects declared in the package specification.
- You can declare and define additional package objects, called *private* objects. Private objects are declared in the package body rather than in the package specification, so they can be referenced only by other objects in the package. They cannot be referenced outside the package.

**See Also:**

- *Oracle Database PL/SQL Language Reference*
- *Oracle Database Advanced Application Developer's Guide* for more information on creating new packages
- *Oracle Database Concepts*

for more information on storing and executing packages

## Separating the Specification and Body

The specification of a package declares the public types, variables, constants, and subprograms that are visible outside the immediate scope of the package. The body of a package defines the objects declared in the specification, as well as private objects that are not visible to applications outside the package.

Oracle stores the specification and body of a package separately in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body. Using this distinction, you can change the definition of a program object in the package body without causing Oracle to invalidate other schema objects that call or reference the program object. Oracle invalidates dependent schema objects only if you change the declaration of the program object in the package specification.

### Creating a New Package: Example

The following example shows a package specification for a package named `EMPLOYEE_MANAGEMENT`. The package contains one stored function and two stored procedures.

```
CREATE PACKAGE employee_management AS
    FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
        mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
        deptno NUMBER) RETURN NUMBER;
    PROCEDURE fire_emp (emp_id NUMBER);
    PROCEDURE sal_raise (emp_id NUMBER, sal_incr NUMBER);
END employee_management;
```

The body for this package defines the function and the procedures:

```
CREATE PACKAGE BODY employee_management AS
    FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
        mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
        deptno NUMBER) RETURN NUMBER IS
```

The function accepts all arguments for the fields in the employee table except for the employee number. A value for this field is supplied by a sequence. The function returns the sequence number generated by the call to this function.

```
    new_empno    NUMBER(10);

    BEGIN
        SELECT emp_sequence.NEXTVAL INTO new_empno FROM dual;
        INSERT INTO emp VALUES (new_empno, name, job, mgr,
            hiredate, sal, comm, deptno);
        RETURN (new_empno);
    END hire_emp;

    PROCEDURE fire_emp(emp_id IN NUMBER) AS
```

The procedure deletes the employee with an employee number that corresponds to the argument `emp_id`. If no employee is found, then an exception is raised.

```
    BEGIN
        DELETE FROM emp WHERE empno = emp_id;
        IF SQL%NOTFOUND THEN
            raise_application_error(-20011, 'Invalid Employee
                Number: ' || TO_CHAR(emp_id));
        END IF;
    END fire_emp;

    PROCEDURE sal_raise (emp_id IN NUMBER, sal_incr IN NUMBER) AS
```

The procedure accepts two arguments. `Emp_id` is a number that corresponds to an employee number. `Sal_incr` is the amount by which to increase the employee's salary.

```
    BEGIN

        -- If employee exists, then update salary with increase.

        UPDATE emp
            SET sal = sal + sal_incr
            WHERE empno = emp_id;
        IF SQL%NOTFOUND THEN
            raise_application_error(-20011, 'Invalid Employee
```

```
        Number: ' || TO_CHAR(emp_id));  
    END IF;  
END sal_raise;  
END employee_management;
```

---

---

**Note:** If you want to try this example, then first create the sequence number `emp_sequence`. You can do this using the following SQL\*Plus statement:

```
SQL> CREATE SEQUENCE emp_sequence  
      > START WITH 8000 INCREMENT BY 10;
```

---

---

## Referencing Package Contents

To reference the types, items, and subprograms declared in a package specification, use the dot notation. For example:

```
package_name.type_name  
package_name.item_name  
package_name.subprogram_name
```

## Summary of Oracle Supplied PL/SQL Packages and Types

Table 1–1 lists the supplied PL/SQL server packages. These packages run as the invoking user, rather than the package owner. Unless otherwise noted, the packages are callable through public synonyms of the same name.

---



---

**Caution:**

- The procedures and functions provided in these packages and their external interfaces are reserved by Oracle and are subject to change.
  - Modifying Oracle supplied packages can cause internal errors and database security violations. Do not modify supplied packages.
- 
- 

**Table 1–1 Summary of Oracle Supplied PL/SQL Packages**

Package Name	Description
APEX_CUSTOM_AUTH	Provides an interface for authentication and session management
APEX_APPLICATION	Enables users to take advantage of global variables
APEX_ITEM	Enables users to create form elements dynamically based on a SQL query instead of creating individual items page by page
APEX_UTIL	Provides utilities for getting and setting session state, getting files, checking authorizations for users, resetting different states for users, and also getting and setting preferences for users
CTX_ADM	Lets you administer servers and the data dictionary
CTX_CLS	Lets you generate CTXRULE rules for a set of documents
CTX_DDL	Lets you create and manage the preferences, section lists and stopgroups required for Text indexes
CTX_DOC	Lets you request document services
CTX_OUTPUT	Lets you manage the index log
CTX_QUERY	Lets you generate query feedback, count hits, and create stored query expressions
CTX_REPORT	Lets you create various index reports
CTX_THES	Lets you to manage and browse thesauri
CTX_ULEXER	For use with the user-lexer
DBMS_ADDM	Facilitates the use of Advisor functionality regarding the Automatic Database Diagnostic Monitor
DBMS_ADVANCED_REWRITE	Contains interfaces for advanced query rewrite users to create, drop, and maintain functional equivalence declarations for query rewrite
DBMS_ADVISOR	Part of the SQLAccess Advisor, an expert system that identifies and helps resolve performance problems relating to the execution of SQL statements

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_ALERT	Provides support for the asynchronous notification of database events
DBMS_APPLICATION_INFO	Lets you register an application name with the database for auditing or performance tracking purposes
DBMS_APPLY_ADM	Provides administrative procedures to start, stop, and configure an apply process
DBMS_AQ	Lets you add a message (of a predefined object type) onto a queue or to dequeue a message
DBMS_AQADM	Lets you perform administrative functions on a queue or queue table for messages of a predefined object type
DBMS_AQELM	Provides procedures to manage the configuration of Advanced Queuing asynchronous notification by e-mail and HTTP
DBMS_AQIN	Plays a part in providing secure access to the Oracle JMS interfaces
DBMS_ASSERT	Provides an interface to validate properties of the input value
DBMS_AUTO_TASK_ADMIN	Used by the DBA as well as Enterprise Manager to access the AUTOTASK controls
DBMS_AUTO_TASK_IMMEDIATE	Cconsists of one subprogram whose function is to initiate gathering of optimizer statistics without delay
DBMS_AW_STATS	Contains a subprogram that generates and stores optimizer statistics for cubes and dimensions
DBMS_CAPTURE_ADM	Describes administrative procedures to start, stop, and configure a capture process; used in Streams
DBMS_CDC_PUBLISH	Identifies new data that has been added to, modified, or removed from, relational tables and publishes the changed data in a form that is usable by an application
DBMS_CDC_SUBSCRIBE	Lets you view and query the change data that was captured and published with the DBMS_LOGMNR_CDC_PUBLISH package
DBMS_COMPARISON	Provides interfaces to compare and converge database objects at different databases
DBMS_COMPRESSION	Provides an interface to facilitate choosing the correct compression level for an application
DBMS_CONNECTION_POOL	Provides an interface to manage the Database Resident Connection Pool
DBMS_CQ_NOTIFICATION	Is part of a set of features that clients use to receive notifications when result sets of a query have changed. The package contains interfaces that can be used by mid-tier clients to register objects and specify delivery mechanisms.
DBMS_CRYPT	Lets you encrypt and decrypt stored data, can be used in conjunction with PL/SQL programs running network communications, and supports encryption and hashing algorithms
DBMS_CSX_ADMIN	Provides an interface to customize the setup when transporting a tablespace containing binary XML data

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_CUBE	Contains subprograms that create OLAP cubes and dimensions, and that load and process the data for querying
DBMS_CUBE_ADVISE	Contains subprograms for evaluating cube materialized views to support log-based fast refresh and query rewrite
DBMS_DATA_MINING	Implements the Oracle Data Mining interface for creating, evaluating, and managing mining models
DBMS_DATA_MINING_TRANSFORM	Provides subroutines that can be used to prepare data for Oracle Data Mining
DBMS_DATAPUMP	Lets you move all, or part of, a database between databases, including both data and metadata
DBMS_DBFS_CONTENT	Provides an interface comprising a file system-like abstraction backed by one or more Store Providers
DBMS_DBFS_CONTENT_SPI	Provides the Application Programming Interface (API) specification for DBMS_DBFS_CONTENT service providers
DBMS_DBFS_HS	Provides users the ability to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management for their database tables
DBMS_DBFS_SFS	Provides an interface to operate a SecureFile-based store (SFS) for the content interface described in the DBMS_DBFS_CONTENT package
DBMS_DB_VERSION	Specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions
DBMS_DDL	Provides access to some SQL DDL statements from stored procedures, and provides special administration operations not available as DDLs
DBMS_DEBUG	Implements server-side debuggers and provides a way to debug server-side PL/SQL program units
DBMS_DEFER	Provides the user interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.
DBMS_DEFER_QUERY	Permits querying the deferred remote procedure calls (RPC) queue data that is not exposed through views. Requires the Distributed Option.
DBMS_DEFER_SYS	Provides the system administrator interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.
DBMS_DESCRIBE	Describes the arguments of a stored procedure with full name translation and security checking
DBMS_DG	Allows applications to notify the primary database in an Oracle Data Guard broker environment to initiate a fast-start failover when the application encounters a condition that warrants a failover
DBMS_DIMENSION	Enables you to verify dimension relationships and provides an alternative to the Enterprise Manager Dimension Wizard for displaying a dimension definition

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_DISTRIBUTED_TRUST_ADMIN	Maintains the Trusted Database List, which is used to determine if a privileged database link from a particular server can be accepted
DBMS_EPG	Implements the embedded PL/SQL gateway that enables a Web browser to invoke a PL/SQL stored procedure through an HTTP listener
DBMS_ERRLOG	Provides a procedure that enables you to create an error logging table so that DML operations can continue after encountering errors rather than abort and roll back
DBMS_EXPFIL	Contains all the procedures used to manage attribute sets, expression sets, expression indexes, optimizer statistics, and privileges by Expression Filter
DBMS_FGA	Provides fine-grained security functions
DBMS_FILE_GROUP	One of a set of Streams packages, provides administrative interfaces for managing file groups, file group versions, files and file group repositories
DBMS_FILE_TRANSFER	Lets you copy a binary file within a database or to transfer a binary file between databases
DBMS_FLASHBACK	Lets you flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN)
DBMS_FLASHBACK_ARCHIVE	Contains procedures for disassociation and reassociation of a Flashback Data Archive (FDA) enabled table from/with its underlying FDA respectively.
DBMS_FREQUENT_ITEMSET	Enables frequent itemset counting
DBMS_HM	Contains constants and procedure declarations for health check management
DBMS_HPROF	Provides an interface for profiling the execution of PL/SQL applications
DBMS_HS_PARALLEL	Enables parallel processing for heterogeneous targets access
DBMS_HS_PASSTHROUGH	Lets you use Heterogeneous Services to send pass-through SQL statements to non-Oracle systems
DBMS_IOT	Creates a table into which references to the chained rows for an Index Organized Table can be placed using the ANALYZE command
DBMS_JAVA	Provides a PL/SQL interface for accessing database functionality from Java
DBMS_JOB	Schedules and manages jobs in the job queue
DBMS_LDAP	Provides functions and procedures to access data from LDAP servers
DBMS_LDAP_UTL	Provides the Oracle Extension utility functions for LDAP
DBMS_LIBCACHE	Prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution
DBMS_LOB	Provides general purpose routines for operations on Oracle Large Object (LOBs) datatypes - BLOB, CLOB (read/write), and BFILES (read-only)



**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_LOCK	Lets you request, convert and release locks through Oracle Lock Management services
DBMS_LOGMNR	Provides functions to initialize and run the log reader
DBMS_LOGMNR_D	Queries the dictionary tables of the current database, and creates a text based file containing their contents
DBMS_LOGSTDBY	Describes procedures for configuring and managing the logical standby database environment
DBMS_METADATA	Lets callers easily retrieve complete database object definitions (metadata) from the dictionary
DBMS_METADATA_DIFF	Contains the interfaces for comparing two metadata documents in SXML format. The result of the comparison is an SXML difference document. This document can be converted to other formats using the DBMS_METADATA submit interface and the CONVERT API.
DBMS_MGD_ID_UTL	Provides a set of utility subprograms
DBMS_MGWADM	Describes the Messaging Gateway administrative interface; used in Advanced Queuing
DBMS_MGWMSG	Describes object types (used by the canonical message types to convert message bodies) and helper methods, constants, and subprograms for working with the Messaging Gateway message types; used in Advanced Queuing.
DBMS_MONITOR	Let you use PL/SQL for controlling additional tracing and statistics gathering
DBMS_MVIEW	Lets you refresh snapshots that are not part of the same refresh group and purge logs. DBMS_SNAPSHOT is a synonym.
DBMS_NETWORK_ACL_ADMIN	Provides the interface to administer the network Access Control List (ACL)
DBMS_NETWORK_UTL	Provides the interface to administer the network Access Control List (ACL)
DBMS_OBFUSCATION_TOOLKIT	Provides procedures for Data Encryption Standards
DBMS_ODCI	Returns the CPU cost of a user function based on the elapsed time of the function
DBMS_OFFLINE_OG	Provides a public interface for offline instantiation of master groups
DBMS_OLAP	Provides procedures for summaries, dimensions, and query rewrites
DBMS_OUTLN	Provides the interface for procedures and functions associated with management of stored outlines Synonymous with OUTLN_PKG
DBMS_OUTPUT	Accumulates information in a buffer so that it can be retrieved later
DBMS_PARALLEL_EXECUTE	Enables the user to incrementally update table data in parallel
DBMS_PCLXUTIL	Provides intra-partition parallelism for creating partition-wise local indexes

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_PIPE	Provides a DBMS pipe service which enables messages to be sent between sessions
DBMS_PREDICTIVE_ANALYTICS	Provides subroutines that implement automatic data mining operations for predict, explain, and profile
DBMS_PREPROCESSOR	Provides an interface to print or retrieve the source text of a PL/SQL unit in its post-processed form
DBMS_PROFILER	Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks
DBMS_PROPAGATION_ADM	Provides administrative procedures for configuring propagation from a source queue to a destination queue
DBMS_RANDOM	Provides a built-in random number generator
DBMS_RECTIFIER_DIFF	Provides an interface to detect and resolve data inconsistencies between two replicated sites
DBMS_REDEFINITION	Lets you perform an online reorganization of tables
DBMS_REFRESH	Lets you create groups of snapshots that can be refreshed together to a transactionally consistent point in time Requires the Distributed Option
DBMS_REPAIR	Provides data corruption repair procedures
DBMS_REPCAT	Provides routines to administer and update the replication catalog and environment. Requires the Replication Option.
DBMS_REPCAT_ADMIN	Lets you create users with the privileges needed by the symmetric replication facility. Requires the Replication Option.
DBMS_REPCAT_INSTANTIATE	Instantiates deployment templates. Requires the Replication Option.
DBMS_REPCAT_RGT	Controls the maintenance and definition of refresh group templates. Requires the Replication Option.
DBMS_REPUTIL	Provides routines to generate shadow tables, triggers, and packages for table replication.
DBMS_RESCONFIG	Provides an interface to operate on the Resource Configuration List, and to retrieve listener information for a resource
DBMS_RESOURCE_MANAGER	Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema
DBMS_RESOURCE_MANAGER_PRIVS	Maintains privileges associated with resource consumer groups
DBMS_RESULT_CACHE	Provides an interface to operate on the Result Cache
DBMS_RESUMABLE	Lets you suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution
DBMS_RLMGR	Contains various procedures to create and manage rules and rule sessions by the Rules Manager
DBMS_RLS	Provides row level security administrative interface
DBMS_ROWID	Provides procedures to create rowids and to interpret their contents

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_RULE	Describes the EVALUATE procedure used in Streams
DBMS_RULE_ADM	Describes the administrative interface for creating and managing rules, rule sets, and rule evaluation contexts; used in Streams
DBMS_SCHEDULER	Provides a collection of scheduling functions that are callable from any PL/SQL program
DBMS_SERVER_ALERT	Lets you issue alerts when some threshold has been violated
DBMS_SERVICE	Lets you create, delete, activate and deactivate services for a single instance
DBMS_SESSION	Provides access to SQL ALTER SESSION statements, and other session information, from stored procedures
DBMS_SHARED_POOL	Lets you keep objects in shared memory, so that they will not be aged out with the normal LRU mechanism
DBMS_SPACE	Provides segment space information not available through standard SQL
DBMS_SPACE_ADMIN	Provides tablespace and segment space administration not available through the standard SQL
DBMS_SPM	Supports the SQL plan management feature by providing an interface for the DBA or other user to perform controlled manipulation of plan history and SQL plan baselines maintained for various SQL statements
DBMS_SQL	Lets you use dynamic SQL to access the database
DBMS_SQLDIAG	Provides an interface to the SQL Diagnosability functionality
DBMS_SQLPA	Provides an interface to implement the SQL Performance Analyzer.
DBMS_SQLTUNE	Provides the interface to tune SQL statements
DBMS_STAT_FUNCS	Provides statistical functions
DBMS_STATS	Provides a mechanism for users to view and modify optimizer statistics gathered for database objects
DBMS_STORAGE_MAP	Communicates with FMON to invoke mapping operations
DBMS_STREAMS	Describes the interface to convert SYS.AnyData objects into LCR objects and an interface to annotate redo entries generated by a session with a binary tag.
DBMS_STREAMS_ADMIN	Describes administrative procedures for adding and removing simple rules, without transformations, for capture, propagation, and apply at the table, schema, and database level
DBMS_STREAMS_ADVISOR_ADM	Provides an interface to gather information about an Oracle Streams environment and advise database administrators based on the information gathered
DBMS_STREAMS_AUTH	Provides interfaces for granting privileges to Streams administrators and revoking privileges from Streams administrators

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_STREAMS_HANDLER_ADM	Provides interfaces to enqueue messages into and dequeue messages from a <code>SYS.AnyData</code> queue
DBMS_STREAMS_MESSAGING	Provides interfaces to enqueue messages into and dequeue messages from a <code>SYS.AnyData</code> queue
DBMS_STREAMS_TABLESPACE_ADM	Provides administrative procedures for copying tablespaces between databases and moving tablespaces from one database to another
DBMS_TDB	Reports whether a database can be transported between platforms using the <code>RMAN CONVERT DATABASE</code> command. It verifies that databases on the current host platform are of the same endian format as the destination platform, and that the state of the current database does not prevent transport of the database.
DBMS_TRACE	Provides routines to start and stop PL/SQL tracing
DBMS_TRANSACTION	Provides access to SQL transaction statements from stored procedures and monitors transaction activities
DBMS_TRANSFORM	Provides an interface to the message format transformation features of Oracle Advanced Queuing
DBMS_TTS	Checks if the transportable set is self-contained
DBMS_TYPES	Consists of constants, which represent the built-in and user-defined types
DBMS_UTILITY	Provides various utility routines
DBMS_WARNING	Provides the interface to query, modify and delete current system or session settings
DBMS_WM	Describes how to use the programming interface to Oracle Database Workspace Manager to work with long transactions
DBMS_WORKLOAD_CAPTURE	Configures the Workload Capture system and produce the workload capture data.
DBMS_WORKLOAD_REPLAY	Provides an interface to replay and report on a record of a workload on a production or test system
DBMS_WORKLOAD_REPOSITORY	Lets you manage the Workload Repository, performing operations such as managing snapshots and baselines
DBMS_XA	Contains the XA/Open interface for applications to call XA interface in PL/SQL
DBMS_XDB	Describes Resource Management and Access Control interface for PL/SQL
DBMS_XDB_ADMIN	Provides an interface to implement <code>XMLIndex</code> administration operation
DBMS_XDBRESOURCE	Provides an interface to operate on the XDB resource's metadata and contents
DBMS_XDB_VERSION	Describes the versioning interface
DBMS_XDBT	Describes how an administrator can create a <code>ConText</code> index on the XML DB hierarchy and configure it for automatic maintenance
DBMS_XDBZ	Controls the Oracle XML DB repository security, which is based on Access Control Lists (ACLs)

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
DBMS_XEVENT	Provides event-related types and supporting subprograms
DBMS_XMLDOM	Explains access to XMLType objects
DBMS_XMLGEN	Converts the results of a SQL query to a canonical XML format
DBMS_XMLINDEX	Provides an interface to implement asynchronous indexing and apply node referencing
DBMS_XMLPARSER	Explains access to the contents and structure of XML documents
DBMS_XMLQUERY	Provides database-to-XMLType functionality
DBMS_XMLSAVE	Provides XML-to-database-type functionality
DBMS_XMLSCHEMA	Explains procedures to register and delete XML schemas
DBMS_XMLSTORE	Provides the ability to store XML data in relational tables
DBMS_XMLTRANSLATIONS	Provides an interface to perform translations so that strings can be searched or displayed in various languages
DBMS_XPLAN	Describes how to format the output of the EXPLAIN PLAN command
DBMS_XSLPROCESSOR	Explains access to the contents and structure of XML documents
DEBUG_EXTPROC	Lets you debug external procedures on platforms with debuggers that attach to a running process
HTF	Hypertext functions generate HTML tags
HTP	Hypertext procedures generate HTML tags
ORD_DICOM	Supports the management and manipulation of Digital Imaging and Communications in Medicine (DICOM) content stored in BLOBs or BFILES rather than in an ORDDiCOM object type
ORD_DICOM_ADMIN	Used by Oracle Multimedia Digital Imaging and Communications in Medicine (DICOM) administrators to maintain the Oracle Multimedia DICOM data model repository
OWA_CACHE	Provides an interface that enables the PL/SQL Gateway cache to improve the performance of PL/SQL Web applications
OWA_COOKIE	Provides an interface for sending and retrieving HTTP cookies from the client's browser
OWA_CUSTOM	Provides a Global PLSQL Agent Authorization callback function
OWA_IMAGE	Provides an interface to access the coordinates where a user clicked on an image
OWA_OPT_LOCK	Contains subprograms that impose optimistic locking strategies so as to prevent lost updates
OWA_PATTERN	Provides an interface to locate text patterns within strings and replace the matched string with another string

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
OWA_SEC	Provides an interface for custom authentication
OWA_TEXT	Contains subprograms used by OWA_PATTERN for manipulating strings. They are externalized so you can use them directly.
OWA_UTIL	Contains utility subprograms for performing operations such as getting the value of CGI environment variables, printing the data that is returned to the client, and printing the results of a query in an HTML table
SDO_CS	Provides functions for coordinate system transformation
SDO_CSW_PROCESS	Contains subprograms for various processing operations related to support for Catalog Services for the Web (CSW)
SDO_GCDR	Contains the Oracle Spatial geocoding subprograms, which let you geocode unformatted postal addresses
SDO_GEOM	Provides functions implementing geometric operations on spatial objects
SDO_GEOR	Contains functions and procedures for the Spatial GeoRaster feature, which lets you store, index, query, analyze, and deliver raster image data and its associated Spatial vector geometry data and metadata
SDO_GEOR_ADMIN	Contains subprograms for administrative operations related to GeoRaster.
SDO_GEOR_UTL	Contains utility functions and procedures for the Spatial GeoRaster feature, including those related to using triggers with GeoRaster data
SDO_LRS	Provides functions for linear referencing system support
SDO_MIGRATE	Provides functions for migrating spatial data from previous releases
SDO_NET	Provides functions and procedures for working with data modeled as nodes and links in a network
SDO_NET_MEM	Contains functions and procedures for performing editing and analysis operations on network data using a network memory object
SDO_OLS	Contains functions and procedures for performing editing and analysis operations on network data using a network memory object
SDO_PC_PKG	Contains subprograms to support the use of point clouds in Spatial
SDO_SAM	Contains functions and procedures for spatial analysis and data mining
SDO_TIN_PKG	Contains subprograms to support the use of triangulated irregular networks (TINs) in Spatial
SDO_TOPO	Provides procedures for creating and managing Spatial topologies
SDO_TOPO_MAP	Contains subprograms for editing Spatial topologies using a cache (TopoMap object)
SDO_TUNE	Provides functions for selecting parameters that determine the behavior of the spatial indexing scheme used in Oracle Spatial

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
SDO_UTIL	Provides utility functions and procedures for Oracle Spatial
SDO_WFS_LOCK	Contains subprograms for WFS support for registering and unregistering feature tables
SDO_WFS_PROC	Provides utility functions and procedures for Oracle Spatial
SEM_APIS	Contains subprograms for working with the Resource Description Framework (RDF) and Web Ontology Language (OWL) in an Oracle database.
SEM_PERF	Contains subprograms for examining and enhancing the performance of the Resource Description Framework (RDF) and Web Ontology Language (OWL) support in an Oracle database
SEM_RDFCTX	Contains subprograms for managing extractor policies and semantic indexes created for documents
SEM_RDFSA	Contains subprograms for providing fine-grained access control to RDF data, using either a virtual private database (VPD) or Oracle Label Security (OLS)
UTL_COLL	Enables PL/SQL programs to use collection locators to query and update
UTL_COMPRESS	Provides a set of data compression utilities
UTL_DBWS	Provides database Web services
UTL_ENCODE	Provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts
UTL_FILE	Enables your PL/SQL programs to read and write operating system text files and provides a restricted version of standard operating system stream file I/O
UTL_HTTP	Enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges
UTL_I18N	Provides a set of services (Oracle Globalization Service) that help developers build multilingual applications
UTL_INADDR	Provides a procedure to support internet addressing
UTL_IDENT	Specifies which database or client PL/SQL is running
UTL_LMS	Retrieves and formats error messages in different languages
UTL_MAIL	A utility for managing email which includes commonly used email features, such as attachments, CC, BCC, and return receipt
UTL_NLA	Exposes a subset of the BLAS and LAPACK (Version 3.0) operations on vectors and matrices represented as VARRAYs
UTL_RAW	Provides SQL functions for manipulating RAW datatypes
UTL_RECOMP	Recompiles invalid PL/SQL modules, invalid views, Java classes, indextypes and operators in a database, either sequentially or in parallels

**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
UTL_REF	Enables a PL/SQL program to access an object by providing a reference to the object
UTL_SMTP	Provides PL/SQL functionality to send emails
UTL_SPADV	Provides subprograms to collect and analyze statistics for the Oracle Streams components in a distributed database environment
UTL_TCP	Provides PL/SQL functionality to support simple TCP/IP-based communications between servers and the outside world
UTL_URL	Provides escape and unescape mechanisms for URL characters
WPG_DOCLOAD	Provides an interface to download files, both BLOBs and BFILEs
ANYDATA TYPE	A self-describing data instance type containing an instance of the type plus a description
ANYDATASET TYPE	Contains a description of a given type plus a set of data instances of that type
ANYTYPE TYPE	Contains a type description of any persistent SQL type, named or unnamed, including object types and collection types; or, it can be used to construct new transient type descriptions
Oracle Streams AQ Types	Describes the types used in Advanced Queuing
DBFS Content Interface Types	Describes public types defined to support the DBMS_DBFS_CONTENT interface.
Database URI Type	Contains URI Support, UriType Super Type, HttpUriType Subtype, DBUriType Subtype, XDBUriType Subtype, UriFactory Package
Expression Filter Types	Expression Filter feature is supplied with a set of predefined types and public synonyms for these types.
JMS TYPES	Describes JMS types so that a PL/SQL application can use JMS queues of JMS types
Oracle Multimedia ORDAudio TYPE	Supports the storage and management of audio data
Oracle Multimedia ORDDicom Type	Supports the storage, management, and manipulation of Digital Imaging and Communications in Medicine (DICOM) data
Oracle Multimedia ORDDoc TYPE	Supports the storage and management of heterogeneous media data including image, audio, and video
Oracle Multimedia ORDImage TYPE	Supports the storage, management, and manipulation of image data
Oracle Multimedia SQL/MM Still Image TYPE	Supports the SQL/MM Still Image Standard, which lets you store, retrieve, and modify images in the database and locate images using visual predicates
Oracle Multimedia ORDVideo TYPE	Supports the storage and management of video data
LOGICAL CHANGE RECORD TYPES	Describes LCR types, which are message payloads that contain information about changes to a database, used in Streams



**Table 1–1 (Cont.) Summary of Oracle Supplied PL/SQL Packages**

<b>Package Name</b>	<b>Description</b>
MG_ID Package Types	Provides an extensible framework that supports current RFID tags with the standard family of EPC bit encodings for the supported encoding types
RULES TYPES	Describes the types used with rules, rule sets, and evaluation contexts
RULES Manager Types	Rules Manager is supplied with one predefined type and a public synonym
UTL Streams Types	Describes describes abstract streams types used with Oracle XML functionality
XMLType	Describes the types and functions used for native XML support in the server



---

---

## APEX\_CUSTOM\_AUTH

The APEX\_CUSTOM\_AUTH package provides an interface for authentication and session management.

- [Documentation of APEX\\_CUSTOM\\_AUTH](#)

---

## Documentation of APEX\_CUSTOM\_AUTH

For a complete description of this package within the context of APEX, see APEX\_CUSTOM\_AUTH in the *Oracle Application Express API Reference*.

---

---

## APEX\_APPLICATION

The APEX\_APPLICATION package enables users to take advantage of global variables.

- [Documentation of APEX\\_APPLICATION](#)

---

## Documentation of APEX\_APPLICATION

For a complete description of this package within the context of APEX, see APEX\_APPLICATION in the *Oracle Application Express API Reference*.

---

---

## APEX\_ITEM

The APEX\_ITEM package enables users to create form elements dynamically based on a SQL query instead of creating individual items page by page.

- [Documentation of APEX\\_ITEM](#)

---

## Documentation of APEX\_ITEM

For a complete description of this package within the context of APEX, see `APEX_ITEM` in the *Oracle Application Express API Reference*.



The APEX\_UTIL package provides utilities for getting and setting session state, getting files, checking authorizations for users, resetting different states for users, and also getting and setting preferences for users.

- [Documentation of APEX\\_UTIL](#)

---

## Documentation of APEX\_UTIL

For a complete description of this package within the context of APEX, see `APEX_UTIL` in the *Oracle Application Express API Reference*.

The CTX\_ADM package lets you administer the Oracle Text data dictionary. Note that you must install this package in order to use it.

- [Documentation of CTX\\_ADM](#)

---

## Documentation of CTX\_ADM

For a complete description of this package within the context of Oracle Text, see CTX\_ADM in the *Oracle Text Reference*.

The CTX\_CLS package enables generation of CTXRULE rules for a set of documents.

- [Documentation of CTX\\_CLS](#)

---

## Documentation of CTX\_CLS

For a complete description of this package within the context of Oracle Text, see CTX\_CLS in the *Oracle Text Reference*.

The CTX\_DDL package lets you create and manage the preferences, section groups, and stoplists required for Text indexes. Note that you must install this package in order to use it.

- [Documentation of CTX\\_DDL](#)

---

## Documentation of CTX\_DDL

For complete description of this package within the context of Oracle Text, see CTX\_DDL in the *Oracle Text Reference*.



The CTX\_DOC package lets you request document services. Note that you must install this package in order to use it.

- [Documentation of CTX\\_DOC](#)

---

## Documentation of CTX\_DOC

For a complete description of this package within the context of Oracle Text, see CTX\_DOC in the *Oracle Text Reference*.

---

---

**Note:** The CTX\_ENTITY package is not available with release 11.2.0.3.

---

---

The CTX\_ENTITY package enables you to search for terms that are unknown to you without specifying a particular search text. It does this by identifying names, places, dates, and other objects when they are mentioned in a document and tagging each occurrence (called a mention) with its type and subtype. This process enables you to produce a structured view of a document that can later be used for text/data mining and more comprehensive intelligence analysis.

- [Documentation of CTX\\_ENTITY](#)

---

## Documentation of CTX\_ENTITY

For complete description of this package within the context of Oracle Text, see CTX\_ENTITY in the *Oracle Text Reference*.

This Oracle Text package lets you manage the index log. Note that you must install this package in order to use it.

- [Documentation of CTX\\_OUTPUT](#)

---

## Documentation of CTX\_OUTPUT

For a complete description of this package within the context of Oracle Text, see CTX\_OUTPUT in the *Oracle Text Reference*.

This Oracle Text package lets you generate query feedback, count hits, and create stored query expressions. Note that you must install this package in order to use it.

- [Documentation of CTX\\_QUERY](#)

---

## Documentation of CTX\_QUERY

For a complete description of this package within the context of Oracle Text, see CTX\_QUERY in the *Oracle Text Reference*.



This Oracle Text package lets you create various index reports. Note that you must install this package in order to use it.

- [Documentation of CTX\\_REPORT](#)

---

## Documentation of CTX\_REPORT

For a complete description of this package within the context of Oracle Text, see CTX\_REPORT in the *Oracle Text Reference*.

This Oracle Text package lets you to manage and browse thesauri. Note that you must install this package in order to use it.

- [Documentation of CTX\\_THES](#)

---

## Documentation of CTX\_THES

For a complete description of this package within the context of Oracle Text, see CTX\_THES in the *Oracle Text Reference*.

This Oracle Text package is for use with the user-lexer. Note that you must install this package in order to use it.

- [Documentation of CTX\\_ULEXER](#)

---

## Documentation of CTX\_ULEXER

For a complete description of this package within the context of Oracle Text, see CTX\_ULEXER in the *Oracle Text Reference*.

The DBMS\_ADDM package facilitates the use of Advisor functionality regarding the Automatic Database Diagnostic Monitor.

**See Also:**

- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about "Automatic Workload Repository in Oracle Real Application Clusters Environments"
- *Oracle Database Performance Tuning Guide* for more information about "Automatic Performance Diagnostics"

This chapter contains the following topics:

- [Using DBMS\\_ADDM](#)
  - Security Model
- [Summary of DBMS\\_ADDM Subprograms](#)

## Using DBMS\_ADDM

---

- [Security Model](#)



## Security Model

The DBMS\_ADDM package runs with the caller's permission, not the definer's, and then applies the security constraints required by the DBMS\_ADVISOR package.

**See Also:** The [DBMS\\_ADVISOR](#) package for more information about "Security Model".

---

## Summary of DBMS\_ADDM Subprograms

**Table 16–1 DBMS\_ADDM Package Subprograms**

Subprogram	Description
<a href="#">ANALYZE_DB Procedure</a> on page 16-5	Creates an ADDM task for analyzing in database analysis mode and executes it
<a href="#">ANALYZE_INST Procedure</a> on page 16-6	Creates an ADDM task for analyzing in instance analysis mode and executes it.
<a href="#">ANALYZE_PARTIAL Procedure</a> on page 16-7	Creates an ADDM task for analyzing a subset of instances in partial analysis mode and executes it
<a href="#">DELETE Procedure</a> on page 16-8	Deletes an already created ADDM task (of any kind)
<a href="#">DELETE_FINDING_DIRECTIVE Procedure</a> on page 16-9	Deletes a finding directive
<a href="#">DELETE_PARAMETER_DIRECTIVE Procedure</a> on page 16-10	Deletes a parameter directive
<a href="#">DELETE_SEGMENT_DIRECTIVE Procedure</a> on page 16-11	Deletes a segment directive
<a href="#">DELETE_SQL_DIRECTIVE Procedure</a> on page 16-12	Deletes a SQL directive
<a href="#">GET_ASH_QUERY Function</a> on page 16-13	Returns a string containing the SQL text of an ASH query identifying the rows in ASH with impact for the finding
<a href="#">GET_REPORT Function</a> on page 16-14	Retrieves the default text report of an executed ADDM task
<a href="#">INSERT_FINDING_DIRECTIVE Procedure</a> on page 16-15	Creates a directive to limit reporting of a specific finding type.
<a href="#">INSERT_PARAMETER_DIRECTIVE Procedure</a> on page 16-16	Creates a directive to prevent ADDM from creating actions to alter the value of a specific system parameter
<a href="#">INSERT_SEGMENT_DIRECTIVE Procedure</a> on page 16-17	Creates a directive to prevent ADDM from creating actions to "run Segment Advisor" for specific segments
<a href="#">INSERT_SQL_DIRECTIVE Procedure</a> on page 16-19	Creates a directive to limit reporting of actions on specific SQL

## ANALYZE\_DB Procedure

This procedure creates an ADDM task for analyzing in database analysis mode and executes it.

### Syntax

```
DBMS_ADDM.ANALYZE_DB (
  task_name          IN OUT VARCHAR2,
  begin_snapshot     IN      NUMBER,
  end_snapshot       IN      NUMBER,
  db_id              IN      NUMBER := NULL);
```

### Parameters

**Table 16–2 ANALYZE\_DB Procedure Parameters**

Parameter	Description
task_name	Name of the task to be created
begin_snapshot	Number of the snapshot that starts the analysis period
end_snapshot	Number of the snapshot that ends the analysis period
db_id	Database ID for the database you to analyze. By default, this is the database currently connected

### Return Values

The name of the created task is returned in the `task_name` parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

### Examples

To create an ADDM task in database analysis mode and execute it, with its name in variable `tname`:

```
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_database_analysis_mode_task';
  DBMS_ADDM.ANALYZE_DB(:tname, 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

## ANALYZE\_INST Procedure

This procedure creates an ADDM task for analyzing in instance analysis mode and executes it.

### Syntax

```
DBMS_ADDM.ANALYZE_INST (
  task_name          IN OUT VARCHAR2,
  begin_snapshot     IN      NUMBER,
  end_snapshot       IN      NUMBER,
  instance_number    IN      NUMBER := NULL,
  db_id              IN      NUMBER := NULL);
```

### Parameters

**Table 16–3 ANALYZE\_INST Procedure Parameters**

Parameter	Description
task_name	Name of the task to be created
begin_snapshot	Number of the snapshot that starts the analysis period
end_snapshot	Number of the snapshot that ends the analysis period
instance_number	Number of the instance to analyze. By default it is the instance currently connected
db_id	Database ID for the database you to analyze. By default, this is the database currently connected

### Return Values

The name of the created task is returned in the `task_name` parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

### Usage Notes

On single instance systems (when not using Oracle RAC) the resulting task is identical to using the `ANALYZE_DB` procedure.

### Examples

To create an ADDM task in instance analysis mode and execute it, with its name in variable `tname`:

```
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

## ANALYZE\_PARTIAL Procedure

This procedure creates an ADDM task for analyzing a subset of instances in partial analysis mode and executes it.

### Syntax

```
DBMS_ADDM.ANALYZE_PARTIAL (
  task_name          IN OUT VARCHAR2,
  instance_numbers  IN      VARCHAR2,
  begin_snapshot     IN      NUMBER,
  end_snapshot       IN      NUMBER,
  db_id              IN      NUMBER := NULL);
```

### Parameters

**Table 16–4 ANALYZE\_PARTIAL Procedure Parameters**

Parameter	Description
task_name	Name of the task to be created
instance_numbers	Comma separated list of instance numbers to analyze
begin_snapshot	Number of the snapshot that starts the analysis period
end_snapshot	Number of the snapshot that ends the analysis period
db_id	Database ID for the database you to analyze. By default, this is the database currently connected

### Return Values

The name of the created task is returned in the `task_name` parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

### Examples

To create an ADDM task in partial analysis mode and execute it, with its name in variable `tname`:

```
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_partial_analysis_modetask';
  DBMS_ADDM.ANALYZE_PARTIAL(:tname, '1,2,3', 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

## DELETE Procedure

This procedure deletes an already created ADDM task (of any kind). For database analysis mode and partial analysis mode this deletes the local tasks associated with the main task.

### Syntax

```
DBMS_ADDM.DELETE (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 16–5** *DELETE Procedure Parameters*

Parameter	Description
task_name	Name of the task to be deleted

### Examples

```
BEGIN  
    DBMS_ADDM.DELETE ('my_partial_analysis_mode_task');  
END
```

## DELETE\_FINDING\_DIRECTIVE Procedure

This procedure deletes a finding directive.

### Syntax

```
DBMS_ADDM.DELETE_FINDING_DIRECTIVE (  
    task_name          IN VARCHAR2,  
    dir_name           IN VARCHAR2);
```

### Parameters

**Table 16–6** *DELETE\_FINDING\_DIRECTIVE Procedure Parameters*

Parameter	Description
task_name	Name of the task this directive applies to. If the value is NULL, it is a system directive.
dir_name	Name of the directive. All directives must be given unique names.

## DELETE\_PARAMETER\_DIRECTIVE Procedure

This procedure deletes a parameter directive. This removes a specific system directive for parameters. Subsequent ADDM tasks are not affected by this directive.

### Syntax

```
DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE (  
    task_name          IN VARCHAR2,  
    dir_name           IN VARCHAR2);
```

### Parameters

**Table 16–7** *DELETE\_PARAMETER\_DIRECTIVE Procedure Parameters*

Parameter	Description
task_name	Name of the task this directive applies to. If the value is NULL, it is a system directive.
dir_name	Name of the directive. All directives must be given unique names.

### Examples

```
BEGIN  
    DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE (NULL, 'my Parameter directive');  
END;
```



## DELETE\_SEGMENT\_DIRECTIVE Procedure

This procedure deletes a segment directive.

### Syntax

```
DBMS_ADDM.DELETE_SEGMENT_DIRECTIVE (  
    task_name          IN VARCHAR2,  
    dir_name           IN VARCHAR2);
```

### Parameters

**Table 16–8** *DELETE\_SEGMENT\_DIRECTIVE Procedure Parameters*

Parameter	Description
task_name	Name of the task this directive applies to. If the value is NULL, it is a system directive.
dir_name	Name of the directive. All directives must be given unique names.

## DELETE\_SQL\_DIRECTIVE Procedure

This procedure deletes a SQL directive.

### Syntax

```
DBMS_ADDM.DELETE_SQL_DIRECTIVE (  
    task_name          IN VARCHAR2,  
    dir_name           IN VARCHAR2);
```

### Parameters

**Table 16–9** *DELETE\_SQL\_DIRECTIVE Procedure Parameters*

Parameter	Description
task_name	Name of the task this directive applies to. If the value is NULL, it is a system directive.
dir_name	Name of the directive. All directives must be given unique names.

## GET\_ASH\_QUERY Function

The function returns a string containing the SQL text of an ASH query identifying the rows in ASH with impact for the finding. For most types of findings this identifies the exact rows in ASH corresponding to the finding. For some types of findings the query is an approximation and should not be used for exact identification of the finding's impact or the finding's specific activity.

### Syntax

```
DBMS_ADDM.GET_ASH_QUERY (  
    task_name          IN   VARCHAR2,  
    finding_id         IN   NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 16–10** GET\_ASH\_QUERY Function Parameters

Parameter	Description
task_name	Name of the task
finding	ID of the finding within the task

### Return Values

A VARCHAR containing an ASH query identifying the rows in ASH with impact for the finding

## GET\_REPORT Function

This function retrieves the default text report of an executed ADDM task.

### Syntax

```
DBMS_ADDM.GET_REPORT (  
    task_name          IN VARCHAR2)  
    RETURN CLOB;
```

### Parameters

**Table 16–11** *GET\_REPORT Function Parameters*

Parameter	Description
task_name	Name of the task

### Examples

```
Set long 1000000  
Set pagesize 50000  
SELECT DBMS_ADDM.GET_REPORT('my_partial_analysis_mode_task') FROM DUAL;
```

## INSERT\_FINDING\_DIRECTIVE Procedure

This procedure creates a directive to limit reporting of a specific finding type. The directive can be created for a specific task (only when the task is in INITIAL status), or for all subsequently created ADDM tasks (such as a system directive).

### Syntax

```
DBMS_ADDM.INSERT_FINDING_DIRECTIVE (
  task_name          IN VARCHAR2,
  dir_name           IN VARCHAR2,
  finding_name       IN VARCHAR2,
  min_active_sessions IN NUMBER := 0,
  min_perc_impact    IN NUMBER := 0);
```

### Parameters

**Table 16–12** INSERT\_FINDING\_DIRECTIVE Procedure Parameters

Parameter	Description
task_name	Name of the task this directive applies to. If the value is NULL, it applies to all subsequently created ADDM Tasks.
dir_name	Name of the directive. All directives must be given unique names.
finding_name	Name of an ADDM finding to which this directive applies. All valid findings names appear in the NAME column of view DBA_ADVISOR_FINDING_NAMES.
min_active_sessions	Minimal number of active sessions for the finding. If a finding has less than this number, it is filtered from the ADDM result.
min_perc_impact	Minimal number for the "percent impact" of the finding relative to total database time in the analysis period. If the finding's impact is less than this number, it is filtered from the ADDM result.

### Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for 'Undersized SGA' findings. The result of GET\_REPORT shows only an 'Undersized SGA' finding if the finding is responsible for at least 2 average active sessions during the analysis period, and this constitutes at least 10% of the total database time during that period.

```
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_FINDING_DIRECTIVE(
    NULL,
    'Undersized SGA directive',
    'Undersized SGA',
    2,
    10);
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing 'Undersized SGA' findings regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

## INSERT\_PARAMETER\_DIRECTIVE Procedure

This procedure creates a directive to prevent ADDM from creating actions to alter the value of a specific system parameter. The directive can be created for a specific task (only when the task is in `INITIAL` status), or for all subsequently created ADDM tasks (such as a system directive).

### Syntax

```
DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE (
    task_name          IN VARCHAR2,
    dir_name           IN VARCHAR2,
    parameter_name     IN VARCHAR2);
```

### Parameters

**Table 16–13** *INSERT\_PARAMETER\_DIRECTIVE Procedure Parameters*

Parameter	Description
<code>task_name</code>	Name of the task this directive applies to. If the value is <code>NULL</code> , it applies to all subsequently created ADDM Tasks.
<code>dir_name</code>	Name of the directive. All directives must be given unique names.
<code>parameter_name</code>	Specifies the parameter to use. Valid parameter names appear in <code>V\$PARAMETER</code> .

### Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for all actions that recommend modifying the parameter `'sga_target'`. The result of `GET_REPORT` does not show these actions.

```
var tname varchar2(60);
BEGIN
    DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE(
        NULL,
        'my Parameter directive',
        'sga_target');
    :tname := 'my_instance_analysis_mode_task';
    DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

## INSERT\_SEGMENT\_DIRECTIVE Procedure

This procedure creates a directive to prevent ADDM from creating actions to "run Segment Advisor" for specific segments. The directive can be created for a specific task (only when the task is in `INITIAL` status), or for all subsequently created ADDM tasks (such as a system directive).

### Syntax

```
DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE (
  task_name          IN VARCHAR2,
  dir_name           IN VARCHAR2,
  owner_name        IN VARCHAR2,
  object_name       IN VARCHAR2 := NULL,
  sub_object_name   IN VARCHAR2 := NULL);
```

```
DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE (
  task_name          IN VARCHAR2,
  dir_name           IN VARCHAR2,
  object_number     IN NUMBER);
```

### Parameters

**Table 16–14** *INSERT\_SEGMENT\_DIRECTIVE Procedure Parameters*

Parameter	Description
<code>task_name</code>	Name of the task this directive applies to. If the value is <code>NULL</code> , it applies to all subsequently created ADDM Tasks.
<code>dir_name</code>	Name of the directive. All directives must be given unique names.
<code>owner_name</code>	Specifies the owner of the segment/s to be filtered. A wildcard is allowed in the same syntax used for "like" constraints.
<code>object_name</code>	Name of the main object to be filtered. Again, wildcards are allowed. The default value of <code>NULL</code> is equivalent to a value of <code>'%'</code> .
<code>sub_object_name</code>	Name of the part of the main object to be filtered. This could be a partition name, or even sub partitions (separated by a <code>'.'</code> ). Again, wildcards are allowed. The default value of <code>NULL</code> is equivalent to a value of <code>'%'</code> .
<code>object_number</code>	Object number of the <code>SEGMENT</code> that this directive is to filter, found in views <code>DBA_OBJECTS</code> or <code>DBA_SEGMENTS</code>

### Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for all segments that belong to user `SCOTT`. The result of `GET_REPORT` does not show actions for running Segment advisor for segments that belong to `SCOTT`.

```
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE(NULL,
                                     'my Segment directive',
                                     'SCOTT');
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```



## INSERT\_SQL\_DIRECTIVE Procedure

This procedure creates a directive to limit reporting of actions on specific SQL. The directive can be created for a specific task (only when the task is in INITIAL status), or for all subsequently created ADDM tasks (such as a system directive).

### Syntax

```
DBMS_ADDM.INSERT_SQL_DIRECTIVE (
  task_name          IN VARCHAR2,
  dir_name           IN VARCHAR2,
  sql_id             IN VARCHAR2,
  min_active_sessions IN NUMBER := 0,
  min_response_time  IN NUMBER := 0);
```

### Parameters

**Table 16–15** INSERT\_SQL\_DIRECTIVE Procedure Parameters

Parameter	Description
task_name	Name of the task this directive applies to. If the value is NULL, it applies to all subsequently created ADDM Tasks.
dir_name	Name of the directive. All directives must be given unique names.
sql_id	Identifies which SQL statement to filter. A valid value contains exactly 13 characters from '0' to '9' and 'a' to 'z'.
min_active_sessions	Minimal number of active sessions for the SQL. If a SQL action has less than this number, it is filtered from the ADDM result.
min_response_time	Minimal value for response time of the SQL (in microseconds). If the SQL had lower response time, it is filtered from the ADDM result.

### Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for SQL with id 'abcd123456789'. The result of GET\_REPORT shows only actions for that SQL (actions to tune the SQL, or to investigate application using it) if the SQL is responsible for at least 2 average active sessions during the analysis period, and the average response time was at least 1 second.

```
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_SQL_DIRECTIVE (
    NULL,
    'my SQL directive',
    'abcd123456789',
    2,
    1000000);
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```



---

---

## DBMS\_ADVANCED\_REWRITE

DBMS\_ADVANCED\_REWRITE contains interfaces for advanced query rewrite users. Using this package, you can create, drop, and maintain functional equivalence declarations for query rewrite.

**See Also:** *Oracle Database Data Warehousing Guide* for more information about query rewrite

This chapter contains the following topics:

- [Using DBMS\\_ADVANCED\\_REWRITE](#)
  - Security Model
- [Summary of DBMS\\_ADVANCED\\_REWRITE Subprograms](#)

---

## Using DBMS\_ADVANCED\_REWRITE

This section contains topics which relate to using the DBMS\_ADVANCED\_REWRITE package.

- [Security Model](#)

## Security Model

No privileges to access these procedures are granted to anyone by default. To gain access to these procedures, you must connect as SYSDBA and explicitly grant execute access to the desired database administrators.

You can control security on this package by granting the EXECUTE privilege to selected database administrators or roles. For example, the user er can be given access to use this package by the following statement, executed as SYSDBA:

```
GRANT EXECUTE ON DBMS_ADVANCED_REWRITE TO er;
```

You may want to write a separate cover package on top of this package for restricting the alert names used. Instead of granting the EXECUTE privilege on the DBMS\_ADVANCED\_REWRITE package directly, you can then grant it to the cover package.

In addition, similar to the privilege required for regular materialized views, the user should be granted the privilege to create an equivalence. For example, the user er can be granted this privilege by executing the following statement as SYSDBA:

```
GRANT CREATE MATERIALIZED VIEW TO er;
```

## Summary of DBMS\_ADVANCED\_REWRITE Subprograms

This table lists all the package subprograms in alphabetical order.

**Table 17-1 DBMS\_ADVANCED\_REWRITE Package Subprograms**

Subprogram	Description
<a href="#">ALTER_REWRITE_EQUIVALENCE</a> Procedure on page 17-5	Changes the mode of the rewrite equivalence declaration to the mode you specify
<a href="#">BUILD_SAFE_REWRITE_EQUIVALENCE</a> Procedure on page 17-6	Enables the rewrite of top-level materialized views using submaterialized views. Oracle Corporation does not recommend you directly use this procedure
<a href="#">DECLARE_REWRITE_EQUIVALENCE</a> Procedures on page 17-7	Creates a declaration indicating that <code>source_stmt</code> is functionally equivalent to <code>destination_stmt</code> for as long as the equivalence declaration remains enabled, and that <code>destination_stmt</code> is more favorable in terms of performance
<a href="#">DROP_REWRITE_EQUIVALENCE</a> Procedure on page 17-9	Drops the specified rewrite equivalence declaration
<a href="#">VALIDATE_REWRITE_EQUIVALENCE</a> Procedure on page 17-10	Validates the specified rewrite equivalence declaration using the same validation method as described with the <code>validate</code> parameter

## ALTER\_REWRITE\_EQUIVALENCE Procedure

This procedure changes the mode of the rewrite equivalence declaration to the mode you specify.

### Syntax

```
DBMS_ADVANCED_REWRITE.ALTER_REWRITE_EQUIVALENCE (
    name          VARCHAR2,
    rewrite_mode  VARCHAR2);
```

### Parameters

**Table 17–2 ALTER\_REWRITE\_EQUIVALENCE Procedure Parameters**

Parameter	Description
name	A name for the equivalence declaration to alter. The name can be of the form <code>owner . name</code> , where <code>owner</code> complies with the rules for a schema name, and <code>name</code> complies with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is altered in the current schema. The invoker must have the appropriate alter materialized view privileges to alter an equivalence declaration outside their own schema.
rewrite_mode	The following modes are supported, in increasing order of power: <ul style="list-style-type: none"> <li><code>disabled</code>: Query rewrite does not use the equivalence declaration. Use this mode to temporarily disable use of the rewrite equivalence declaration.</li> <li><code>text_match</code>: Query rewrite uses the equivalence declaration only in its text match modes. This mode is useful for simple transformations.</li> <li><code>general</code>: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. However, query rewrite makes no attempt to rewrite the specified <code>destination_query</code>.</li> <li><code>recursive</code>: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. Moreover, query rewrite further attempts to rewrite the specified <code>destination_query</code> for further performance enhancements whenever it uses the equivalence declaration.</li> </ul> Oracle recommends you use the least powerful mode that is sufficient to solve your performance problem.

## **BUILD\_SAFE\_REWRITE\_EQUIVALENCE Procedure**

This procedure enables the rewrite and refresh of top-level materialized views using submaterialized views. It is provided for the exclusive use by scripts generated by the `DBMS_ADVISOR.TUNE_MVIEW` procedure. It is required to enable query rewrite and fast refresh when `DBMS_ADVISOR.TUNE_MVIEW` decomposes a materialized view into a top-level materialized view and one or more submaterialized views.

Oracle does not recommend you directly use the `BUILD_SAFE_REWRITE_EQUIVALENCE` procedure. You should use either the `DBMS_ADVISOR.TUNE_MVIEW` or the `DBMS_ADVANCED_REWRITE.CREATE_REWRITE_EQUIVALENCE` procedure as appropriate.



## DECLARE\_REWRITE\_EQUIVALENCE Procedures

This procedure creates a declaration indicating that `source_stmt` is functionally equivalent to `destination_stmt` for as long as the equivalence declaration remains enabled, and that `destination_stmt` is more favorable in terms of performance. The scope of the declaration is system wide. The query rewrite engine uses such declarations to perform rewrite transformations in `QUERY_REWRITE_INTEGRITY = trusted` and `stale_tolerated` modes.

Because the underlying equivalences between the source and destination statements cannot be enforced by the query rewrite engine, queries can be only rewritten in `trusted` and `stale_tolerated` integrity modes.

### Syntax

```
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
    name                VARCHAR2,
    source_stmt         VARCHAR2,
    destination_stmt    VARCHAR2,
    validate            BOOLEAN    := TRUE,
    rewrite_mode        VARCHAR2  := 'TEXT_MATCH');
```

```
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE (
    name                VARCHAR2,
    source_stmt         CLOB,
    destination_stmt    CLOB,
    validate            BOOLEAN    := TRUE,
    rewrite_mode        VARCHAR2  := 'TEXT_MATCH');
```

### Parameters

**Table 17–3** *DECLARE\_REWRITE\_EQUIVALENCE Procedure Parameters*

Parameter	Description
<code>name</code>	A name for the equivalence declaration. The name can be of the form <code>owner.name</code> , where <code>owner</code> complies with the rules for a schema name, and <code>name</code> complies with the rules for a table name.  Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is created in the current schema. The invoker must have the appropriate <code>CREATE MATERIALIZED VIEW</code> privileges to alter an equivalence declaration.
<code>source_stmt</code>	A sub- <code>SELECT</code> expression in either <code>VARCHAR2</code> or <code>CLOB</code> format. This is the query statement that is the target of optimization.
<code>destination_stmt</code>	A sub- <code>SELECT</code> expression in either <code>VARCHAR2</code> or <code>CLOB</code> format.
<code>validate</code>	A Boolean indicating whether to validate that the specified <code>source_stmt</code> is functionally equivalent to the specified <code>destination_stmt</code> . If <code>validate</code> is specified as <code>TRUE</code> , <code>DECLARE_REWRITE_EQUIVALENCE</code> evaluates the two sub- <code>SELECT</code> s and compares their results. If the results are not the same, <code>DECLARE_REWRITE_EQUIVALENCE</code> does not create the rewrite equivalence and returns an error condition. If <code>FALSE</code> , <code>DECLARE_REWRITE_EQUIVALENCE</code> does not validate the equivalence.

**Table 17-3 (Cont.) DECLARE\_REWRITE\_EQUIVALENCE Procedure Parameters**

Parameter	Description
rewrite_mode	<p>The following modes are supported, in increasing order of power:</p> <ul style="list-style-type: none"> <li>▪ disabled: Query rewrite does not use the equivalence declaration. Use this mode to temporarily disable use of the rewrite equivalence declaration.</li> <li>▪ text_match: Query rewrite uses the equivalence declaration only in its text match modes. This mode is useful for simple transformations.</li> <li>▪ general: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. However, query rewrite makes no attempt to rewrite the specified destination_query.</li> <li>▪ recursive: Query rewrite uses the equivalence declaration in all of its transformation modes against the incoming request queries. Moreover, query rewrite further attempts to rewrite the specified destination_query for further performance enhancements whenever it uses the equivalence declaration.</li> </ul> <p>Oracle recommends you use the least powerful mode that is sufficient to solve your performance problem.</p>

## Exceptions

**Table 17-4 DECLARE\_REWRITE\_EQUIVALENCE Procedure Exceptions**

Exception	Description
ORA-30388	Name of the rewrite equivalence is not specified
ORA-30391	The specified rewrite equivalence does not exist
ORA-30392	The checksum analysis for the rewrite equivalence failed
ORA-30393	A query block in the statement did not write
ORA-30396	Rewrite equivalence procedures require the COMPATIBLE parameter to be set to 10.1 or greater

## Usage Notes

Query rewrite using equivalence declarations occurs simultaneously and in concert with query rewrite using materialized views. The same query rewrite engine is used for both. The query rewrite engine uses the same rewrite rules to rewrite queries using both equivalence declarations and materialized views. Because the rewrite equivalence represents a specific rewrite crafted by a sophisticated user, the query rewrite engine gives priority to rewrite equivalences over materialized views when it is possible to perform a rewrite with either a materialized view or a rewrite equivalence. For this same reason, the cost-based optimizer (specifically, cost-based rewrite) will not choose an unrewritten query plan over a query plan that is rewritten to use a rewrite equivalence even if the cost of the unrewritten plan appears more favorable. Query rewrite matches properties of the incoming request query against the equivalence declaration's source\_stmt or the materialized view's defining statement, respectively, and derives an equivalent relational expression in terms of the equivalence declaration's destination\_stmt or the materialized view's container table, respectively.

## DROP\_REWRITE\_EQUIVALENCE Procedure

This procedure drops the specified rewrite equivalence declaration.

### Syntax

```
DBMS_ADVANCED_REWRITE.DROP_REWRITE_EQUIVALENCE (  
    name          VARCHAR2);
```

### Parameters

**Table 17–5** *DROP\_REWRITE\_EQUIVALENCE Procedure Parameters*

Parameter	Description
name	A name for the equivalence declaration to drop. The name can be of the form <code>owner.name</code> , where <code>owner</code> complies with the rules for a schema name, and <code>name</code> complies with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is dropped in the current schema. The invoker must have the appropriate drop materialized view privilege to drop an equivalence declaration outside their own schema.

## VALIDATE\_REWRITE\_EQUIVALENCE Procedure

This procedure validates the specified rewrite equivalence declaration using the same validation method as described with the `VALIDATE` parameter in "[VALIDATE\\_REWRITE\\_EQUIVALENCE Procedure](#)" on page 17-10.

### Syntax

```
DBMS_ADVANCED_REWRITE.VALIDATE_REWRITE_EQUIVALENCE (  
    name          VARCHAR2);
```

### Parameters

**Table 17–6** *VALIDATE\_REWRITE\_EQUIVALENCE Procedure Parameters*

Parameter	Description
name	A name for the equivalence declaration to validate. The name can be of the form <code>owner.name</code> , where <code>owner</code> complies with the rules for a schema name, and <code>name</code> complies with the rules for a table name. Alternatively, a simple name that complies with the rules for a table name can be specified. In this case, the rewrite equivalence is validated in the current schema. The invoker must have sufficient privileges to execute both the <code>source_stmt</code> and <code>destination_stmt</code> of the specified equivalence declaration.

---

---

## DBMS\_ADVISOR

DBMS\_ADVISOR is part of the server manageability suite of advisors, a set of expert systems that identifies and helps resolve performance problems relating to database server components.

Some advisors have their own packages. For these advisors, Oracle recommends that you use the advisor-specific package rather than DBMS\_ADVISOR. Each of the following advisors has its own package, tailored to its specific functionality:

- Automatic Database Diagnostic Monitor (DBMS\_ADDM)
- SQL Performance Analyzer (DBMS\_SQLPA)
- SQL Repair Advisor (DBMS\_SQLDIAG)
- SQL Tuning Advisor (DBMS\_SQLTUNE)
- Compression Advisor (DBMS\_COMPRESSION.GET\_COMPRESSION\_RATIO)

SQL Access Advisor and Segment Advisor are the only advisors with common use cases for DBMS\_ADVISOR. Undo Advisor and Compression Advisor do not support DBMS\_ADVISOR subprograms.

### See Also:

- *Oracle Database Administrator's Guide* to learn about Segment Advisor
- *Oracle Database 2 Day + Performance Tuning Guide* for information regarding how to use various advisors in Enterprise Manager
- *Oracle Database Performance Tuning Guide* for information regarding SQL Tuning Advisor and SQL Access Advisor
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn about Undo Advisor

This chapter contains the following topics:

- [Using DBMS\\_ADVISOR](#)
  - Deprecated Subprograms
  - Security Model
- [Summary of DBMS\\_ADVISOR Subprograms](#)

---

## Using DBMS\_ADVISOR

This section contains topics which relate to using the DBMS\_ADVISOR package.

- [Deprecated Subprograms](#)
- [Security Model](#)

## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

---

The following subprograms are deprecated with Oracle Database 11g:

- [ADD\\_SQLWKLD\\_REF Procedure](#)
- [CREATE\\_SQLWKLD Procedure](#)
- [DELETE\\_SQLWKLD Procedure](#)
- [DELETE\\_SQLWKLD\\_REF Procedure](#)
- [DELETE\\_SQLWKLD\\_STATEMENT Procedures](#)
- [IMPORT\\_SQLWKLD\\_SCHEMA Procedure](#)
- [IMPORT\\_SQLWKLD\\_SQLCACHE Procedure](#)
- [IMPORT\\_SQLWKLD\\_STS Procedure](#)
- [IMPORT\\_SQLWKLD\\_SUMADV Procedure](#)
- [IMPORT\\_SQLWKLD\\_USER Procedure](#)
- [RESET\\_SQLWKLD Procedure](#)
- [SET\\_SQLWKLD\\_PARAMETER Procedures](#)
- [UPDATE\\_SQLWKLD\\_ATTRIBUTES Procedure](#)
- [UPDATE\\_SQLWKLD\\_STATEMENT Procedure](#)

## Security Model

The `ADVISOR` privilege is required to use this package.



## Summary of DBMS\_ADVISOR Subprograms

Table 18–1 summarizes the subprograms in this package. The `Used in` column lists advisors relevant for each subprogram, but excludes ADDM, SQL Performance Analyzer, SQL Repair Advisor, and SQL Tuning Advisor because these advisors have their own packages.

**Table 18–1 DBMS\_ADVISOR Package Subprograms**

Subprogram	Description	Used in
<a href="#">ADD_SQLWKLD_REF Procedure</a> on page 18-8	Adds a workload reference to an Advisor task (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">ADD_SQLWKLD_STATEMENT Procedure</a> on page 18-9	Adds a single statement to a workload	SQL Access Advisor
<a href="#">ADD_STS_REF Procedure</a> on page 18-11	Establishes a link between the current SQL Access Advisor task and a SQL tuning set	SQL Access Advisor
<a href="#">CANCEL_TASK Procedure</a> on page 18-12	Cancels a currently executing task operation	Segment Advisor, SQL Access Advisor
<a href="#">COPY_SQLWKLD_TO_STS Procedure</a> on page 18-13	Copies the contents of a SQL workload object to a SQL tuning set	SQL Access Advisor
<a href="#">CREATE_FILE Procedure</a> on page 18-14	Creates an external file from a PL/SQL CLOB variable, which is useful for creating scripts and reports	SQL Access Advisor
<a href="#">CREATE_OBJECT Procedure</a> on page 18-15	Creates a new task object	Segment Advisor
<a href="#">CREATE_SQLWKLD Procedure</a> on page 18-17	Creates a new workload object (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">CREATE_TASK Procedures</a> on page 18-18	Creates a new Advisor task in the repository	Segment Advisor, SQL Access Advisor
<a href="#">DELETE_SQLWKLD Procedure</a> on page 18-20	Deletes an entire workload object (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">DELETE_SQLWKLD_REF Procedure</a> on page 18-21	Deletes an entire workload object (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">DELETE_SQLWKLD_STATEMENT Procedures</a> on page 18-22	Deletes one or more statements from a workload (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">DELETE_STS_REF Procedure</a> on page 18-23	Removes a link between the current SQL Access Advisor task and a SQL tuning set object	SQL Access Advisor
<a href="#">DELETE_TASK Procedure</a> on page 18-24	Deletes the specified task from the repository	SQL Access Advisor
<a href="#">EXECUTE_TASK Procedure</a> on page 18-25	Executes the specified task	Segment Advisor, SQL Access Advisor
<a href="#">GET_REC_ATTRIBUTES Procedure</a> on page 18-27	Retrieves specific recommendation attributes from a task	SQL Access Advisor

**Table 18–1 (Cont.) DBMS\_ADVISOR Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Used in</b>
<a href="#">GET_TASK_REPORT Function</a> on page 18-28	Creates and returns a report for the specified task	
<a href="#">GET_TASK_SCRIPT Function</a> on page 18-29	Creates and returns an executable SQL script of the Advisor task's recommendations in a buffer	SQL Access Advisor
<a href="#">IMPLEMENT_TASK Procedure</a> on page 18-31	Implements the recommendations for a task	SQL Access Advisor
<a href="#">IMPORT_SQLWKLD_SCHEMA Procedure</a> on page 18-32	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">IMPORT_SQLWKLD_SQLCACHE Procedure</a> on page 18-34	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">IMPORT_SQLWKLD_STS Procedure</a> on page 18-36	Imports data from a SQL tuning set into a SQL workload data object (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">IMPORT_SQLWKLD_SUMADV Procedure</a> on page 18-38	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">IMPORT_SQLWKLD_USER Procedure</a> on page 18-40	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">INTERRUPT_TASK Procedure</a> on page 18-42	Stops a currently executing task, ending its operations as it would at a normal exit, so that the recommendations are visible	Segment Advisor, SQL Access Advisor
<a href="#">MARK_RECOMMENDATION Procedure</a> on page 18-43	Sets the <code>annotation_status</code> for a particular recommendation	Segment Advisor, SQL Access Advisor
<a href="#">QUICK_TUNE Procedure</a> on page 18-44	Performs an analysis on a single SQL statement	SQL Access Advisor
<a href="#">RESET_SQLWKLD Procedure</a> on page 18-45	Resets a workload to its initial starting point (Caution: Deprecated Subprogram)	SQL Access Advisor
<a href="#">RESET_TASK Procedure</a> on page 18-46	Resets a task to its initial state	Segment Advisor, SQL Access Advisor
<a href="#">SET_DEFAULT_SQLWKLD_PARAMETER Procedures</a> on page 18-47	Imports data into a workload from schema evidence	SQL Access Advisor
<a href="#">SET_DEFAULT_TASK_PARAMETER Procedures</a> on page 18-48	Modifies a default task parameter	Segment Advisor, SQL Access Advisor
<a href="#">SET_SQLWKLD_PARAMETER Procedures</a> on page 18-49	Sets the value of a workload parameter	SQL Access Advisor
<a href="#">SET_TASK_PARAMETER Procedure</a> on page 18-50	Sets the specified task parameter value	Segment Advisor, SQL Access Advisor
<a href="#">TUNE_MVIEW Procedure</a> on page 18-63	Shows how to decompose a materialized view into two or more materialized views or to restate the materialized view in a way that is more advantageous for fast refresh and query rewrite	SQL Access Advisor

**Table 18-1 (Cont.) DBMS\_ADVISOR Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Used in</b>
<a href="#">UPDATE_OBJECT Procedure</a> on page 18-65	Updates a task object	Segment Advisor
<a href="#">UPDATE_REC_ATTRIBUTES Procedure</a> on page 18-67	Updates an existing recommendation for the specified task	SQL Access Advisor
<a href="#">UPDATE_SQLWKLD_ATTRIBUTES Procedure</a> on page 18-69	Updates a workload object	SQL Access Advisor
<a href="#">UPDATE_SQLWKLD_STATEMENT Procedure</a> on page 18-70	Updates one or more SQL statements in a workload	SQL Access Advisor
<a href="#">UPDATE_TASK_ATTRIBUTES Procedure</a> on page 18-72	Updates a task's attributes	Segment Advisor, SQL Access Advisor

## ADD\_SQLWKLD\_REF Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure establishes a link between the current SQL Access Advisor task and a SQL Workload object. The link allows an advisor task to access interesting data for doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL Workload object is made, the workload is protected from removal or modification.

Users should use ADD\_STS\_REF instead of ADD\_SQLWKLD\_REF for all SQL tuning set-based advisor runs. This function is only provided for backward compatibility.

### Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_REF (
    task_name          IN VARCHAR2,
    workload_name      IN VARCHAR2,
    is_sts             IN NUMBER :=0);
```

### Parameters

**Table 18–2 ADD\_SQLWKLD\_REF Procedure Parameters**

Parameter	Description
task_name	The SQL Access Advisor task name that uniquely identifies an existing task.
workload_name	The name of the workload object to be linked. Once a object has been linked to a task, it becomes read-only and cannot be deleted. There is no limit to the number of links to workload objects. To remove the link to the workload object, use the procedure DELETE_REFERENCE.
is_sts	Indicates the type of workload source. Possible values are: <ul style="list-style-type: none"> <li>■ 0 - SQL workload object</li> <li>■ 1 - SQL tuning set</li> </ul>

### Examples

```
DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name, 1);
END;
```

## ADD\_SQLWKLD\_STATEMENT Procedure

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure adds a single statement to the specified workload.

### Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT (
  workload_name      IN VARCHAR2,
  module             IN VARCHAR2,
  action             IN VARCHAR2,
  cpu_time           IN NUMBER := 0,
  elapsed_time       IN NUMBER := 0,
  disk_reads         IN NUMBER := 0,
  buffer_gets        IN NUMBER := 0,
  rows_processed     IN NUMBER := 0,
  optimizer_cost     IN NUMBER := 0,
  executions         IN NUMBER := 1,
  priority           IN NUMBER := 2,
  last_execution_date IN DATE := 'SYSDATE',
  stat_period        IN NUMBER := 0,
  username           IN VARCHAR2,
  sql_text           IN CLOB);
```

### Parameters

**Table 18-3 ADD\_SQLWKLD\_STATEMENT Procedure Parameters**

Parameter	Description
workload_name	The workload name that uniquely identifies an existing workload.
module	An optional business application module that will be associated with the SQL statement.
action	An optional application action that will be associated with the SQL statement.
cpu_time	The total CPU time in seconds that is consumed by the SQL statement.
elapsed_time	The total elapsed time in seconds that is consumed by the SQL statement.
disk_reads	The total disk-read operations that are consumed by the SQL statement.
buffer_gets	The total buffer-get operations that are consumed by the SQL statement.
rows_processed	The average number of rows processed by the SQL statement.
optimizer_cost	The optimizer's calculated cost value.
executions	The total execution count by the SQL statement. This value should be greater than zero.
priority	The relative priority of the SQL statement. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.
last_execution_date	The date and time at which the SQL statement last executed. If the value is NULL, then the current date and time will be used.

**Table 18–3 (Cont.) ADD\_SQLWKLD\_STATEMENT Procedure Parameters**

Parameter	Description
stat_period	Time interval in seconds from which statement statistics were calculated.
username	The Oracle user name that executed the SQL statement. Because a username is an Oracle identifier, the username value must be entered exactly as it is stored in the server. For example, if the user SCOTT is the executing user, then you must provide the user identifier SCOTT in all uppercase letters. It will not recognize the user scott as a match for SCOTT.
sql_text	The complete SQL statement. To increase the quality of a recommendation, the SQL statement should not contain bind variables.

## Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 for directions on setting a task to its initial state.

The ADD\_SQLWKLD\_STATEMENT procedure accepts several parameters that may be ignored by the caller. `cpu_time`, `elapsed_time`, `disk_reads`, `buffer_gets`, and `optimizer_cost` are only used to sort workload data when actual analysis occurs, so actual values are only necessary when the `order_list` task parameter references a particular statistic.

To determine what statistics to provide when adding a new SQL statement to a workload, examine or set the task parameter `order_list`. The `order_list` parameter accepts any combination of the keys: `buffer_gets`, `optimizer_cost`, `cpu_time`, `disk_reads`, `elapsed_time`, `executions`, and `priority`. A typical setting of `priority`, `optimizer_cost` would indicate the SQL Access Advisor will sort the workload data by `priority` and `optimizer_cost` and process the highest cost statements first. Any statements added to the workload would need to include appropriate `priority` and `optimizer_cost` values. All other statistics can be defaulted or set to zero.

For the statistical keys referenced by the `order_list` task parameter, the actual parameter values should be reasonably accurate since they will be compared to other statements in the workload. If the caller is unable to estimate values, choose values that would determine its importance relative to other statements in the workload. For example, if the current statement is considered the most critical query in your business, then an appropriate value would be anything greater than all other values for the same statistic found in the workload.

## Examples

```

DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
FROM sh.sales');
END;
/

```

## ADD\_STS\_REF Procedure

This procedure establishes a link between the current SQL Access Advisor task and a SQL tuning set. The link enables an advisor task to access data for the purpose of doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL tuning set is made, the STS is protected from removal or modification.

Use ADD\_STS\_REF for any STS-based advisor runs. The older method of using ADD\_SQLWKLD\_REF with parameter IS\_STS=1 is only supported for backward compatibility. Furthermore, the ADD\_STS\_REF function accepts a SQL tuning set owner name, whereas ADD\_SQLWKLD\_REF does not.

### Syntax

```
DBMS_ADVISOR.ADD_STS_REF (
    task_name      IN VARCHAR2 NOT NULL,
    sts_owner      IN VARCHAR2,
    workload_name  IN VARCHAR2 NOT NULL);
```

### Parameters

**Table 18–4 ADD\_STS\_REF Procedure Parameters**

Parameter	Description
task_name	The SQL Access Advisor task name that uniquely identifies an existing task.
sts_owner	The owner of the SQL tuning set. The value of this parameter may be NULL, in which case the advisor assumes the SQL tuning set to be owned by the currently logged-in user.
workload_name	The name of the workload to be linked. A workload consists of one or more SQL statements, plus statistics and attributes that fully describe each statement. The database stores a workload as a SQL tuning set.  After a workload has been linked to a task, it becomes read-only and cannot be deleted.  There is no limit to the number of links to workloads.  To remove the link to the workload, use the procedure DBMS_ADVISOR.DELETE_STS_REF.

### Examples

```
DBMS_ADVISOR.ADD_STS_REF ('My Task', 'SCOTT', 'My Workload');
```

## CANCEL\_TASK Procedure

This procedure causes a currently executing operation to terminate. This call does a soft interrupt. It will not break into a low-level database access call like a hard interrupt such as `Ctrl-C`. The SQL Access Advisor periodically checks for soft interrupts and acts appropriately. As a result, this operation may take a few seconds to respond to a call.

### Syntax

```
DBMS_ADVISOR.CANCEL_TASK (  
    task_name      IN  VARCHAR2);
```

### Parameters

**Table 18–5** CANCEL\_TASK Procedure Parameter

Parameter	Description
task_name	A valid Advisor task name that uniquely identifies an existing task.

### Usage Notes

A cancel command restores the task to its condition prior to the start of the cancelled operation. Therefore, a cancelled task or data object cannot be resumed.

Because all Advisor task procedures are synchronous, to cancel an operation, you must use a separate database session.

### Examples

```
DECLARE  
    task_id NUMBER;  
    task_name VARCHAR2(30);  
    workload_name VARCHAR2(30);  
BEGIN  
    task_name := 'My Task';  
    workload_name := 'My Workload';  
  
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);  
    DBMS_ADVISOR.CANCEL_TASK('My Task');  
END;  
/
```



## COPY\_SQLWKLD\_TO\_STS Procedure

This procedure copies the contents of a SQL workload object to a SQL tuning set.

### Syntax

To use this procedure, the caller must have privileges to create and modify a SQL tuning set.

```
DBMS_ADVISOR.COPY_SQLWKLD_TO_STS (
  workload_name      IN VARCHAR2,
  sts_name           IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW');
```

### Parameters

**Table 18–6 COPY\_SQLWKLD\_TO\_STS Procedure Parameter**

Parameter	Description
workload_name	The SQL Workload object name to copy.
sts_name	The SQL tuning set name into which the SQL Workload object will be copied.
import_mode	Specifies the handling of the target SQL tuning set. Possible values are: <ul style="list-style-type: none"> <li>▪ APPEND Causes SQL Workload data to be appended to the target SQL tuning set.</li> <li>▪ NEW Indicates the SQL tuning set can only contain the copied contents. If the SQL tuning set exists and has data, an error will be reported.</li> <li>▪ REPLACE Causes any existing data in the target SQL tuning set to be purged prior to the workload copy.</li> </ul> In all cases, if the specified SQL tuning set does not exist, it will be created.

### Usage Notes

To use this procedure, the caller must have privileges to create and modify a SQL tuning set.

### Examples

```
BEGIN
  DBMS_ADVISOR.COPY_SQLWKLD_TO_STS('MY_OLD_WORKLOAD', 'MY_NEW_STS', 'NEW');
END;
/
```

## CREATE\_FILE Procedure

This procedure creates an external file from a PL/SQL CLOB variable, which is used for creating scripts and reports. CREATE\_FILE accepts a CLOB input parameter and writes the character string contents to the specified file.

### Syntax

```
DBMS_ADVISOR.CREATE_FILE (
    buffer      IN  CLOB,
    location    IN  VARCHAR2,
    filename    IN  VARCHAR2);
```

### Parameters

**Table 18–7 CREATE\_FILE Procedure Parameters**

Parameter	Description
buffer	A CLOB buffer containing report or script information.
location	Specifies the directory that will contain the new file. You must use the directory alias as defined by the CREATE DIRECTORY statement. The Advisor will translate the alias into the actual directory location.
filename	Specifies the output file to receive the script commands. The filename can only contain the name and an optional file type of the form filename.filetype.

### Usage Notes

All formatting must be embedded within the CLOB.

The Oracle server restricts file access within Oracle Stored Procedures. This means that file locations and names must adhere to the known file permissions in the server.

### Examples

```
CREATE DIRECTORY MY_DIR as '/homedir/user4/gssmith';
GRANT READ,WRITE ON DIRECTORY MY_DIR TO PUBLIC;

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
    workload_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                        100,400,5041,103,640445,680000,2,
                                        1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                        FROM sh.sales');

    DBMS_ADVISOR.EXECUTE_TASK(task_name);
    DBMS_ADVISOR.CREATE_FILE(DBMS_ADVISOR.GET_TASK_SCRIPT(task_name),
        'MY_DIR','script.sql');

END;
/
```

## CREATE\_OBJECT Procedure

This procedure creates a new task object.

### Syntax

```
DBMS_ADVISOR.CREATE_OBJECT (
  task_name          IN VARCHAR2,
  object_type        IN VARCHAR2,
  attr1              IN VARCHAR2 := NULL,
  attr2              IN VARCHAR2 := NULL,
  attr3              IN VARCHAR2 := NULL,
  attr4              IN CLOB      := NULL,
  object_id          OUT NUMBER);
```

```
DBMS_ADVISOR.CREATE_OBJECT (
  task_name          IN VARCHAR2,
  object_type        IN VARCHAR2,
  attr1              IN VARCHAR2 := NULL,
  attr2              IN VARCHAR2 := NULL,
  attr3              IN VARCHAR2 := NULL,
  attr4              IN CLOB      := NULL,
  attr5              IN VARCHAR2 := NULL,
  object_id          OUT NUMBER);
```

### Parameters

**Table 18–8 CREATE\_OBJECT Procedure Parameters**

Parameter	Description
task_name	A valid Advisor task name that uniquely identifies an existing task.
object_type	Specifies the external object type.
attr1	Advisor-specific data.
attr2	Advisor-specific data.
attr3	Advisor-specific data.
attr4	Advisor-specific data.
attr5	Advisor-specific data.
object_id	The advisor-assigned object identifier.

The attribute parameters have different values depending upon the object type. See *Oracle Database Administrator's Guide* for details regarding these parameters and object types.

### Return Values

Returns the new object identifier.

### Usage Notes

Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level. If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or

LOB column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

See *Oracle Database Administrator's Guide* for further information regarding the Segment Advisor.

## Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name, 'SQL', NULL, NULL, NULL,
                              'SELECT * FROM SH.SALES', obj_id);

END;
/
```

## CREATE\_SQLWKLD Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure creates a new private SQL Workload object for the user. A SQL Workload object manages a SQL workload on behalf of the SQL Access Advisor. A SQL Workload object must exist prior to performing any other SQL Workload operations, such as importing or updating SQL statements.

### Syntax

```
DBMS_ADVISOR.CREATE_SQLWKLD (
    workload_name      IN OUT VARCHAR2,
    description        IN VARCHAR2 := NULL,
    template           IN VARCHAR2 := NULL,
    is_template        IN VARCHAR2 := 'FALSE');
```

### Parameters

**Table 18–9 CREATE\_SQLWKLD Procedure Parameters**

Parameter	Description
workload_name	A name that uniquely identifies the created workload. If not specified, the system will generate a unique name. Names can be up to 30 characters long.
description	Specifies an optional workload description. Descriptions can be up to 256 characters.
template	An optional SQL Workload name of an existing workload data object or data object template.
is_template	An optional value that enables you to set the newly created workload as a template. Valid values are TRUE and FALSE.

### Return Values

The SQL Access Advisor returns a unique workload object identifier number that must be used for subsequent activities within the new SQL Workload object.

### Usage Notes

By default, workload objects are created using built-in default settings. To create a workload using the parameter settings of an existing workload or workload template, the user may specify an existing workload name.

After a SQL Workload object is present, it can then be referenced by one or more SQL Access Advisor tasks using the ADD\_SQLWKLD\_REF procedure.

### Examples

```
DECLARE
    workload_name VARCHAR2(30);
BEGIN
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
END;
/
```

## CREATE\_TASK Procedures

This procedure creates a new Advisor task in the repository.

### Syntax

```
DBMS_ADVISOR.CREATE_TASK (
  advisor_name      IN VARCHAR2,
  task_id           OUT NUMBER,
  task_name         IN OUT VARCHAR2,
  task_desc         IN VARCHAR2 := NULL,
  template          IN VARCHAR2 := NULL,
  is_template       IN VARCHAR2 := 'FALSE',
  how_created       IN VARCHAR2 := NULL);
```

```
DBMS_ADVISOR.CREATE_TASK (
  advisor_name      IN VARCHAR2,
  task_name         IN VARCHAR2,
  task_desc         IN VARCHAR2 := NULL,
  template          IN VARCHAR2 := NULL,
  is_template       IN VARCHAR2 := 'FALSE',
  how_created       IN VARCHAR2 := NULL);
```

```
DBMS_ADVISOR.CREATE_TASK (
  parent_task_name  IN VARCHAR2,
  rec_id            IN NUMBER,
  task_id           OUT NUMBER,
  task_name         IN OUT VARCHAR2,
  task_desc         IN VARCHAR2,
  template          IN VARCHAR2);
```

### Parameters

**Table 18–10 CREATE\_TASK Procedure Parameters**

Parameter	Description
advisor_name	Specifies the unique advisor name as defined in the view DBA_ADVISOR_DEFINITIONS.
task_id	A number that uniquely identifies the created task. The number is generated by the procedure and returned to the user.
task_name	Specifies a new task name. Names must be unique among all tasks for the user.  When using the second form of the CREATE_TASK syntax listed above (with OUT), a unique name can be generated. Names can be up to 30 characters long.
task_desc	Specifies an optional task description. Descriptions can be up to 256 characters in length.
template	An optional task name of an existing task or task template. To specify built-in SQL Access Advisor templates, use the template name as described earlier.
is_template	An optional value that allows the user to set the newly created task as template. Valid values are: TRUE and FALSE.
how_created	An optional value that identifies how the source was created.

### Return Values

Returns a unique task ID number and a unique task name if one is not specified.

## Usage Notes

A task must be associated with an advisor, and once the task has been created, it is permanently associated with the original advisor. By default, tasks are created using built-in default settings. To create a task using the parameter settings of an existing task or task template, the user may specify an existing task name.

For the SQL Access Advisor, use the identifier `DBMS_ADVISOR.SQLACCESS_ADVISOR` as the `advisor_name`.

The SQL Access Advisor provides three built-in task templates, using the following constants:

- `DBMS_ADVISOR.SQLACCESS_OLTP`  
Parameters are preset to favor an OLTP application environment.
- `DBMS_ADVISOR.SQLACCESS_WAREHOUSE`  
Parameters are preset to favor a data warehouse application environment.
- `DBMS_ADVISOR.SQLACCESS_GENERAL`  
Parameters are preset to favor a hybrid application environment where both OLTP and data warehouse operations may occur. For the SQL Access Advisor, this is the default template.

## Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
END;
/
```

## DELETE\_SQLWKLD Procedure

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure deletes an existing SQL Workload object from the repository.

### Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD (
    workload_name          IN VARCHAR2);
```

### Parameters

**Table 18–11** *DELETE\_SQLWKLD Procedure Parameters*

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload. The wildcard % is supported as a WORKLOAD_NAME. The rules of use are identical to the LIKE operator. For example, to delete all tasks for the current user, use the wildcard % as the WORKLOAD_NAME. If a wildcard is provided, the DELETE_SQLWKLD operation will not delete any workloads marked as READ_ONLY or TEMPLATE.

### Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

### Examples

```
DECLARE
    workload_name VARCHAR2(30);
BEGIN
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.DELETE_SQLWKLD(workload_name);
END;
/
```



## DELETE\_SQLWKLD\_REF Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure removes a link between the current SQL Access task and a SQL Workload data object.

Users should use DELETE\_STS\_REF instead of DELETE\_SQLWKLD\_REF for all SQL tuning set-based advisor runs. This function is only provided for backward compatibility.

### Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD_REF (
  task_name          IN VARCHAR2,
  workload_name      IN VARCHAR2,
  is_sts             IN NUMBER :=0);
```

### Parameters

**Table 18–12** *DELETE\_SQLWKLD\_REF Procedure Parameters*

Parameter	Description
task_name	The SQL Access task name that uniquely identifies an existing task.
workload_name	The name of the workload object to be unlinked. The wildcard % is supported as a workload_name. The rules of use are identical to the LIKE operator. For example, to remove all links to workload objects, use the wildcard % as the workload_name.
is_sts	Indicates the type of workload source. Possible values are: <ul style="list-style-type: none"> <li>■ 0 - SQL workload object</li> <li>■ 1 - SQL tuning set</li> </ul>

### Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.DELETE_SQLWKLD_REF(task_name, workload_name);
END;
/
```

## DELETE\_SQLWKLD\_STATEMENT Procedures

This procedure has been deprecated.

This procedure deletes one or more statements from a workload.

### Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (
    workload_name    IN VARCHAR2,
    sql_id           IN NUMBER);
```

```
DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (
    workload_name    IN VARCHAR2,
    search           IN VARCHAR2,
    deleted          OUT NUMBER);
```

### Parameters

**Table 18–13** *DELETE\_SQLWKLD\_STATEMENT Procedure Parameters*

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
sql_id	The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant <code>ADVISOR_ALL</code> .
search	Disabled.
deleted	Returns the number of statements deleted by the searched deleted operation.

### Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

### Examples

```
DECLARE
    workload_name VARCHAR2(30);
    deleted NUMBER;
    id NUMBER;
BEGIN
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'YEARLY', 'ROLLUP',
                                       100,400,5041,103,640445,680000,2,
                                       1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                       FROM sh.sales');

    SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
    WHERE workload_name = 'My Workload';

    DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT(workload_name, id);
END;
/
```

## DELETE\_STS\_REF Procedure

This procedure removes a link between the current SQL Access Advisor task and a SQL tuning set.

Use DELETE\_STS\_REF for any STS-based advisor runs. The older method of using DELETE\_SQLWKLD\_REF with parameter IS\_STS=1 is only supported for backward compatibility. Furthermore, the DELETE\_STS\_REF function accepts an STS owner name, whereas DELETE\_SQLWKLD\_REF does not.

### Syntax

```
DBMS_ADVISOR.DELETE_STS_REF (
    task_name      IN VARCHAR2 NOT NULL,
    sts_owner      IN VARCHAR2,
    workload_name  IN VARCHAR2 NOT NULL);
```

### Parameters

**Table 18–14** DELETE\_STS\_REF Procedure Parameters

Parameter	Description
task_name	The SQL Access Advisor task name that uniquely identifies an existing task.
sts_owner	The owner of the SQL tuning set. The value of this parameter may be NULL, in which case the advisor assumes the SQL tuning set to be owned by the currently logged-in user.
workload_name	The name of the workload to be unlinked. A workload consists of one or more SQL statements, plus statistics and attributes that fully describe each statement. The database stores a workload as a SQL tuning set.  The wildcard % is supported as a workload name. The rules of use are identical to the SQL LIKE operator. For example, to remove all links to SQL tuning set objects, use the wildcard % as the STS_NAME.

### Examples

```
DBMS_ADVISOR.DELETE_STS_REF ('My task', 'SCOTT', 'My workload');
```

## DELETE\_TASK Procedure

This procedure deletes an existing task from the repository.

### Syntax

```
DBMS_ADVISOR.DELETE_TASK (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 18–15** *DELETE\_TASK Procedure Parameters*

Parameter	Description
task_name	A single Advisor task name that will be deleted from the repository.  The wildcard % is supported as a TASK_NAME. The rules of use are identical to the LIKE operator. For example, to delete all tasks for the current user, use the wildcard % as the TASK_NAME.  If a wildcard is provided, the DELETE_TASK operation will not delete any tasks marked as READ_ONLY or TEMPLATE.

### Examples

```
DECLARE  
    task_id NUMBER;  
    task_name VARCHAR2(30);  
BEGIN  
    task_name := 'My Task';  
  
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);  
    DBMS_ADVISOR.DELETE_TASK(task_name);  
END;  
/
```

## EXECUTE\_TASK Procedure

This procedure performs the Advisor analysis or evaluation for the specified task. The procedure is overloaded.

The execution-related arguments are optional and you do not need to set them for advisors that do not allow their tasks to be executed multiple times.

Advisors can execute a task multiple times and use the results for further processing and analysis.

### Syntax

```
DBMS_ADVISOR.EXECUTE_TASK (
    task_name          IN VARCHAR2);

DBMS_ADVISOR.EXECUTE_TASK (
    task_name          IN VARCHAR2,
    execution_type     IN VARCHAR2          := NULL,
    execution_name     IN VARCHAR2          := NULL,
    execution_params   IN dbms_advisor.argList := NULL,
    execution_desc     IN VARCHAR2          := NULL,
    RETURN VARCHAR2;
```

### Parameters

**Table 18–16 EXECUTE\_TASK Procedure Parameters**

Parameter	Description
task_name	The task name that uniquely identifies an existing task.
execution_type	The type of action to be performed by the function. If NULL, it will default to the value of the <code>DEFAULT_EXECUTION_TYPE</code> parameter.  As an example, the SQL Performance Analyzer accepts the following possible values: <ul style="list-style-type: none"> <li>EXPLAIN PLAN: Generate an explain plan for a SQL statement. This is similar to an <code>EXPLAIN PLAN</code> command. The resulting plans will be stored in the advisor framework in association with the task.</li> <li>TEST EXECUTE: Test execute the SQL statement and collect its execute plan and statistics. The resulting plans and statistics are stored in the advisor framework.</li> <li>ANALYZE PERFORMANCE: Analyze and compare two versions of SQL performance data. The performance data is generated by test executing a SQL statement or generating its explain plan.</li> </ul>
execution_name	A name to qualify and identify an execution. If not specified, it will be generated by the Advisor and returned by function.
execution_params	A list of parameters (name, value) for the specified execution. Note that execution parameters are real task parameters, but they affect only the execution they are specified for.  As an example, consider the following: <code>DBMS_ADVISOR.ARGLIST('time_limit', 12, 'username', 'foo')</code>
execution_desc	A 256-length string describing the execution.

## Usage Notes

Task execution is a synchronous operation. Control will not be returned to the caller until the operation has completed, or a user-interrupt was detected.

Upon return, you can check the DBA\_ADVISOR\_LOG table for the execution status.

## Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.ADD_SQLWKL_REF(task_name, workload_name);
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/
```

## GET\_REC\_ATTRIBUTES Procedure

This procedure retrieves a specified attribute of a new object as recommended by Advisor analysis.

### Syntax

```
DBMS_ADVISOR.GET_REC_ATTRIBUTES (
  workload_name      IN VARCHAR2,
  rec_id             IN NUMBER,
  action_id          IN NUMBER,
  attribute_name     IN VARCHAR2,
  value              OUT VARCHAR2,
  owner_name         IN VARCHAR2 := NULL);
```

### Parameters

**Table 18–17** *GET\_REC\_ATTRIBUTES Procedure Parameters*

Parameter	Description
task_name	The task name that uniquely identifies an existing task.
rec_id	The Advisor-generated identifier number that is assigned to the recommendation.
action_id	The Advisor-generated action identifier that is assigned to the particular command.
attribute_name	Specifies the attribute to change.
value	The buffer to receive the requested attribute value.
owner_name	Optional owner name of the target task. This permits access to task data not owned by the current user.

### Return Values

The requested attribute value is returned in the VALUE argument.

### Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
  DBMS_ADVISOR.GET_REC_ATTRIBUTES(task_name, 1, 1, 'NAME', attribute);
END;
/
```

## GET\_TASK\_REPORT Function

This function creates and returns a report for the specified task.

### Syntax

```
DBMS_ADVISOR.GET_TASK_REPORT (  
    task_name      IN VARCHAR2,  
    type           IN VARCHAR2 := 'TEXT',  
    level          IN VARCHAR2 := 'TYPICAL',  
    section        IN VARCHAR2 := 'ALL',  
    owner_name     IN VARCHAR2 := NULL,  
    execution_name IN VARCHAR2 := NULL,  
    object_id      IN NUMBER   := NULL)  
RETURN CLOB;
```

### Parameters

**Table 18–18** GET\_TASK\_REPORT Function Parameters

Parameter	Description
task_name	The name of the task from which the script will be created.
type	The only valid value is TEXT.
level	The possible values are BASIC, TYPICAL, and ALL.
section	Advisor-specific report sections.
owner_name	Owner of the task. If specified, the system will check to see if the current user has read privileges to the task data.
execution_name	An identifier of a specific execution of the task. It is needed only for advisors that allow their tasks to be executed multiple times.
object_id	An identifier of an advisor object that can be targeted by the script.

### Return Values

Returns the buffer receiving the script.



## GET\_TASK\_SCRIPT Function

This function creates a SQL\*Plus-compatible SQL script and sends the output to file. The script will contain all of the accepted recommendations from the specified task.

### Syntax

```
DBMS_ADVISOR.GET_TASK_SCRIPT (
  task_name          IN VARCHAR2
  type               IN VARCHAR2 := 'IMPLEMENTATION',
  rec_id             IN NUMBER   := NULL,
  act_id             IN NUMBER   := NULL,
  owner_name         IN VARCHAR2 := NULL,
  execution_name     IN VARCHAR2 := NULL,
  object_id          IN NUMBER   := NULL)
RETURN CLOB;
```

### Parameters

**Table 18–19** GET\_TASK\_SCRIPT Function Parameters

Parameter	Description
task_name	The task name that uniquely identifies an existing task.
type	Specifies the type of script to generate. The possible values are IMPLEMENTATION and UNDO.
rec_id	An optional recommendation identifier number that can be used to extract a subset of the implementation script.  A zero or the value DBMS_ADVISOR.ADVISOR_ALL indicates all accepted recommendations would be included. The default is to include all accepted recommendations for the task.
act_id	Optional action identifier number that can be used to extract a single action as a DDL command.  A zero or the value DBMS_ADVISOR.ADVISOR_ALL indicates all actions for the recommendation would be included. The default is to include all actions for a recommendation.
owner_name	An optional task owner name.
execution_name	An identifier of a specific execution of the task. It is needed only for advisors that allow their tasks to be executed multiple times.
object_id	An identifier of an advisor object that can be targeted by the script.

### Return Values

Returns the script as a CLOB buffer.

### Usage Notes

Though the script is ready to execute, Oracle recommends that the user review the script for acceptable locations for new materialized views and indexes.

For a recommendation to appear in a generated script, it must be marked as accepted.

### Examples

```
DECLARE
  task_id NUMBER;
```

```
task_name VARCHAR2(30);
workload_name VARCHAR2(30);
buf CLOB;
BEGIN
task_name := 'My Task';
workload_name := 'My Workload';

DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
100,400,5041,103,640445,680000,2,
1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
FROM sh.sales');
DBMS_ADVISOR.EXECUTE_TASK(task_name);
buf := DBMS_ADVISOR.GET_TASK_SCRIPT(task_name);
END;
/
```

## IMPLEMENT\_TASK Procedure

This procedure implements the recommendations of the specified task.

### Syntax

```
DBMS_ADVISOR.IMPLEMENT_TASK (  
  task_name          IN VARCHAR2,  
  rec_id             IN NUMBER := NULL,  
  exit_on_error      IN BOOLEAN := NULL);
```

### Parameters

**Table 18–20** *IMPLEMENT\_TASK Procedure Parameters*

Parameter	Description
task_name	The name of the task.
rec_id	An optional recommendation ID.
exit_on_error	An optional boolean to exit on the first error.

## IMPORT\_SQLWKLD\_SCHEMA Procedure

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure constructs and loads a SQL workload based on schema evidence. The workload is also referred to as a hypothetical workload.

### Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SCHEMA (
  workload_name      IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW',
  priority           IN NUMBER := 2,
  saved_rows         OUT NUMBER,
  failed_rows        OUT NUMBER);
```

### Parameters

**Table 18–21** *IMPORT\_SQLWKLD\_SCHEMA Procedure Parameters*

Parameter	Description
<code>workload_name</code>	The workload object name that uniquely identifies an existing workload.
<code>import_mode</code>	Specifies the action to be taken when storing the workload. Possible values are: <ul style="list-style-type: none"> <li>▪ <code>APPEND</code> Indicates that the collected workload will be added to any existing workload in the task.</li> <li>▪ <code>NEW</code> Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</li> <li>▪ <code>REPLACE</code> Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</li> </ul> The default value is <code>NEW</code> .
<code>priority</code>	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.
<code>failed_rows</code>	Returns the number of rows that were not saved due to syntax or validation errors.
<code>saved_rows</code>	Returns the number of rows actually saved in the repository.

### Return Values

This call returns the number of rows saved and failed as output parameters.

### Usage Notes

To successfully import a hypothetical workload, the target schemas must contain dimensions.

If the `VALID_TABLE_LIST` parameter is not set, the search space may become very large and require a significant amount of time to complete. Oracle recommends that you limit your search space to specific set of tables.

If a task contains valid recommendations from a prior run, adding or modifying task will mark the task as invalid, preventing the viewing and reporting of potentially valuable recommendation data.

## Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKL(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKL_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKL_SCHEMA(workload_name, 'REPLACE', 1, saved,
    failed);
END;
/
```

## IMPORT\_SQLWKLD\_SQLCACHE Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure creates a SQL workload from the current contents of the server's SQL cache.

### Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE (
  workload_name      IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW',
  priority           IN NUMBER := 2,
  saved_rows         OUT NUMBER,
  failed_rows        OUT NUMBER);
```

### Parameters

**Table 18–22** *IMPORT\_SQLWKLD\_SQLCACHE Procedure Parameters*

Parameter	Description
<code>workload_name</code>	The workload object name that uniquely identifies an existing workload.
<code>import_mode</code>	Specifies the action to be taken when storing the workload. Possible values are: <ul style="list-style-type: none"> <li>▪ <code>APPEND</code> Indicates that the collected workload will be added to any existing workload in the task.</li> <li>▪ <code>NEW</code> Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</li> <li>▪ <code>REPLACE</code> Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</li> </ul> The default value is <code>NEW</code> .
<code>priority</code>	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following 1-HIGH, 2-MEDIUM, or 3-LOW.
<code>saved_rows</code>	Returns the number of rows saved as output parameters.
<code>failed_rows</code>	Returns the number of rows that were not saved due to syntax or validation errors.

### Return Values

This call returns the number of rows saved and failed as output parameters.

### Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

### Examples

```
DECLARE
```

```
workload_name VARCHAR2(30);
saved NUMBER;
failed NUMBER;
BEGIN
workload_name := 'My Workload';

DBMS_ADVISOR.CREATE_SQLWKL(workload_name, 'My Workload');
DBMS_ADVISOR.SET_SQLWKL_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
DBMS_ADVISOR.IMPORT_SQLWKL_SQLCACHE(workload_name, 'REPLACE', 1, saved,
failed);
END;
/
```

## IMPORT\_SQLWKLD\_STS Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure loads a SQL workload from an existing SQL tuning set. A SQL tuning set is typically created from the server workload repository using various time and data filters.

### Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
  workload_name      IN VARCHAR2,
  sts_name           IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW',
  priority           IN NUMBER := 2,
  saved_rows         OUT NUMBER,
  failed_rows        OUT NUMBER);
```

```
DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
  workload_name      IN VARCHAR2,
  sts_owner          IN VARCHAR2,
  sts_name           IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW',
  priority           IN NUMBER := 2,
  saved_rows         OUT NUMBER,
  failed_rows        OUT NUMBER);
```

### Parameters

**Table 18–23** *IMPORT\_SQLWKLD\_STS Procedure Parameters*

Parameter	Description
<code>workload_name</code>	The workload object name that uniquely identifies an existing workload.
<code>sts_owner</code>	The optional owner of the SQL tuning set.
<code>sts_name</code>	The name of an existing SQL tuning set workload from which the data will be imported. If the <code>sts_owner</code> value is not provided, the owner will default to the current user.
<code>import_mode</code>	Specifies the action to be taken when storing the workload. Possible values are: <ul style="list-style-type: none"> <li>▪ <code>APPEND</code> Indicates that the collected workload will be added to any existing workload in the task.</li> <li>▪ <code>NEW</code> Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</li> <li>▪ <code>REPLACE</code> Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</li> </ul> <p>The default value is <code>NEW</code>.</p>
<code>priority</code>	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW. The default value is 2.
<code>saved_rows</code>	Returns the number of rows actually saved in the repository.



**Table 18–23 (Cont.) IMPORT\_SQLWKLD\_STS Procedure Parameters**

Parameter	Description
failed_rows	Returns the number of rows that were not saved due to syntax or validation errors.

## Return Values

This call returns the number of rows saved and failed as output parameters.

## Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

## Examples

```

DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_STS(workload_name, 'MY_SQLSET', 'REPLACE', 1,
    saved, failed);
END;
/

```

## IMPORT\_SQLWKLD\_SUMADV Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure collects a SQL workload from a Summary Advisor workload. This procedure is intended to assist Oracle9i Database Summary Advisor users in the migration to SQL Access Advisor.

### Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV (
  workload_name      IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW',
  priority           IN NUMBER := 2,
  sumadv_id          IN NUMBER,
  saved_rows         OUT NUMBER,
  failed_rows        OUT NUMBER);
```

### Parameters

**Table 18–24** *IMPORT\_SQLWKLD\_SUMADV Procedure Parameters*

Parameter	Description
<code>workload_name</code>	The workload object name that uniquely identifies an existing workload.
<code>import_mode</code>	Specifies the action to be taken when storing the workload. Possible values are: <ul style="list-style-type: none"> <li>▪ <code>APPEND</code> Indicates that the collected workload will be added to any existing workload in the task.</li> <li>▪ <code>NEW</code> Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</li> <li>▪ <code>REPLACE</code> Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</li> </ul> The default value is <code>NEW</code> .
<code>priority</code>	Specifies the default application priority for each statement that is saved in the workload object. If a Summary Advisor workload statement contains a priority of zero, the default priority will be applied. If the workload statement contains a valid priority, then the Summary Advisor priority will be converted to a comparable SQL Access Advisor priority. The value must be one of the following: 1-HIGH, 2-MEDIUM, or 3-LOW.
<code>sumadv_id</code>	Specifies the Summary Advisor workload identifier number.
<code>saved_rows</code>	Returns the number of rows actually saved in the repository.
<code>failed_rows</code>	Returns the number of rows that were not saved due to syntax or validation errors.

### Return Values

This call returns the number of rows saved and failed as output parameters.

## Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

## Examples

```
DECLARE
    workload_name VARCHAR2(30);
    saved NUMBER;
    failed NUMBER;
    sumadv_id NUMBER;
BEGIN
    workload_name := 'My Workload';
    sumadv_id := 394;

    DBMS_ADVISOR.CREATE_SQLWKL(workload_name, 'My Workload');
    DBMS_ADVISOR.SET_SQLWKL_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
    DBMS_ADVISOR.IMPORT_SQLWKL_SUMADV(workload_name, 'REPLACE', 1, sumadv_id,
        saved, failed);
END;
/
```

## IMPORT\_SQLWKLD\_USER Procedure

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure collects a SQL workload from a specified user table.

### Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_USER (
  workload_name      IN VARCHAR2,
  import_mode        IN VARCHAR2 := 'NEW',
  owner_name         IN VARCHAR2,
  table_name         IN VARCHAR2,
  saved_rows         OUT NUMBER,
  failed_rows        OUT NUMBER);
```

### Parameters

**Table 18–25** *IMPORT\_SQLWKLD\_USER Procedure Parameters*

Parameter	Description
<code>workload_name</code>	The workload object name that uniquely identifies an existing workload.
<code>import_mode</code>	Specifies the action to be taken when storing the workload. Possible values are: <ul style="list-style-type: none"> <li>▪ <code>APPEND</code> Indicates that the collected workload will be added to any existing workload in the task.</li> <li>▪ <code>NEW</code> Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown.</li> <li>▪ <code>REPLACE</code> Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload.</li> </ul> The default value is <code>NEW</code> .
<code>owner_name</code>	Specifies the owner name of the table or view from which workload data will be collected.
<code>table_name</code>	Specifies the name of the table or view from which workload data will be collected.
<code>saved_rows</code>	Returns the number of rows actually saved in the workload object.
<code>failed_rows</code>	Returns the number of rows that were not saved due to syntax or validation errors.

### Return Values

This call returns the number of rows saved and failed as output parameters.

### Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

## Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKL(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKL_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKL_USER(workload_name, 'REPLACE', 'SH',
    'USER_WORKLOAD', saved, failed);
END;
/
```

## INTERRUPT\_TASK Procedure

This procedure stops a currently executing task. The task will end its operations as it would at a normal exit. The user will be able to access any recommendations that exist to this point.

### Syntax

```
DBMS_ADVISOR.INTERRUPT_TASK (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 18–26** *INTERRUPT\_TASK Procedure Parameters*

Parameter	Description
task_name	A single Advisor task name that will be interrupted.

### Examples

```
DECLARE  
    task_id NUMBER;  
    task_name VARCHAR2(30);  
BEGIN  
    task_name := 'My Task';  
  
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);  
    DBMS_ADVISOR.EXECUTE_TASK(task_name);  
END;  
/
```

While this session is executing its task, you can interrupt the task from a second session using the following statement:

```
BEGIN  
    DBMS_ADVISOR.INTERRUPT_TASK('My Task');  
END;  
/
```

## MARK\_RECOMMENDATION Procedure

This procedure marks a recommendation for import or implementation.

### Syntax

```
DBMS_ADVISOR.MARK_RECOMMENDATION (
  task_name      IN VARCHAR2
  id             IN NUMBER,
  action        IN VARCHAR2);
```

### Parameters

**Table 18–27** *MARK\_RECOMMENDATION Procedure Parameters*

Parameter	Description
task_name	Name of the task.
id	The recommendation identifier number assigned by the Advisor.
action	The recommendation action setting. The possible actions are: <ul style="list-style-type: none"> <li>▪ ACCEPT Marks the recommendation as accepted. With this setting, the recommendation will appear in implementation and undo scripts.</li> <li>▪ IGNORE Marks the recommendation as ignore. With this setting, the recommendation will not appear in an implementation or undo script.</li> <li>▪ REJECT Marks the recommendation as rejected. With this setting, the recommendation will not appear in any implementation or undo scripts.</li> </ul>

### Usage Notes

For a recommendation to be implemented, it must be marked as accepted. By default, all recommendations are considered accepted and will appear in any generated scripts.

### Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
  rec_id NUMBER;
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

  rec_id := 1;
  DBMS_ADVISOR.MARK_RECOMMENDATION(task_name, rec_id, 'REJECT');
END;
```

## QUICK\_TUNE Procedure

This procedure performs an analysis and generates recommendations for a single SQL statement.

This provides a shortcut method of all necessary operations to analyze the specified SQL statement. The operation creates a task using the specified task name. The task will be created using a specified Advisor task template. Finally, the task will be executed and the results will be saved in the repository.

### Syntax

```
DBMS_ADVISOR.QUICK_TUNE (
    advisor_name      IN VARCHAR2,
    task_name         IN VARCHAR2,
    attr1             IN CLOB,
    attr2             IN VARCHAR2 := NULL,
    attr3             IN NUMBER := NULL,
    task_or_template  IN VARCHAR2 := NULL);
```

### Parameters

**Table 18–28 QUICK\_TUNE Procedure Parameters**

Parameter	Description
advisor_name	Name of the Advisor that will perform the analysis.
task_name	Name of the task.
attr1	Advisor-specific attribute in the form of a CLOB variable.
attr2	Advisor-specific attribute in the form of a VARCHAR2 variable.
attr3	Advisor-specific attribute in the form of a NUMBER.
task_or_template	An optional task name of an existing task or task template.

### Usage Notes

If indicated by the user, the final recommendations can be implemented by the procedure.

The task will be created using either a specified SQL Access task template or the built-in default template of `SQLACCESS_GENERAL`. The workload will only contain the specified statement, and all task parameters will be defaulted.

`attr1` must be the single SQL statement to tune. For the SQL Access Advisor, `attr2` is the user who would execute the single statement. If omitted, the current user will be used.

### Examples

```
DECLARE
    task_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';

    DBMS_ADVISOR.QUICK_TUNE(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_name,
        'SELECT AVG(amount_sold) FROM sh.sales WHERE promo_id=10');
END;
/
```



## RESET\_SQLWKLD Procedure

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure resets a workload to its initial starting point. This has the effect of removing all journal messages, log messages, and recalculating necessary volatility and usage statistics.

### Syntax

```
DBMS_ADVISOR.RESET_SQLWKLD (
    workload_name          IN VARCHAR2);
```

### Parameters

**Table 18–29** *RESET\_SQLWKLD Procedure Parameters*

Parameter	Description
workload_name	The SQL Workload object name that uniquely identifies an existing workload.

### Usage Notes

RESET\_SQLWKLD should be executed after any workload adjustments such as adding or removing SQL statements.

### Examples

```
DECLARE
    workload_name VARCHAR2(30);
BEGIN
    workload_name := 'My Workload';

    DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
    DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                        100,400,5041,103,640445,680000,2,
                                        1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                        FROM sh.sales WHERE promo_id = 10');

    DBMS_ADVISOR.RESET_SQLWKLD(workload_name);
END;
/
```

## RESET\_TASK Procedure

This procedure resets a task to its initial state. All intermediate and recommendation data will be removed from the task. The task status will be set to `INITIAL`.

### Syntax

```
DBMS_ADVISOR.RESET_TASK (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 18–30** *RESET\_TASK Procedure Parameters*

Parameter	Description
task_name	The task name that uniquely identifies an existing task.

### Examples

```
DECLARE  
    task_id NUMBER;  
    task_name VARCHAR2(30);  
    workload_name VARCHAR2(30);  
BEGIN  
    task_name := 'My Task';  
    workload_name := 'My Workload';  
  
    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);  
    DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);  
    DBMS_ADVISOR.EXECUTE_TASK(task_name);  
    DBMS_ADVISOR.RESET_TASK(task_name);  
END;  
/
```

## SET\_DEFAULT\_SQLWKLD\_PARAMETER Procedures

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure modifies the default value for a user parameter within a SQL Workload object or SQL Workload object template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting. When a default value is changed for a parameter, workload objects will inherit the new value when they are created.

### Syntax

```
DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (
  parameter          IN VARCHAR2,
  value              IN VARCHAR2);
```

```
DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (
  parameter          IN VARCHAR2,
  value              IN NUMBER);
```

### Parameters

**Table 18–31** SET\_DEFAULT\_SQLWKLD\_PARAMETER Procedure Parameters

Parameter	Description
parameter	The name of the data parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the workload object type, but not necessarily unique to all workload object types. Various object types may use the same parameter name for different purposes.
value	The value of the specified parameter. The value can be specified as a string or a number. If the value is DBMS_ADVISOR.DEFAULT, the value will be reset to the default value.

### Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

### Examples

```
BEGIN
  DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER('VALID_TABLE_LIST', 'SH.%');
END;
/
```

## SET\_DEFAULT\_TASK\_PARAMETER Procedures

This procedure modifies the default value for a user parameter within a task or a template. A user parameter is a simple variable that stores various attributes that affect various Advisor operations. When a default value is changed for a parameter, tasks will inherit the new value when they are created.

A default task is different from a regular task. The default value is the initial value that will be inserted into a newly created task, while setting a task parameter with SET\_TASK\_PARAMETER sets the local value only. Thus, SET\_DEFAULT\_TASK\_PARAMETER has no effect on an existing task.

### Syntax

```
DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (
    advisor_name      IN VARCHAR2
    parameter         IN VARCHAR2,
    value             IN VARCHAR2);
```

```
DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (
    advisor_name      IN VARCHAR2
    parameter         IN VARCHAR2,
    value             IN NUMBER);
```

### Parameters

**Table 18–32** SET\_DEFAULT\_TASK\_PARAMETER Procedure Parameters

Parameter	Description
advisor_name	Specifies the unique advisor name as defined in the view DBA_ADVISOR_DEFINITIONS.
parameter	The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.
value	The value of the specified task parameter. The value can be specified as a string or a number.

### Examples

```
BEGIN
    DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(DBMS_ADVISOR.SQLACCESS_ADVISOR,
        'VALID_TABLE_LIST', 'SH.%');
END;
/
```

## SET\_SQLWKLD\_PARAMETER Procedures

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure modifies a user parameter within a SQL Workload object or SQL Workload object template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

### Syntax

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (
  workload_name      IN VARCHAR2,
  parameter          IN VARCHAR2,
  value              IN VARCHAR2);
```

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (
  workload_name      IN VARCHAR2,
  parameter          IN VARCHAR2,
  value              IN NUMBER);
```

### Parameters

**Table 18–33** SET\_SQLWKLD\_PARAMETER Procedure Parameters

Parameter	Description
workload_name	The SQL Workload object name that uniquely identifies an existing workload.
parameter	The name of the data parameter to be modified. Parameter names are not case sensitive.
value	The value of the specified parameter. The value can be specified as a string or a number. If the value is DBMS_ADVISOR.DEFAULT, the value will be reset to the default value.

### Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

### Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name, 'VALID_TABLE_LIST', 'SH.%');
END;
/
```

## SET\_TASK\_PARAMETER Procedure

This procedure modifies a user parameter within an Advisor task or a template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

### Syntax

```
DBMS_ADVISOR.SET_TASK_PARAMETER (
  task_name          IN VARCHAR2
  parameter          IN VARCHAR2,
  value              IN VARCHAR2);
```

```
DBMS_ADVISOR.SET_TASK_PARAMETER (
  task_name          IN VARCHAR2
  parameter          IN VARCHAR2,
  value              IN NUMBER);
```

### Parameters

**Table 18–34** SET\_TASK\_PARAMETER Procedure Parameters

Parameter	Description
task_name	The Advisor task name that uniquely identifies an existing task.
parameter	The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.
value	The value of the specified task parameter. The value can be specified as a string or a number. If the value is <code>DEFAULT</code> , the value will be reset to the default value.

### Usage Notes

A task cannot be modified unless it is in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state. See your Advisor-specific documentation for further information on using this procedure.

### SQL Access Advisor Task Parameters

[Table 18–35](#) lists SQL Access Advisor task parameters.

**Table 18–35 SQL Access Advisor Task Parameters**

Parameter	Description
ANALYSIS_SCOP E	<p>A comma-separated list that specifies the tuning artifacts to consider during analysis.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ ALL Short name for specifying INDEX, MVIEW, TABLE, and PARTITION.</li> <li>▪ EVALUATION Causes a read-only evaluation of the specified workload. No new recommendations will be made. Can only be specified alone.</li> <li>▪ INDEX Allows the SQL Access Advisor to recommend index structure changes.</li> <li>▪ MVIEW Allows the SQL Access Advisor to recommend materialized view and log changes.</li> <li>▪ PARTITION Allows the SQL Access Advisor to recommend partition options. Use this in conjunction with the INDEX, MVIEW, and TABLE options.</li> <li>▪ TABLE Allows the SQL Access Advisor to make base-table recommendations. In this release, the only base-table recommendation is partitioning.</li> </ul> <p>Using the new keywords, the following combinations are valid:</p> <ul style="list-style-type: none"> <li>▪ INDEX</li> <li>▪ MVIEW</li> <li>▪ INDEX, PARTITION</li> <li>▪ INDEX, MVIEW, PARTITION</li> <li>▪ INDEX, TABLE, PARTITION</li> <li>▪ MVIEW, PARTITION</li> <li>▪ MVIEW, TABLE, PARTITION</li> <li>▪ INDEX, MVIEW, TABLE, PARTITION</li> <li>▪ TABLE, PARTITION</li> <li>▪ EVALUATION</li> </ul> <p>The default value is INDEX. The datatype is STRINGLIST.</p>
CREATION_COST	<p>When set to <code>true</code> (default), the SQL Access Advisor will weigh the cost of creation of the access structure (index or materialized view) against the frequency of the query and potential improvement in the query execution time. When set to <code>false</code>, the cost of creation is ignored. The datatype is STRING.</p>
DAYS_TO_EXPIR E	<p>Specifies the expiration time in days for the current SQL Access Advisor task. The value is relative to the last modification date. Once the task expires, it will become a candidate for removal by an automatic purge operation.</p> <p>Specifies the expiration time in days for the current Access Advisor task. The value is relative to the last modification date. The datatype is NUMBER.</p> <p>Once the task expires, it becomes a candidate for removal by an automatic purge operation.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ an integer in the range of 0 to 2147483647</li> <li>▪ ADVISOR_UNLIMITED</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is 30.</p>
DEF_EM_TEMPLA TE	<p>Contains the default task or template name from which the Enterprise Manager SQL Access Advisor Wizard reads its initial values.</p> <p>The default value is SQLACCESS_EMTASK. The datatype is STRING.</p>
DEF_INDEX_OWNE R	<p>Specifies the default owner for new index recommendations. When a script is created, this value will be used to qualify the index name.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>▪ Existing schema name. Quoted identifiers are supported.</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRING.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
DEF_INDEX_TABLESPACE	<p>Specifies the default tablespace for new index recommendations. When a script is created, this value will be used to specify a tablespace clause.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>Existing tablespace name. Quoted identifiers are supported.</li> <li>ADVISOR_UNUSED No tablespace clause will be present in the script for indexes.</li> </ul> <p>The default value is <code>ADVISOR_UNUSED</code>. The datatype is <code>STRING</code>.</p>
DEF_MVIEW_OWNER	<p>Specifies the default owner for new materialized view recommendations. When a script is created, this value will be used to qualify the materialized view name.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>Existing schema name. Quoted identifiers are supported.</li> <li>ADVISOR_UNUSED</li> </ul> <p>The default value is <code>ADVISOR_UNUSED</code>. The datatype is <code>STRING</code>.</p>
DEF_MVIEW_TABLESPACE	<p>Specifies the default tablespace for new materialized view recommendations. When a script is created, this value will be used to specify a tablespace clause.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>Existing tablespace name. Quoted identifiers are supported.</li> <li>ADVISOR_UNUSED. No tablespace clause will be present in the script for materialized view logs.</li> </ul> <p>The default value is <code>ADVISOR_UNUSED</code>. The datatype is <code>STRING</code>.</p>
DEF_MVLOG_TABLESPACE	<p>Specifies the default tablespace for new materialized view log recommendations. When a script is created, this value will be used to specify a tablespace clause.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>Existing tablespace name. Quoted identifiers are supported.</li> <li>ADVISOR_UNUSED. No tablespace clause will be present in the script for materialized view logs.</li> </ul> <p>The default value is <code>ADVISOR_UNUSED</code>. The datatype is <code>STRING</code>.</p>
DEF_PARTITION_TABLESPACE	<p>Specifies the default tablespace for new partitioning recommendations. When a script is created, this value will be used to specify a tablespace clause.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>Existing tablespace name. Quoted identifiers are supported.</li> <li>ADVISOR_UNUSED. No tablespace clause will be present in the script for materialized views.</li> </ul> <p>The default value is <code>ADVISOR_UNUSED</code>. The datatype is <code>STRING</code>.</p>
DML_VOLATILITY	<p>When set to <code>TRUE</code>, the SQL Access Advisor will consider the impact of index maintenance and materialized view refresh in determining the recommendations. It will limit the access structure recommendations involving columns or tables that are frequently updated. For example, if there are too many DMLs on a column, then it may favor a B-tree index over a bitmap index on that column. For this process to be effective, the workload must include DML (insert/update/delete/merge/direct path inserts) statements that represent the update behavior of the application. The datatype is <code>STRING</code>.</p> <p>See the related parameter <code>refresh_mode</code>.</p>
END_TIME	<p>Specifies an end time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed.</p> <p>Each date must be in the standard Oracle form of <code>MM-DD-YYYY HH24:MI:SS</code>, where:</p> <ul style="list-style-type: none"> <li>DD is the numeric date</li> <li>MM is the numeric month</li> <li>YYYY is the numeric year</li> <li>HH is the hour in 24 hour format</li> <li>MI is the minute</li> <li>SS is the second</li> </ul> <p>The datatype is <code>STRING</code>.</p>



**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
EVALUATION_ON LY	<p>This parameter is maintained for backward compatibility. All values will be translated and placed into the ANALYSIS_SCOPE task parameter.</p> <p>If set to TRUE, causes SQL Access Advisor to analyze the workload, but only comment on how well the current configuration is supporting it. No tuning recommendations will be generated.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>▪ FALSE</li> <li>▪ TRUE</li> </ul> <p>The default value is FALSE. The datatype is STRING.</p>
EXECUTION_TYP E	<p>This parameter is maintained for backward compatibility. All values will be translated and placed into the ANALYSIS_SCOPE task parameter.</p> <p>The translated values are:</p> <ul style="list-style-type: none"> <li>▪ FULL =&gt; FULL</li> <li>▪ INDEX_ONLY =&gt; INDEX</li> <li>▪ MVIEW_ONLY =&gt; MVIEW</li> <li>▪ MVIEW_LOG_ONLY =&gt; MVIEW_LOG_ONLY</li> </ul> <p>The type of recommendations that is desired. Possible values:</p> <ul style="list-style-type: none"> <li>▪ FULL All supported recommendation types will be considered.</li> <li>▪ INDEX_ONLY The SQL Access Advisor will only consider index solutions as recommendations.</li> <li>▪ MVIEW_ONLY The SQL Access Advisor will consider materialized view and materialized view log solutions as recommendations.</li> <li>▪ MVIEW_LOG_ONLY The SQL Access Advisor will only consider materialized view log solutions as recommendations.</li> </ul> <p>The default value is FULL. The datatype is STRINGLIST.</p>
IMPLEMENT_EXI T_ON_ERROR	<p>When performing an IMPLEMENT_TASK operation, this parameter will control behavior when an action fails to implement. If set to TRUE, IMPLEMENT_TASK will stop on the first unexpected error.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ TRUE</li> <li>▪ FALSE</li> </ul> <p>The default value is TRUE. The datatype is STRING.</p>
INDEX_NAME_TE MPLATE	<p>Specifies the method by which new index names are formed.</p> <p>If the TASK_ID is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the TASK_ID in the template. Once formatted, the maximum size of a name is 30 characters.</p> <p>Valid keywords are:</p> <ul style="list-style-type: none"> <li>▪ Any literal value up to 22 characters.</li> <li>▪ TABLE Causes the parent table name to be substituted into the index name. If the name is too long, it will be trimmed to fit.</li> <li>▪ TASK_ID Causes the current task identifier number to be inserted in hexadecimal form.</li> <li>▪ SEQ Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token.</li> </ul> <p>The default template is &lt;TABLE&gt;_IDX\$\$_&lt;TASK_ID&gt;&lt;SEQ&gt;. The datatype is STRING.</p>
INVALID_ACTIO N_LIST	<p>Contains a fully qualified list of actions that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness.</p> <p>During a task execution, if a SQL statement's action matches a name in the action list, it will not be processed by the task. An action name is case sensitive.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single action</li> <li>▪ comma-delimited action list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
INVALID_MODULE_LIST	<p>Contains a fully qualified list of modules that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness.</p> <p>During a task execution, if a SQL statement's module matches a name in the list, it will not be processed by the task. A module name is case sensitive.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single application</li> <li>▪ comma-delimited module list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>
INVALID_SQLSTRING_LIST	<p>Contains a fully qualified list of text strings that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted values are supported.</p> <p>A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness.</p> <p>During a task execution, if a SQL statement contains a string in the SQL string list, it will not be processed by the task.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single string</li> <li>▪ comma-delimited string list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>
INVALID_USERNAME_LIST	<p>Contains a fully qualified list of usernames that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>During a task execution, if a SQL statement's username matches a name in the username list, it will not be processed by the task. A username is not case sensitive unless it is quoted.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single username</li> <li>▪ comma-delimited username list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>
JOURNALING	<p>Controls the logging of messages to the journal (DBA_ADVISOR_JOURNAL and USER_ADVISOR_JOURNAL views). The higher the setting, the more information is logged to the journal.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>▪ UNUSED: no journal messages</li> <li>▪ FATAL: explanation of fatal conditions</li> <li>▪ ERROR: explanation of errors</li> <li>▪ WARNING: explanation of warnings</li> <li>▪ INFORMATION: information message</li> <li>▪ INFORMATION2: common information</li> <li>▪ INFORMATION3: common information</li> <li>▪ INFORMATION4: common information</li> <li>▪ INFORMATION5: common information</li> <li>▪ INFORMATION6: common information</li> </ul> <p>Each journal value represents all recorded messages at that level or lower. For example, when choosing WARNING, all messages marked WARNING as well as ERROR and FATAL will be recorded in the repository.</p> <p>INFORMATION6 represents the most thorough message recording and UNUSED is the least.</p> <p>The default value is INFORMATION. The datatype is NUMBER.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
LIMITED_PARTITION_SCHEMES	<p>User can suggest that the Partition Expert cut off the number of partitioning schemes to investigate. This can help with cutting down the run time of the advisor.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>■ An integer in the range of 1 to 10</li> <li>■ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is NUMBER.</p>
MAX_NUMBER_PARTITIONS	<p>Limits the number of partitions the advisor will recommend for any base table, index, or materialized view.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>■ An integer in the range of 1 to 4294967295</li> <li>■ ADVISOR_UNLIMITED</li> <li>■ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNLIMITED. The datatype is NUMBER.</p>
MODE	<p>Specifies the mode by which Access Advisor will operate during an analysis.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ LIMITED Indicates the Advisor will attempt to a quick job by limiting the search-space of candidate recommendations, and correspondingly, the results may be of a low quality.</li> <li>■ COMPREHENSIVE Indicates the Advisor will search a large pool of candidates that may take long to run, but the resulting recommendations will be of the highest quality.</li> </ul> <p>The default value is COMPREHENSIVE. The datatype is STRING.</p>
MVIEW_NAME_TEMPLATE	<p>Specifies the method by which new materialized view names are formed.</p> <p>If the <i>TASK_ID</i> is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the <i>TASK_ID</i> in the template.</p> <p>The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters.</p> <p>Valid tokens are:</p> <ul style="list-style-type: none"> <li>■ Any literal value up to 22 characters.</li> <li>■ <i>TASK_ID</i> Causes the current task identifier number to be inserted in hexadecimal form.</li> <li>■ <i>SEQ</i> Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token.</li> </ul> <p>The default template is: <code>MV\$\$_&lt;TASK_ID&gt;&lt;SEQ&gt;</code>. The datatype is STRING.</p>
ORDER_LIST	<p>This parameter has been deprecated.</p> <p>Contains the primary natural order in which the Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, Access Advisor sorts the workload using <i>ORDER_LIST</i> values. A comma must separate multiple order keys.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>■ BUFFER_GETS Sets the order using the SQL statement's buffer-get count value.</li> <li>■ CPU_TIME Sets the order using the SQL statement's CPU time value.</li> <li>■ DISK_READS Sets the order using the SQL statement's disk-read count value.</li> <li>■ ELAPSED_TIME Sets the order using the SQL statement's elapsed time value.</li> <li>■ EXECUTIONS Sets the order using the SQL statement's execution frequency value.</li> <li>■ OPTIMIZER_COST Sets the order using the SQL statement's optimizer cost value.</li> <li>■ I/O Sets the order using the SQL statement's I/O count value.</li> <li>■ PRIORITY Sets the order using the user-supplied business priority value.</li> </ul> <p>All values are accessed in descending order, where a high value is considered more interesting than a low value.</p> <p>The default value is PRIORITY, OPTIMIZER_COST. The datatype is STRINGLIST.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
PARTITION_NAME_TEMPLATE	<p>Specifies the method by which new partition names are formed. The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters.</p> <p>Valid tokens are:</p> <ul style="list-style-type: none"> <li>Any literal value up to 22 characters.</li> <li>&lt;TABLE&gt; Causes the parent table name to be substituted into the partition name. If the name is too long, it will be trimmed to fit.</li> <li>&lt;TASK_ID&gt; Causes the current task identifier number to be inserted in hexadecimal form.</li> <li>&lt;SEQ&gt; Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token.</li> </ul> <p>The default template is PTN\$\$&lt;TABLE&gt;_&lt;TASK_ID&gt;&lt;SEQ&gt;. The datatype is <i>STRING</i>.</p>
PARTITIONING_GOAL	<p>Specifies the approach used to make partitioning recommendations. One possible value is <i>PERFORMANCE</i>, which is the default. The datatype is <i>STRING</i>.</p>
PARTITIONING_TYPES	<p>Specifies the type of partitioning used. Possible values are <i>RANGE</i> and <i>HASH</i>. The datatype is <i>STRING</i>.</p>
RANKING_MEASURE	<p>Contains the primary natural order in which the SQL Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, SQL Access Advisor sorts the workload using <i>RANKING_MEASURE</i> values. A comma must separate multiple order keys.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li><i>BUFFER_GETS</i> Sets the order using the SQL statement's buffer-get count value.</li> <li><i>CPU_TIME</i> Sets the order using the SQL statement's CPU time value.</li> <li><i>DISK_READS</i> Sets the order using the SQL statement's disk-read count value.</li> <li><i>ELAPSED_TIME</i> Sets the order using the SQL statement's elapsed time value.</li> <li><i>EXECUTIONS</i> Sets the order using the SQL statement's elapsed time value.</li> <li><i>OPTIMIZER_COST</i> Sets the order using the SQL statement's optimizer cost value.</li> <li><i>PRIORITY</i> Sets the order using the user-supplied business priority value.</li> </ul> <p>All values are accessed in descending order, where a high value is considered more interesting than a low value.</p> <p>The default value is <i>PRIORITY, OPTIMIZER_COST</i>. The datatype is <i>STRINGLIST</i>.</p>
RECOMMEND_MV_EXACT_TEXT_MATCH	<p>When considering candidate materialized views, exact text match solutions will only be included if this parameter contains <i>TRUE</i>.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li><i>TRUE</i></li> <li><i>FALSE</i></li> </ul> <p>The default value is <i>TRUE</i>. The datatype is <i>STRING</i>.</p>
RECOMMENDED_TABLESPACES	<p>Allows the SQL Access Advisor to recommend optimal tablespaces for any partitioning scheme. If this is not set, the SQL Access Advisor will simply recommend a partitioning method but give no advice on physical storage.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li><i>TRUE</i></li> <li><i>FALSE</i> (the default)</li> </ul> <p>The datatype is <i>STRING</i>.</p>
REFRESH_MODE	<p>Specifies whether materialized views are refreshed <i>ON_DEMAND</i> or <i>ON_COMMIT</i>. This will be used to weigh the impact of materialized view refresh when the parameter <i>dml_volatility</i> is set to <i>TRUE</i>.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li><i>ON_DEMAND</i></li> <li><i>ON_COMMIT</i></li> </ul> <p>The default value is <i>ON_DEMAND</i>. The datatype is <i>STRING</i>.</p>
REPORT_DATE_FORMAT	<p>This is the default date and time formatting template. The default format is <i>DD/MM/YYYYHH24:MI</i>. The datatype is <i>STRING</i>.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
SHOW_RETAINS	<p>Controls the display of RETAIN actions within an implementation script and the SQL Access Advisor wizard.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>■ TRUE</li> <li>■ FALSE</li> </ul> <p>The default value is TRUE. The datatype is STRING.</p>
SQL_LIMIT	<p>Specifies the number of SQL statements to be analyzed. The SQL_LIMIT filter is applied after all other filters have been applied. For example, if only statements referencing the table foo.bar are to be accepted, the SQL_LIMIT value will be only apply to those statements.</p> <p>When used in conjunction with the parameter ORDER_LIST, SQL Access Advisor will process the most interesting SQL statements by ordering the statements according to the specified sort keys.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>■ an integer in the range of 1 to 2147483647</li> <li>■ ADVISOR_UNLIMITED</li> <li>■ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is NUMBER.</p>
START_TIME	<p>Specifies a start time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed.</p> <p>Each date must be in the standard Oracle form of MM-DD-YYYY HH24:MI:SS, where:</p> <ul style="list-style-type: none"> <li>■ DD is the numeric date</li> <li>■ MM is the numeric month</li> <li>■ YYYY is the numeric year</li> <li>■ HH is the hour in 24 hour format</li> <li>■ MI is the minute</li> <li>■ SS is the second</li> </ul> <p>The datatype is STRING.</p>
STORAGE_CHANGE	<p>Contains the amount of space adjustment that can be consumed by SQL Access Advisor recommendations. Zero or negative values are only permitted if the workload scope is marked as FULL.</p> <p>When the SQL Access Advisor produces a set of recommendations, the resultant physical structures must be able to fit into the budgeted space. A space budget is computed by adding the STORAGE_CHANGE value to the space quantity currently used by existing access structures. A negative STORAGE_CHANGE value may force SQL Access Advisor to remove existing structures in order to shrink space demand.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>■ Any valid integer including negative values, zero and positive values.</li> </ul> <p>The default value is ADVISOR_UNLIMITED. The datatype is NUMBER.</p>
TIME_LIMIT	<p>Specifies the time in minutes that the SQL Access Advisor can use to perform an analysis operation. If the SQL Access Advisor reaches a specified recommendation quality or all input data has been analyzed, processing will terminate regardless of any remaining time.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>■ An integer in the range of 1 to 10,000</li> <li>■ ADVISOR_UNLIMITED</li> </ul> <p>The default value is 720 (12 hours). The datatype is NUMBER.</p> <p>Note that specifying ADVISOR_UNLIMITED has the same effect as setting the parameter to the maximum of 10,000 (about one week). The SQL Access Advisor will never run for more than 10,000 minutes.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
VALID_ACTION_LIST	<p>Contains a fully qualified list of actions that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness.</p> <p>During a task execution, if a SQL statement's action does not match a name in the action list, it will not be processed by the task. An action name is case sensitive.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single action</li> <li>▪ comma-delimited action list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>
VALID_MODULE_LIST	<p>Contains a fully qualified list of application modules that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness.</p> <p>During a task execution, if a SQL statement's module does not match a name in the module list, it will not be processed by the task. A module name is case sensitive.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single application</li> <li>▪ comma-delimited module list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>
VALID_SQLSTRING_LIST	<p>Contains a fully qualified list of text strings that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness.</p> <p>During a task execution, if a SQL statement does not contain string in the SQL string list, it will not be processed by the task.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single string</li> <li>▪ comma-delimited string list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>

**Table 18–35 (Cont.) SQL Access Advisor Task Parameters**

Parameter	Description
VALID_TABLE_LIST	<p>Contains a fully qualified list of tables that are eligible for tuning. The list elements are comma-delimited, and quoted identifiers are supported. Wildcard specifications are supported for tables. The default value is all tables within the user's scope are eligible for tuning. Supported wildcard character is %. A % wildcard matches any set of consecutive characters.</p> <p>When a SQL statement is processed, it will not be accepted unless at least one referenced table is specified in the valid table list. If the list is unused, then all table references within a SQL statement are considered valid.</p> <p>The valid syntax for a table reference is:</p> <ul style="list-style-type: none"> <li>▪ schema.table</li> <li>▪ schema</li> <li>▪ schema.% (equivalent to schema)</li> <li>▪ comma-delimited action list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single table reference</li> <li>▪ comma-delimited reference list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is TABLELIST.</p>
VALID_USERNAME_LIST	<p>Contains a fully qualified list of usernames that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported.</p> <p>During a task execution, if a SQL statement's username does not match a name in the username list, it will not be processed by the task. A username is not case sensitive unless it is quoted.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>▪ single username</li> <li>▪ comma-delimited username list</li> <li>▪ ADVISOR_UNUSED</li> </ul> <p>The default value is ADVISOR_UNUSED. The datatype is STRINGLIST.</p>
WORKLOAD_SCOPE	<p>Describes the level of application coverage the workload represents. Possible values are FULL and PARTIAL.</p> <p>FULL Should be used if the workload contains all interesting application SQL statements for the targeted tables.</p> <p>PARTIAL (default) Should be used if the workload contains anything less than a full representation of the interesting application SQL statements for the targeted tables.</p> <p>The datatype is STRING.</p>

## Segment Advisor Parameters

Table 18–36 lists the input task parameters that can be set in the Segment Advisor using the SET\_TASK\_PARAMETER procedure.

**Table 18–36 Segment Advisor Task Parameters**

Parameter	Description
MODE	<p>The data to use for analysis. The default value is COMPREHENSIVE, and the possible values are:</p> <ul style="list-style-type: none"> <li>▪ LIMITED: Analysis restricted to statistics available in the Automatic Workload Repository</li> <li>▪ COMPREHENSIVE: Analysis based on sampling and Automatic Workload Repository statistics</li> </ul>
TIME_LIST	<p>The time limit for which the Advisor should run. It is specified in seconds, and the default and possible values are UNLIMITED.</p>

**Table 18–36 (Cont.) Segment Advisor Task Parameters**

Parameter	Description
RECOMMEND_ALL	Whether to generate recommendations for all segments.  The default value is TRUE. If set to TRUE, it generates recommendations all segments specified by the user. If set to FALSE, it generates recommendations for only those objects that are eligible for shrink.

## Examples

```

DECLARE
    task_id NUMBER;
    task_name VARCHAR2(30);
BEGIN
    task_name := 'My Task';

    DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
    DBMS_ADVISOR.SET_TASK_PARAMETER(task_name, 'VALID_TABLELIST',
        'SH.%,SCOTT.EMP');
END;
/

```

## Undo Advisor Task Parameters

Table 18–37 lists the input task parameters that can be set in the Undo Advisor using the SET\_TASK\_PARAMETER procedure.

**Table 18–37 Undo Advisor Task Parameters**

Parameter	Description
TARGET_OBJECTS	The undo tablespace of the system. There is no default value, and the possible value is UNDO_TBS.
START_SNAPSHOT	The starting time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value and the possible values are the valid snapshot numbers in the AWR repository.
END_SNAPSHOT	The ending time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value and the possible values are the valid snapshot numbers in the AWR repository.
BEGIN_TIME_SEC	The number of seconds between the beginning time of the period and now. Describes a period of time for the system to perform analysis. BEGIN_TIME_SEC should be greater than END_TIME_SEC. There is no default value and the possible values are any positive integer.
END_TIME_SEC	The number of seconds between the ending time of the period and now. END_TIME_SEC should be less than BEGIN_TIME_SEC. There is no default value and the possible values are any positive integer.

## Examples

```

DECLARE
    tname VARCHAR2(30);
    oid NUMBER;
BEGIN
    DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
    DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null', oid);

```



```

DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
DBMS_ADVISOR.EXECUTE_TASK(tname);
END;
/

```

## Automatic Database Diagnostic Monitor (ADDM) Task Parameters

Table 18–38 lists the input task parameters that can be set in ADDM using the SET\_TASK\_PARAMETER procedure. See *Oracle Database Performance Tuning Guide* for more information on using these parameters.

**Table 18–38** ADDM Task Parameters

Parameter	Description
START_SNAPSHOT	The starting time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value, and the possible values are the valid snapshot numbers in the AWR repository.
END_SNAPSHOT	The ending time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value, and the possible values are the valid snapshot numbers in the AWR repository.
DB_ID	The database for START_SNAPSHOT and END_SNAPSHOT. The default value is the current database ID.
INSTANCE	The instance for START_SNAPSHOT and END_SNAPSHOT. The default value is 0 or UNUSED, and the possible values are all positive integers. By default, all instances are analyzed.
INSTANCES	If the INSTANCE parameter has been set, INSTANCES is ignored. The default value is UNUSED, and the possible values are comma-separated list of instance numbers (for example, "1, 3, 5"). By default, all instances are analyzed.
DBIO_EXPECTED	The average time to read the database block in microseconds. The default value is 10 milliseconds, and the possible values are system-dependent.

## Examples

The following creates and executes an ADDM task for the current database and an AWR snapshot range between 19 and 26. Note that this example will analyze all instances, whether you have only one or an Oracle RAC database.

```

DECLARE
  tid      NUMBER;
  tname    VARCHAR2(30) := 'ADDM_TEST';
BEGIN
  DBMS_ADVISOR.CREATE_TASK('ADDM', tid, tname, 'my test');
  DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', '19');
  DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', '26');
  DBMS_ADVISOR.EXECUTE_TASK(tname);
END;
/

```

**See Also:**

- *Oracle Database Performance Tuning Guide* for more information regarding ADDM usage
- The [DBMS\\_ADDM](#) package for details on how to create and execute ADDM tasks

**SQL Tuning Advisor Task Parameters**

See the [DBMS\\_SQLTUNE](#) package on page 139-1 and *Oracle Database Performance Tuning Guide* for more information.

## TUNE\_MVIEW Procedure

This procedure shows how to decompose a materialized view into two or more materialized views and to restate the materialized view in a way that is more advantageous for fast refresh and query rewrite. It also shows how to fix materialized view logs and to enable query rewrite.

### Syntax

```
DBMS_ADVISOR.TUNE_MVIEW (
  task_name IN OUT VARCHAR2,
  mv_create_stmt IN [CLOB | VARCHAR2]);
```

### Parameters

**Table 18–39 TUNE\_MVIEW Procedure Parameters**

Parameter	Description
task_name	The task name for looking up the results in a catalog view. If not specified, the system will generate a name and return.
mv_create_stmt	The original materialized view creation statement.

**See Also:** *Oracle Database Performance Tuning Guide* for more information about using the TUNE\_MVIEW procedure

### Usage Notes

Executing TUNE\_MVIEW generates two sets of output results: one is for CREATE implementation and the other is for undoing the CREATE MATERIALIZED VIEW implementation. The output results are accessible through USER\_TUNE\_MVIEW and DBA\_TUNE\_MVIEW views. You can also use DBMS\_ADVISOR.GET\_TASK\_SCRIPT and DBMS\_ADVISOR.CREATE\_FILE to output the TUNE\_MVIEW results into a script file for later execution.

#### USER\_TUNE\_MVIEW and DBA\_TUNE\_MVIEW Views

These views are to get the result after executing the TUNE\_MVIEW procedure.

**Table 18–40 USER\_TUNE\_MVIEW and DBA\_TUNE\_MVIEW Views**

Column Name	Column Description
OWNER	The materialized view owner's name.
TASK_NAME	The task name as a key to access the set of recommendations
SCRIPT_TYPE	Recommendation ID used to indicate the row is for IMPLEMENTATION or UNDO script.
ACTION_ID	Action ID used as the command order number.
STATEMENT	For TUNE_MVIEW output, this column represents the following statements, and includes statement properties such as REFRESH and REWRITE options: <ul style="list-style-type: none"> <li>■ CREATE MATERIALIZED VIEW LOG</li> <li>■ ALTER MATERIALIZED VIEW LOG FORCE</li> <li>■ [CREATE   DROP] MATERIALIZED VIEW</li> </ul>

## Examples

```
name VARCHAR2(30);
DBMS_ADVISOR.TUNE_MVIEW.(name, 'SELECT AVG(C1) FROM my_fact_table WHERE c10 = 7');
```

The following is an example to show how to use TUNE\_MVIEW to optimize a CREATE MATERIALIZED VIEW statement:

```
NAME VARCHAR2(30) := 'my_tune_mview_task';
EXECUTE DBMS_ADVISOR.TUNE_MVIEW (name, 'CREATE MATERIALIZED VIEW MY_MV
REFRESH FAST AS SELECT C2, AVG(C1) FROM MY_FACT_TABLE WHERE C10 = 7
GROUP BY C2');
```

You can view the CREATE output results by querying USER\_TUNE\_MVIEW or DBA\_TUNE\_MVIEW as the following example:

```
SELECT * FROM USER_TUNE_MVIEW WHERE TASK_NAME='my_tune_mview_task' AND
SCRIPT_TYPE='CREATE';
```

Alternatively, you can save the output results in an external script file as in the following example:

```
CREATE DIRECTORY TUNE_RESULTS AS  '/myscript_dir' ;
GRANT READ, WRITE ON DIRECTORY TUNE_RESULTS TO PUBLIC;
EXECUTE
DBMS_ADVISOR.CREATE_FILE(DBMS_ADVISOR.GET_TASK_SCRIPT('my_tune_mview_task'), -
'/homes/tune','my_tune_mview_create.sql');
```

The preceding statement will save the CREATE output results in /myscript\_dir/my\_tune\_mview\_create.sql.

## UPDATE\_OBJECT Procedure

This procedure updates an existing task object. Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level.

### Syntax

```
DBMS_ADVISOR.UPDATE_OBJECT (
  task_name      IN VARCHAR2
  object_id      IN NUMBER,
  attr1         IN VARCHAR2 := NULL,
  attr2         IN VARCHAR2 := NULL,
  attr3         IN VARCHAR2 := NULL,
  attr4         IN CLOB := NULL,
  attr5         IN VARCHAR2 := NULL);
```

### Parameters

**Table 18–41 UPDATE\_OBJECT Procedure Parameters**

Parameter	Description
task_name	A valid advisor task name that uniquely identifies an existing task.
object_id	The advisor-assigned object identifier.
attr1	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr2	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr3	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr4	Advisor-specific data. If set to NULL, there will be no effect on the target object.
attr5	Advisor-specific data. If set to null, there will be no effect on the target object.

The attribute parameters have different values depending upon the object type. See *Oracle Database Administrator's Guide* for details regarding these parameters and object types.

### Usage Notes

If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or lob column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

See *Oracle Database Administrator's Guide* for further information regarding the Segment Advisor.

### Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
```

```
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name, 'SQL', NULL, NULL, NULL,
                              'SELECT * FROM SH.SALES', obj_id);
  DBMS_ADVISOR.UPDATE_OBJECT (task_name, obj_id, NULL, NULL, NULL,
                              'SELECT count(*) FROM SH.SALES');
END;
/
```

## UPDATE\_REC\_ATTRIBUTES Procedure

This procedure updates the owner, name, and tablespace for a recommendation.

### Syntax

```
DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES (
  task_name          IN VARCHAR2
  rec_id             IN NUMBER,
  action_id          IN NUMBER,
  attribute_name     IN VARCHAR2,
  value              IN VARCHAR2);
```

### Parameters

**Table 18–42 UPDATE\_REC\_ATTRIBUTES Procedure Parameters**

Parameter	Description
task_name	The task name that uniquely identifies an existing task.
rec_id	The Advisor-generated identifier number that is assigned to the recommendation.
action_id	The Advisor-generated action identifier that is assigned to the particular command.
attribute_name	Name of the attribute to be changed. The valid values are: <ul style="list-style-type: none"> <li>▪ owner The new owner of the object.</li> <li>▪ name The new name of the object.</li> <li>▪ tablespace The new tablespace for the object.</li> </ul>
value	Specifies the new value for the recommendation attribute.

### Usage Notes

Recommendation attributes cannot be modified unless the task has successfully executed.

### Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
    100,400,5041,103,640445,680000,2,
    1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
    FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

  attribute := 'SH';
```

```
    DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES(task_name, 1, 3, 'OWNER', attribute);  
END;  
/
```



## UPDATE\_SQLWKLD\_ATTRIBUTES Procedure

---

**Note:** This procedure is deprecated in Release 11gR1.

---

This procedure changes various attributes of a SQL Workload object or template.

### Syntax

```
DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES (
  workload_name      IN VARCHAR2,
  new_name           IN VARCHAR2 := NULL,
  description        IN VARCHAR2 := NULL,
  read_only         IN VARCHAR2 := NULL,
  is_template       IN VARCHAR2 := NULL,
  how_created       IN VARCHAR2 := NULL);
```

### Parameters

**Table 18–43** UPDATE\_SQLWKLD\_ATTRIBUTES Procedure Parameters

Parameter	Description
workload_name	The workload object name that uniquely identifies an existing workload.
new_name	The new workload object name. If the value is NULL or contains the value ADVISOR_UNUSED, the workload will not be renamed. A task name can be up to 30 characters long.
description	A new workload description. If the value is NULL or contains the value ADVISOR_UNUSED, the description will not be changed. Names can be up to 256 characters long.
read_only	Set to TRUE so it cannot be changed.
is_template	TRUE if workload is to be used as a template.
how_created	Indicates a source application name that initiated the workload creation. If the value is NULL or contains the value ADVISOR_UNUSED, the source will not be changed.

### Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES(workload_name,'New workload name');
END;
/
```

## UPDATE\_SQLWKLD\_STATEMENT Procedure

---



---

**Note:** This procedure is deprecated in Release 11gR1.

---



---

This procedure updates an existing SQL statement in a specified SQL workload.

### Syntax

```
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (
    workload_name    IN VARCHAR2,
    sql_id           IN NUMBER,
    application      IN VARCHAR2 := NULL,
    action          IN VARCHAR2 := NULL,
    priority         IN NUMBER := NULL,
    username        IN VARCHAR2 := NULL);
```

```
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (
    workload_name    IN VARCHAR2,
    search           IN VARCHAR2,
    updated          OUT NUMBER,
    application      IN VARCHAR2 := NULL,
    action          IN VARCHAR2 := NULL,
    priority         IN NUMBER := NULL,
    username        IN VARCHAR2 := NULL);
```

### Parameters

**Table 18–44 UPDATE\_SQLWKLD\_STATEMENT Procedure Parameters**

Parameter	Description
workload_name	The SQL Workload object name that uniquely identifies an existing workload.
sql_id	The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant <code>DBMS_ADVISOR.ADVISOR_ALL</code> .
updated	Returns the number of statements changed by a searched update.
application	Specifies a business application name that will be associated with the SQL statement. If the value is <code>NULL</code> or contains the value <code>ADVISOR_UNUSED</code> , then the column will not be updated in the repository.
action	Specifies the application action for the statement. If the value is <code>NULL</code> or contains the value <code>ADVISOR_UNUSED</code> , then the column will not be updated in the repository.
priority	The relative priority of the SQL statement. The value must be one of the following: 1 - HIGH, 2 - MEDIUM, or 3 - LOW.  If the value is <code>NULL</code> or contains the value <code>ADVISOR_UNUSED</code> , then the column will not be updated in the repository.
username	The Oracle user name that executed the SQL statement. If the value is <code>NULL</code> or contains the value <code>ADVISOR_UNUSED</code> , then the column will not be updated in the repository.  Because a username is an Oracle identifier, the username value must be entered exactly like it is stored in the server. For example, if the user <code>SCOTT</code> is the executing user, then you must provide the user identifier <code>SCOTT</code> in all uppercase letters. It will not recognize the user <code>scott</code> as a match for <code>SCOTT</code> .

**Table 18-44 (Cont.) UPDATE\_SQLWKLD\_STATEMENT Procedure Parameters**

Parameter	Description
search	Disabled.

## Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See [RESET\\_TASK Procedure](#) on page 18-46 to set a task to its initial state.

## Examples

```

DECLARE
  workload_name VARCHAR2(30);
  updated NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');

  SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
  WHERE workload_name = 'My Workload';

  DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT(workload_name, id);
END;
/

```

## UPDATE\_TASK\_ATTRIBUTES Procedure

This procedure changes various attributes of a task or a task template.

### Syntax

```
DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES (
  task_name      IN VARCHAR2
  new_name       IN VARCHAR2 := NULL,
  description    IN VARCHAR2 := NULL,
  read_only     IN VARCHAR2 := NULL,
  is_template    IN VARCHAR2 := NULL,
  how_created   IN VARCHAR2 := NULL);
```

### Parameters

**Table 18–45** UPDATE\_TASK\_ATTRIBUTES Procedure Parameters

Parameter	Description
task_name	The Advisor task name that uniquely identifies an existing task.
new_name	The new Advisor task name. If the value is NULL or contains the value ADVISOR_UNUSED, the task will not be renamed. A task name can be up to 30 characters long.
description	A new task description. If the value is NULL or contains the value ADVISOR_UNUSED, the description will not be changed. Names can be up to 256 characters long.
read_only	Sets the task to read-only. Possible values are: TRUE and FALSE. If the value is NULL or contains the value ADVISOR_UNUSED, the setting will not be changed.
is_template	Marks the task as a template. Physically, there is no difference between a task and a template; however, a template cannot be executed. Possible values are: TRUE and FALSE. If the value is NULL or contains the value ADVISOR_UNUSED, the setting will not be changed.
how_created	Indicates a source application name that initiated the task creation. If the value is NULL or contains the value ADVISOR_UNUSED, the source will not be changed.

### Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, task_name);
  DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES(task_name, 'New Task Name');
  DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES('New Task Name', NULL, 'New description');
END;
/
```

DBMS\_ALERT supports asynchronous notification of database events (alerts). By appropriate use of this package and database triggers, an application can notify itself whenever values of interest in the database are changed.

This chapter contains the following topics:

- [Using DBMS\\_ALERT](#)
  - Overview
  - Security Model
  - Constants
  - Restrictions
  - Exceptions
  - Operational Notes
  - Examples
- [Summary of DBMS\\_ALERT Subprograms](#)

## Using DBMS\_ALERT

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Restrictions](#)
- [Exceptions](#)
- [Operational Notes](#)
- [Examples](#)

## Overview

Suppose a graphics tool is displaying a graph of some data from a database table. The graphics tool can, after reading and graphing the data, wait on a database alert (`WAITONE`) covering the data just read. The tool automatically wakes up when the data is changed by any other user. All that is required is that a trigger be placed on the database table, which performs a signal (`SIGNAL`) whenever the trigger is fired.

## Security Model

Security on this package can be controlled by granting `EXECUTE` on this package to selected users or roles. You might want to write a cover package on top of this one that restricts the alert names used. `EXECUTE` privilege on this cover package can then be granted rather than on this package.



## Constants

The DBMS\_ALERT package uses the constants shown in [Table 19-1](#):

**Table 19-1** DBMS\_ALERT Constants

Name	Type	Value	Description
MAXWAIT	INTEGER	86400000	The maximum time to wait for an alert (1000 days which is essentially forever).

## Restrictions

Because database alerters issue commits, they cannot be used with Oracle Forms. For more information on restrictions on calling stored procedures while Oracle Forms is active, refer to your Oracle Forms documentation.

## Exceptions

DBMS\_ALERT raises the application error -20000 on error conditions. [Table 19-2](#) shows the messages and the procedures that can raise them.

## Operational Notes

The following notes relate to general and specific applications:

- Alerts are transaction-based. This means that the waiting session is not alerted until the transaction signalling the alert commits. There can be any number of concurrent signalers of a given alert, and there can be any number of concurrent waiters on a given alert.
- A waiting application is blocked in the database and cannot do any other work.
- An application can register for multiple events and can then wait for any of them to occur using the `WAITANY` procedure.
- An application can also supply an optional `timeout` parameter to the `WAITONE` or `WAITANY` procedures. A `timeout` of 0 returns immediately if there is no pending alert.
- The signalling session can optionally pass a message that is received by the waiting session.
- Alerts can be signalled more often than the corresponding application wait calls. In such cases, the older alerts are discarded. The application always gets the latest alert (based on transaction commit times).
- If the application does not require transaction-based alerts, the `DBMS_PIPE` package may provide a useful alternative.

**See Also:** [Chapter 103, "DBMS\\_PIPE"](#)

- If the transaction is rolled back after the call to `SIGNAL`, no alert occurs.
- It is possible to receive an alert, read the data, and find that no data has changed. This is because the data changed after the *prior* alert, but before the data was read for that *prior* alert.
- Usually, Oracle is event-driven; this means that there are no polling loops. There are two cases where polling loops can occur:
  - Shared mode. If your database is running in shared mode, a polling loop is required to check for alerts from another instance. The polling loop defaults to one second and can be set by the `SET_DEFAULTS` procedure.
  - `WAITANY` procedure. If you use the `WAITANY` procedure, and if a signalling session does a signal but does not commit within one second of the signal, a polling loop is required so that this uncommitted alert does not camouflage other alerts. The polling loop begins at a one second interval and exponentially backs off to 30-second intervals.

**Table 19-2** *DBMS\_ALERT* Error Messages

Error Message	Procedure
ORU-10001 lock request error, status: N	SIGNAL
ORU-10015 error: N waiting for pipe status	WAITANY
ORU-10016 error: N sending on pipe 'X'	SIGNAL
ORU-10017 error: N receiving on pipe 'X'	SIGNAL
ORU-10019 error: N on lock request	WAIT
ORU-10020 error: N on lock request	WAITANY

**Table 19-2 (Cont.) DBMS\_ALERT Error Messages**

<b>Error Message</b>	<b>Procedure</b>
ORU-10021 lock request error; status: N	REGISTER
ORU-10022 lock request error, status: N	SIGNAL
ORU-10023 lock request error; status N	WAITONE
ORU-10024 there are no alerts registered	WAITANY
ORU-10025 lock request error; status N	REGISTER
ORU-10037 attempting to wait on uncommitted signal from same session	WAITONE

## Examples

Suppose you want to graph average salaries by department, for all employees. Your application needs to know whenever EMP is changed. Your application would look similar to this code:

```
DBMS_ALERT.REGISTER('emp_table_alert');
<<readagain>>:
/* ... read the emp table and graph it */
DBMS_ALERT.WAITONE('emp_table_alert', :message, :status);
if status = 0 then goto <<readagain>>; else
/* ... error condition */
```

The EMP table would have a trigger similar to this:

```
CREATE TRIGGER emptrig AFTER INSERT OR UPDATE OR DELETE ON emp
BEGIN
    DBMS_ALERT.SIGNAL('emp_table_alert', 'message_text');
END;
```

When the application is no longer interested in the alert, it makes this request:

```
DBMS_ALERT.REMOVE('emp_table_alert');
```

This reduces the amount of work required by the alert signaller. If a session exits (or dies) while registered alerts exist, the alerts are eventually cleaned up by future users of this package.

The example guarantees that the application always sees the latest data, although it may not see every intermediate value.

---

## Summary of DBMS\_ALERT Subprograms

**Table 19–3** *DBMS\_ALERT Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">REGISTER Procedure</a> on page 19-12	Receives messages from an alert
<a href="#">REMOVE Procedure</a> on page 19-13	Disables notification from an alert
<a href="#">REMOVEALL Procedure</a> on page 19-14	Removes all alerts for this session from the registration list
<a href="#">SET_DEFAULTS Procedure</a> on page 19-15	Sets the polling interval
<a href="#">SIGNAL Procedure</a> on page 19-16	Signals an alert (send message to registered sessions)
<a href="#">WAITANY Procedure</a> on page 19-17	Waits <code>timeout</code> seconds to receive alert message from an alert registered for session
<a href="#">WAITONE Procedure</a> on page 19-18	Waits <code>timeout</code> seconds to receive message from named alert

## REGISTER Procedure

This procedure lets a session register interest in an alert.

### Syntax

```
DBMS_ALERT.REGISTER (  
    name      IN  VARCHAR2,  
    cleanup   IN  BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 19–4** REGISTER Procedure Parameters

Parameter	Description
name	Name of the alert in which this session is interested
cleanup	Specifies whether to perform cleanup of any extant orphaned pipes used by the <code>DBMS_ALERT</code> package. This cleanup is only performed on the first call to <code>REGISTER</code> for each package instantiation. The default for the parameter is <code>TRUE</code> .

---

---

**Caution:** Alert names beginning with 'ORA\$' are reserved for use for products provided by Oracle. Names must be 30 bytes or less. The name is case insensitive.

---

---

### Usage Notes

A session can register interest in an unlimited number of alerts. Alerts should be deregistered when the session no longer has any interest, by calling `REMOVE`.



## REMOVE Procedure

This procedure enables a session that is no longer interested in an alert to remove that alert from its registration list. Removing an alert reduces the amount of work done by signalers of the alert.

### Syntax

```
DBMS_ALERT.REMOVE (  
    name IN VARCHAR2);
```

### Parameters

**Table 19–5 REMOVE Procedure Parameters**

Parameter	Description
name	Name of the alert (case-insensitive) to be removed from registration list.

### Usage Notes

Removing alerts is important because it reduces the amount of work done by signalers of the alert. If a session dies without removing the alert, that alert is eventually (but not immediately) cleaned up.

## REMOVEALL Procedure

This procedure removes all alerts for this session from the registration list. You should do this when the session is no longer interested in any alerts.

This procedure is called automatically upon first reference to this package during a session. Therefore, no alerts from prior sessions which may have terminated abnormally can affect this session.

This procedure always performs a commit.

### Syntax

```
DBMS_ALERT.REMOVEALL;
```

## SET\_DEFAULTS Procedure

In case a polling loop is required, use the `SET_DEFAULTS` procedure to set the polling interval.

### Syntax

```
DBMS_ALERT.SET_DEFAULTS (  
    sensitivity IN NUMBER);
```

### Parameters

**Table 19–6** *SET\_DEFAULTS Procedure Parameters*

Parameter	Description
<code>sensitivity</code>	Polling interval, in seconds, to sleep between polls. The default interval is five seconds.

## SIGNAL Procedure

This procedure signals an alert. The effect of the `SIGNAL` call only occurs when the transaction in which it is made commits. If the transaction rolls back, `SIGNAL` has no effect.

All sessions that have registered interest in this alert are notified. If the interested sessions are currently waiting, they are awakened. If the interested sessions are not currently waiting, they are notified the next time they do a wait call.

Multiple sessions can concurrently perform signals on the same alert. Each session, as it signals the alert, blocks all other concurrent sessions until it commits. This has the effect of serializing the transactions.

### Syntax

```
DBMS_ALERT.SIGNAL (
    name      IN  VARCHAR2,
    message   IN  VARCHAR2);
```

### Parameters

**Table 19–7** *SIGNAL Procedure Parameters*

Parameter	Description
name	Name of the alert to signal.
message	Message, of 1800 bytes or less, to associate with this alert.  This message is passed to the waiting session. The waiting session might be able to avoid reading the database after the alert occurs by using the information in the message.

## WAITANY Procedure

Call this procedure to wait for an alert to occur for any of the alerts for which the current session is registered.

### Syntax

```
DBMS_ALERT.WAITANY (
  name      OUT  VARCHAR2,
  message   OUT  VARCHAR2,
  status    OUT  INTEGER,
  timeout   IN   NUMBER DEFAULT MAXWAIT);
```

### Parameters

**Table 19–8** *WAITANY Procedure Parameters*

Parameter	Description
name	Returns the name of the alert that occurred.
message	Returns the message associated with the alert.  This is the message provided by the <code>SIGNAL</code> call. If multiple signals on this alert occurred before <code>WAITANY</code> , the message corresponds to the most recent <code>SIGNAL</code> call. Messages from prior <code>SIGNAL</code> calls are discarded.
status	Values returned: 0 - alert occurred 1 - timeout occurred
timeout	Maximum time to wait for an alert.  If no alert occurs before <code>timeout</code> seconds, this returns a status of 1.

### Usage Notes

An implicit `COMMIT` is issued before this procedure is executed. The same session that waits for the alert may also first signal the alert. In this case remember to commit after the signal and before the wait; otherwise, `DBMS_LOCK.REQUEST` (which is called by `DBMS_ALERT`) returns status 4.

### Exceptions

-20000, ORU-10024: there are no alerts registered.

## WAITONE Procedure

This procedure waits for a specific alert to occur. An implicit `COMMIT` is issued before this procedure is executed. A session that is the first to signal an alert can also wait for the alert in a subsequent transaction. In this case, remember to commit after the signal and before the wait; otherwise, `DBMS_LOCK.REQUEST` (which is called by `DBMS_ALERT`) returns status 4.

### Syntax

```
DBMS_ALERT.WAITONE (
    name      IN   VARCHAR2,
    message   OUT  VARCHAR2,
    status    OUT  INTEGER,
    timeout   IN   NUMBER DEFAULT MAXWAIT);
```

### Parameters

**Table 19–9** WAITONE Procedure Parameters

Parameter	Description
<code>name</code>	Name of the alert to wait for.
<code>message</code>	Returns the message associated with the alert.  This is the message provided by the <code>SIGNAL</code> call. If multiple signals on this alert occurred before <code>WAITONE</code> , the message corresponds to the most recent <code>SIGNAL</code> call. Messages from prior <code>SIGNAL</code> calls are discarded.
<code>status</code>	Values returned: 0 - alert occurred 1 - timeout occurred
<code>timeout</code>	Maximum time to wait for an alert.  If the named alert does not occur before <code>timeout</code> seconds, this returns a status of 1.

---

---

## DBMS\_APPLICATION\_INFO

Application developers can use the `DBMS_APPLICATION_INFO` package with Oracle Trace and the SQL trace facility to record names of executing modules or transactions in the database for later use when tracking the performance of various modules and debugging.

This chapter contains the following topics:

- [Using DBMS\\_APPLICATION\\_INFO](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_APPLICATION\\_INFO Subprograms](#)

---

## Using DBMS\_APPLICATION\_INFO

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)



## Overview

Registering the application allows system administrators and performance tuning specialists to track performance by module. System administrators can also use this information to track resource use by module. When an application registers with the database, its name and actions are recorded in the V\$SESSION and V\$SQLAREA views.

## Security Model

---

---

**Note:** The public synonym for `DBMS_APPLICATION_INFO` is not dropped before creation so that you can redirect the public synonym to point to your own package.

---

---

No further privileges are required. The `DBMSAPIN.SQL` script is already run as a part of standard database creation .

## Operational Notes

Your applications should set the name of the module and name of the action automatically each time a user enters that module. The module name could be the name of a form in an Oracle Forms application, or the name of the code segment in an Oracle Precompilers application. The action name should usually be the name or description of the current transaction within a module.

If you want to gather your own statistics based on module, you can implement a wrapper around this package by writing a version of this package in another schema that first gathers statistics and then calls the *SYS* version of the package. The public synonym for *DBMS\_APPLICATION\_INFO* can then be changed to point to the *DBA*'s version of the package.

## Summary of DBMS\_APPLICATION\_INFO Subprograms

**Table 20–1** *DBMS\_APPLICATION\_INFO Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">READ_CLIENT_INFO Procedure</a> on page 20-7	Reads the value of the <code>client_info</code> field of the current session
<a href="#">READ_MODULE Procedure</a> on page 20-8	Reads the values of the <code>module</code> and <code>action</code> fields of the current session
<a href="#">SET_ACTION Procedure</a> on page 20-9	Sets the name of the current action within the current module
<a href="#">SET_CLIENT_INFO Procedure</a> on page 20-10	Sets the <code>client_info</code> field of the session
<a href="#">SET_MODULE Procedure</a> on page 20-11	Sets the name of the module that is currently running to a new module
<a href="#">SET_SESSION_LONGOPS Procedure</a> on page 20-12	Sets a row in the <code>V\$SESSION_LONGOPS</code> table

## READ\_CLIENT\_INFO Procedure

This procedure reads the value of the `client_info` field of the current session.

### Syntax

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO (  
    client_info OUT VARCHAR2);
```

### Parameters

**Table 20–2** *READ\_CLIENT\_INFO Procedure Parameters*

Parameter	Description
<code>client_info</code>	Last client information value supplied to the <code>SET_CLIENT_INFO</code> procedure.

## READ\_MODULE Procedure

This procedure reads the values of the module and action fields of the current session.

### Syntax

```
DBMS_APPLICATION_INFO.READ_MODULE (
    module_name OUT VARCHAR2,
    action_name OUT VARCHAR2);
```

### Parameters

**Table 20–3 READ\_MODULE Procedure Parameters**

Parameter	Description
module_name	Last value that the module name was set to by calling SET_MODULE.
action_name	Last value that the action name was set to by calling SET_ACTION or SET_MODULE.

### Usage Notes

Module and action names for a registered application can be retrieved by querying V\$SQLAREA or by calling the READ\_MODULE procedure. Client information can be retrieved by querying the V\$SESSION view, or by calling the [READ\\_CLIENT\\_INFO Procedure](#).

### Examples

The following sample query illustrates the use of the MODULE and ACTION column of the V\$SQLAREA.

```
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';

SQL_TEXT DISK_READS MODULE ACTION
-----
INSERT INTO emp 1 add_employee insert into emp
(ename, empno, sal, mgr, job, hiredate, comm, deptno)
VALUES
(name, next.emp_seq, manager, title, SYSDATE, commission, department)

1 row selected.
```

## SET\_ACTION Procedure

This procedure sets the name of the current action within the current module.

### Syntax

```
DBMS_APPLICATION_INFO.SET_ACTION (
    action_name IN VARCHAR2);
```

### Parameters

**Table 20–4 SET\_ACTION Procedure Parameters**

Parameter	Description
action_name	The name of the current action within the current module. When the current action terminates, call this procedure with the name of the next action if there is one, or NULL if there is not. Names longer than 32 bytes are truncated.

### Usage Notes

The action name should be descriptive text about the current action being performed. You should probably set the action name before the start of every transaction.

Set the transaction name to NULL after the transaction completes, so that subsequent transactions are logged correctly. If you do not set the transaction name to NULL, subsequent transactions may be logged with the previous transaction's name.

### Example

The following is an example of a transaction that uses the registration procedure:

```
CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN

-- balance transfer transaction

    DBMS_APPLICATION_INFO.SET_ACTION(
        action_name => 'transfer from chk to sav');
    UPDATE chk SET bal = bal + :amt
        WHERE acct# = :acct;
    UPDATE sav SET bal = bal - :amt
        WHERE acct# = :acct;
    COMMIT;
    DBMS_APPLICATION_INFO.SET_ACTION(null);

END;
```

## SET\_CLIENT\_INFO Procedure

This procedure supplies additional information about the client application.

### Syntax

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO (  
    client_info IN VARCHAR2);
```

### Parameters

**Table 20–5** SET\_CLIENT\_INFO Procedure Parameters

Parameter	Description
client_info	Supplies any additional information about the client application. This information is stored in the V\$SESSION view. Information exceeding 64 bytes is truncated.

---

---

**Note:** CLIENT\_INFO is readable and writable by any user. For storing secured application attributes, you can use the application context feature.

---

---



## SET\_MODULE Procedure

This procedure sets the name of the current application or module.

### Syntax

```
DBMS_APPLICATION_INFO.SET_MODULE (
  module_name IN VARCHAR2,
  action_name IN VARCHAR2);
```

### Parameters

**Table 20–6 SET\_MODULE Procedure Parameters**

Parameter	Description
module_name	Name of module that is currently running. When the current module terminates, call this procedure with the name of the new module if there is one, or NULL if there is not. Names longer than 48 bytes are truncated.
action_name	Name of current action within the current module. If you do not want to specify an action, this value should be NULL. Names longer than 32 bytes are truncated.

### Usage Notes

### Example

```
CREATE or replace PROCEDURE add_employee(
  name VARCHAR2,
  salary NUMBER,
  manager NUMBER,
  title VARCHAR2,
  commission NUMBER,
  department NUMBER) AS
BEGIN
  DBMS_APPLICATION_INFO.SET_MODULE(
    module_name => 'add_employee',
    action_name => 'insert into emp');
  INSERT INTO emp
    (ename, empno, sal, mgr, job, hiredate, comm, deptno)
  VALUES (name, emp_seq.nextval, salary, manager, title, SYSDATE,
    commission, department);
  DBMS_APPLICATION_INFO.SET_MODULE(null,null);
END;
```

## SET\_SESSION\_LONGOPS Procedure

This procedure sets a row in the `V$SESSION_LONGOPS` view. This is a view that is used to indicate the on-going progress of a long running operation. Some Oracle functions, such as parallel execution and Server Managed Recovery, use rows in this view to indicate the status of, for example, a database backup.

Applications may use the `SET_SESSION_LONGOPS` procedure to advertise information on the progress of application specific long running tasks so that the progress can be monitored by way of the `V$SESSION_LONGOPS` view.

### Syntax

```
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS (
  rindex      IN OUT BINARY_INTEGER,
  slno        IN OUT BINARY_INTEGER,
  op_name     IN      VARCHAR2         DEFAULT NULL,
  target      IN      BINARY_INTEGER  DEFAULT 0,
  context     IN      BINARY_INTEGER  DEFAULT 0,
  sofar       IN      NUMBER           DEFAULT 0,
  totalwork   IN      NUMBER           DEFAULT 0,
  target_desc IN      VARCHAR2        DEFAULT 'unknown target',
  units       IN      VARCHAR2        DEFAULT NULL)

set_session_longops_nohint constant BINARY_INTEGER := -1;
```

### Parameters

**Table 20–7 SET\_SESSION\_LONGOPS Procedure Parameters**

Parameter	Description
<code>rindex</code>	A token which represents the <code>v\$session_longops</code> row to update. Set this to <code>set_session_longops_nohint</code> to start a new row. Use the returned value from the prior call to reuse a row.
<code>slno</code>	Saves information across calls to <code>set_session_longops</code> : It is for internal use and should not be modified by the caller.
<code>op_name</code>	Specifies the name of the long running task. It appears as the <code>OPNAME</code> column of <code>v\$session_longops</code> . The maximum length is 64 bytes.
<code>target</code>	Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the <code>TARGET</code> column of <code>v\$session_longops</code> .
<code>context</code>	Any number the client wants to store. It appears in the <code>CONTEXT</code> column of <code>v\$session_longops</code> .
<code>sofar</code>	Any number the client wants to store. It appears in the <code>SOFAR</code> column of <code>v\$session_longops</code> . This is typically the amount of work which has been done so far.
<code>totalwork</code>	Any number the client wants to store. It appears in the <code>TOTALWORK</code> column of <code>v\$session_longops</code> . This is typically an estimate of the total amount of work needed to be done in this long running operation.

**Table 20–7 (Cont.) SET\_SESSION\_LONGOPS Procedure Parameters**

Parameter	Description
target_desc	Specifies the description of the object being manipulated in this long operation. This provides a caption for the <code>target</code> parameter. This value appears in the <code>TARGET_DESC</code> field of <code>v\$session_longops</code> . The maximum length is 32 bytes.
units	Specifies the units in which <code>sofar</code> and <code>totalwork</code> are being represented. It appears as the <code>UNITS</code> field of <code>v\$session_longops</code> . The maximum length is 32 bytes.

## Example

This example performs a task on 10 objects in a loop. As the example completes each object, Oracle updates `V$SESSION_LONGOPS` on the procedure's progress.

```

DECLARE
    rindex    BINARY_INTEGER;
    slno      BINARY_INTEGER;
    totalwork number;
    sofar     number;
    obj       BINARY_INTEGER;

BEGIN
    rindex := dbms_application_info.set_session_longops_nohint;
    sofar := 0;
    totalwork := 10;

    WHILE sofar < 10 LOOP
        -- update obj based on sofar
        -- perform task on object target

        sofar := sofar + 1;
        dbms_application_info.set_session_longops(rindex, slno,
            "Operation X", obj, 0, sofar, totalwork, "table", "tables");
    END LOOP;
END;
```



---

---

## DBMS\_APPLY\_ADM

The DBMS\_APPLY\_ADM package provides subprograms to configure and manage Oracle Streams apply processes, XStream outbound servers, and XStream inbound servers.

This chapter contains the following topics:

- [Using DBMS\\_APPLY\\_ADM](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_APPLY\\_ADM Subprograms](#)

**See Also:**

- *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and apply processes
- *Oracle Database XStream Guide* for more information about XStream outbound servers and inbound servers

## Using DBMS\_APPLY\_ADM

This section contains topics which relate to using the DBMS\_APPLY\_ADM package.

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)

## Overview

This package provides interfaces to start, stop, and configure Oracle Streams apply processes, XStream outbound servers, and XStream inbound servers. This package includes subprograms for configuring apply handlers, setting enqueue destinations for messages, and specifying execution directives for messages. This package also provides administrative subprograms that set the instantiation SCN for objects at a destination database. This package also includes subprograms for managing apply errors.

---

---

**Note:**

- For simplicity, this chapter refers to apply processes, XStream outbound servers, and XStream inbound servers as **apply components**. This chapter identifies a specific type of apply component when necessary.
  - Using XStream requires purchasing a license for the Oracle GoldenGate product.
- 
- 

**See Also:**

- *Oracle Streams Concepts and Administration*
- *Oracle Streams Replication Administrator's Guide*
- *Oracle Database XStream Guide*

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

When the DBMS\_APPLY\_ADM package is used to manage an Oracle Streams configuration, it requires that the user is granted the privileges of an Oracle Streams administrator.

When the DBMS\_APPLY\_ADM package is used to manage an XStream configuration, it requires that the user is granted the privileges of an XStream administrator.

---

---

**Note:** The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as setting an apply user. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.

---

---

**See Also:**

- *Oracle Streams Concepts and Administration* for information about configuring an Oracle Streams administrator
- *Oracle Database XStream Guide* for information about configuring an XStream administrator



## Operational Notes

The following sections contain operational notes for this package:

- [Deprecated Apply Component Parameter Value](#)

### Deprecated Apply Component Parameter Value

---

---

**Note:** Oracle recommends that you do not use deprecated apply component parameter values. Support for deprecated features is for backward compatibility only.

---

---

The NONE value for the `commit_serialization` apply component parameter is deprecated. It is replaced by the `DEPENDENT_TRANSACTIONS` value.

**See Also:** [SET\\_PARAMETER Procedure](#) on page 21-56

## Summary of DBMS\_APPLY\_ADM Subprograms

**Table 21-1 DBMS\_APPLY\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ADD_STMT_HANDLER Procedure</a> on page 21-8	Adds a statement DML handler for a specified operation on a specified database object to a single apply component or to all apply components in the database
<a href="#">ALTER_APPLY Procedure</a> on page 21-10	Alters an apply component
<a href="#">COMPARE_OLD_VALUES Procedure</a> on page 21-16	Specifies whether to compare the old value of one or more columns in a row logical change record (row LCR) with the current value of the corresponding columns at the destination site during apply
<a href="#">CREATE_APPLY Procedure</a> on page 21-18	Creates an apply component
<a href="#">CREATE_OBJECT_DEPENDENCY Procedure</a> on page 21-24	Creates an object dependency
<a href="#">DELETE_ALL_ERRORS Procedure</a> on page 21-25	Deletes all the error transactions for the specified apply component
<a href="#">DELETE_ERROR Procedure</a> on page 21-26	Deletes the specified error transaction
<a href="#">DROP_APPLY Procedure</a> on page 21-27	Drops an apply component
<a href="#">DROP_OBJECT_DEPENDENCY Procedure</a> on page 21-29	Drops an object dependency
<a href="#">EXECUTE_ALL_ERRORS Procedure</a> on page 21-30	Reexecutes the error transactions for the specified apply component
<a href="#">EXECUTE_ERROR Procedure</a> on page 21-31	Reexecutes the specified error transaction
<a href="#">GET_ERROR_MESSAGE Function</a> on page 21-34	Returns the message payload from the error queue for the specified message number and transaction identifier
<a href="#">REMOVE_STMT_HANDLER</a> on page 21-36	Removes a statement DML handler for a specified operation on a specified database object from a single apply component or from all apply components in the database
<a href="#">SET_CHANGE_HANDLER Procedure</a> on page 21-38	Sets or unsets a statement DML handler that tracks changes for a specified operation on a specified database object for a single apply component
<a href="#">SET_DML_HANDLER Procedure</a> on page 21-41	Sets a user procedure as a procedure DML handler for a specified operation on a specified database object for a single apply component or for all apply components in the database
<a href="#">SET_ENQUEUE_DESTINATION Procedure</a> on page 21-46	Sets the queue where the apply component automatically enqueues a message that satisfies the specified rule
<a href="#">SET_EXECUTE Procedure</a> on page 21-48	Specifies whether a message that satisfies the specified rule is executed by an apply component

**Table 21-1 (Cont.) DBMS\_APPLY\_ADM Package Subprograms**

Subprogram	Description
<a href="#">SET_GLOBAL_INSTANTIATION_SCN Procedure</a> on page 21-50	Records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas
<a href="#">SET_KEY_COLUMNS Procedures</a> on page 21-53	Records the set of columns to be used as the substitute primary key for local apply purposes and removes existing substitute primary key columns for the specified object if they exist
<a href="#">SET_PARAMETER Procedure</a> on page 21-56	Sets an apply parameter to the specified value
<a href="#">SET_SCHEMA_INSTANTIATION_SCN Procedure</a> on page 21-66	Records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database
<a href="#">SET_TABLE_INSTANTIATION_SCN Procedure</a> on page 21-69	Records the specified instantiation SCN for the specified table in the specified source database
<a href="#">SET_UPDATE_CONFLICT_HANDLER Procedure</a> on page 21-71	Adds, updates, or drops an update conflict handler for the specified object
<a href="#">SET_VALUE_DEPENDENCY Procedure</a> on page 21-75	Sets or removes a value dependency
<a href="#">START_APPLY Procedure</a> on page 21-77	Directs the apply component to start applying messages
<a href="#">STOP_APPLY Procedure</a> on page 21-78	Stops the apply component from applying any messages and rolls back any unfinished transactions being applied

---

**Note:** All procedures commit unless specified otherwise. However, the `GET_ERROR_MESSAGE` function does not commit.

---

## ADD\_STMT\_HANDLER Procedure

This procedure adds a statement DML handler for a specified operation on a specified database object. The procedure adds the statement DML handler to a single apply component or to all apply components in the database.

This procedure is overloaded. One version of this procedure contains the `statement` and `comment` parameters, and the other does not. The `statement` parameter enables you to create the statement DML handler and add it to one or more processes in one step. Otherwise, create the statement DML handler using the `DBMS_STREAMS_HANDLER_ADM` package before adding it to one or more processes.

### See Also:

- [Chapter 147, "DBMS\\_STREAMS\\_HANDLER\\_ADM"](#)
- *Oracle Streams Concepts and Administration*

### Syntax

```
DBMS_APPLY_ADM.ADD_STMT_HANDLER(
  object_name      IN  VARCHAR2,
  operation_name   IN  VARCHAR2,
  handler_name     IN  VARCHAR2,
  statement        IN  CLOB,
  apply_name       IN  VARCHAR2  DEFAULT NULL,
  comment          IN  VARCHAR2  DEFAULT NULL);
```

```
DBMS_APPLY_ADM.ADD_STMT_HANDLER(
  object_name      IN  VARCHAR2,
  operation_name   IN  VARCHAR2,
  handler_name     IN  VARCHAR2,
  apply_name       IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 21–2 ADD\_STMT\_HANDLER Procedure Parameters**

Parameter	Description
<code>object_name</code>	The name of the source object specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure.  If <code>NULL</code> , then the procedure raises an error.
<code>operation_name</code>	The name of the operation, which can be specified as: <ul style="list-style-type: none"> <li>■ <code>INSERT</code></li> <li>■ <code>UPDATE</code></li> <li>■ <code>DELETE</code></li> </ul> If <code>NULL</code> , then the procedure raises an error.  <b>Note:</b> Statement DML handlers cannot be specified for LOB operations.

**Table 21-2 (Cont.) ADD\_STMT\_HANDLER Procedure Parameters**

Parameter	Description
handler_name	<p>The name of the statement DML handler.</p> <p>If the specified statement DML handler exists, then the statement in the <code>statement</code> parameter is added to the existing handler. Ensure that the existing statement DML handler is for the same operation on the same database object as the settings for the <code>operation_name</code> and <code>object_name</code> parameters, respectively. If the existing handler is for a different operation or database object, then an apply error results when the handler is invoked.</p> <p>If the specified statement DML handler does not exist and a non-NULL statement parameter is specified, then this procedure creates the statement DML handler.</p> <p>If the specified statement DML handler does not exist and the <code>statement</code> parameter is not specified or is NULL, then this procedure raises an error.</p> <p>If NULL, then the procedure raises an error.</p>
statement	<p>The text of the SQL statement to add to the statement DML handler.</p> <p>If NULL, then the procedure raises an error.</p>
apply_name	<p>The name of the apply component that uses the statement DML handler.</p> <p>If NULL, then the procedure adds the statement DML handler as a general handler to all apply components in the database.</p>
comment	<p>A comment for the statement DML handler.</p> <p>If NULL, then no comment is recorded for the statement DML handler.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [The ADD\\_STMT\\_HANDLER Procedure and XStream Outbound Servers](#)
- [The ADD\\_STMT\\_HANDLER Procedure and XStream Inbound Servers](#)

### The ADD\_STMT\_HANDLER Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers. Outbound servers ignore all apply handlers.

### The ADD\_STMT\_HANDLER Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

## ALTER\_APPLY Procedure

This procedure alters an apply component.

### Syntax

```
DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name          IN  VARCHAR2,
    rule_set_name       IN  VARCHAR2  DEFAULT NULL,
    remove_rule_set     IN  BOOLEAN   DEFAULT FALSE,
    message_handler     IN  VARCHAR2  DEFAULT NULL,
    remove_message_handler IN  BOOLEAN  DEFAULT FALSE,
    ddl_handler         IN  VARCHAR2  DEFAULT NULL,
    remove_ddl_handler  IN  BOOLEAN   DEFAULT FALSE,
    apply_user          IN  VARCHAR2  DEFAULT NULL,
    apply_tag           IN  RAW        DEFAULT NULL,
    remove_apply_tag    IN  BOOLEAN   DEFAULT FALSE,
    precommit_handler   IN  VARCHAR2  DEFAULT NULL,
    remove_precommit_handler IN  BOOLEAN  DEFAULT FALSE,
    negative_rule_set_name IN  VARCHAR2  DEFAULT NULL,
    remove_negative_rule_set IN  BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 21–3 ALTER\_APPLY Procedure Parameters**

Parameter	Description
apply_name	The name of the apply component being altered. You must specify the name of an existing apply component. Do not specify an owner.
rule_set_name	<p>The name of the positive rule set for the apply component. The positive rule set contains the rules that instruct the apply component to apply messages.</p> <p>If you want to use a positive rule set for the apply component, then you must specify an existing rule set in the form <code>[schema_name.] rule_set_name</code>. For example, to specify a positive rule set in the hr schema named job_apply_rules, enter <code>hr.job_apply_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</p> <p>If you specify NULL and the <code>remove_rule_set</code> parameter is set to FALSE, then this procedure retains any existing positive rule set for the specified apply component. If you specify NULL and the <code>remove_rule_set</code> parameter is set to TRUE, then this procedure removes any existing positive rule set from the specified apply component.</p>

**Table 21-3 (Cont.) ALTER\_APPLY Procedure Parameters**

Parameter	Description
remove_rule_set	<p>If TRUE, then the procedure removes the positive rule set for the specified apply component. If you remove the positive rule set for an apply component, and the apply component does not have a negative rule set, then the apply component dequeues all messages in its queue.</p> <p>If you remove the positive rule set for an apply component, and a negative rule set exists for the apply component, then the apply component dequeues all messages in its queue that are not discarded by the negative rule set.</p> <p>If FALSE, then the procedure retains the positive rule set for the specified apply component.</p> <p>If the <code>rule_set_name</code> parameter is non-NULL, then this parameter should be set to FALSE.</p>
message_handler	<p>A user-defined procedure that processes non-LCR messages in the queue for the apply component.</p> <p>See "Usage Notes" on page 21-22 in the <a href="#">CREATE_APPLY Procedure</a> for more information about a message handler procedure.</p>
remove_message_handler	<p>If TRUE, then the procedure removes the message handler for the specified apply component.</p> <p>If FALSE, then the procedure retains any message handler for the specified apply component.</p> <p>If the <code>message_handler</code> parameter is non-NULL, then this parameter should be set to FALSE.</p>
ddl_handler	<p>A user-defined procedure that processes DDL logical change records (DDL LCRs) in the queue for the apply component.</p> <p>All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically.</p> <p>See "Usage Notes" on page 21-22 in the <a href="#">CREATE_APPLY Procedure</a> for more information about a DDL handler procedure.</p>
remove_ddl_handler	<p>If TRUE, then the procedure removes the DDL handler for the specified apply component.</p> <p>If FALSE, then the procedure retains any DDL handler for the specified apply component.</p> <p>If the <code>ddl_handler</code> parameter is non-NULL, then this parameter should be set to FALSE.</p>

**Table 21–3 (Cont.) ALTER\_APPLY Procedure Parameters**

Parameter	Description
apply_user	<p>The user in whose security domain an apply component dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations, and runs apply handlers. If NULL, then the apply user is not changed.</p> <p>If a non-NULL value is specified to change the apply user, then the user who invokes the ALTER_APPLY procedure must be granted the DBA role. Only the SYS user can set the apply_user to SYS.</p> <p>If you change the apply user, then this procedure grants the new apply user dequeue privilege on the queue used by the apply component. It also configures the user as a secure queue user of the queue.</p> <p>In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user:</p> <ul style="list-style-type: none"> <li>■ The necessary privileges to perform DML and DDL changes on the apply objects</li> <li>■ EXECUTE privilege on the rule sets used by the apply component</li> <li>■ EXECUTE privilege on all rule-based transformation functions used in the rule set</li> <li>■ EXECUTE privilege on all apply handler procedures</li> </ul> <p>These privileges can be granted directly to the apply user, or they can be granted through roles.</p> <p>In addition, the apply user must be granted the EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply component. These privileges must be granted directly to the apply user. They cannot be granted through roles.</p> <p>By default, this parameter is set to the user who created the apply component by running either the CREATE_APPLY procedure in this package or a procedure in the DBMS_STREAMS_ADM package.</p> <p><b>Note:</b> If the apply user for an apply component is dropped using DROP USER . . . CASCADE, then the apply component is also dropped automatically.</p>
apply_tag	<p>A binary tag that is added to redo entries generated by the specified apply component. The tag is a binary value that can be used to track LCRs.</p> <p>The tag is relevant only if a capture process at the database where the apply component is running captures changes made by the apply component. If so, then the captured changes include the tag specified by this parameter.</p> <p>If NULL, the default, then the apply tag for the apply component is not changed.</p> <p>The following is an example of a tag with a hexadecimal value of 17:</p> <pre>HEXTORAW ( ' 17 ' )</pre> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>



**Table 21-3 (Cont.) ALTER\_APPLY Procedure Parameters**

Parameter	Description
remove_apply_tag	<p>If TRUE, then the procedure sets the apply tag for the specified apply component to NULL, and the apply component generates redo entries with NULL tags.</p> <p>If FALSE, then the procedure retains any apply tag for the specified apply component.</p> <p>If the apply_tag parameter is non-NULL, then this parameter should be set to FALSE.</p>
precommit_handler	<p>A user-defined procedure that can receive internal commit directives in the queue for the apply component before they are processed by the apply component. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply component.</p> <p>An internal commit directive is enqueued in the following ways:</p> <ul style="list-style-type: none"> <li>■ When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs.</li> <li>■ When a user or application enqueues messages and then issues a COMMIT statement, the commit directive is enqueued automatically.</li> </ul> <p>For a captured row LCR, a commit directive contains the commit SCN of the transaction from the source database. For a user message, the commit SCN is generated by the apply component.</p> <p>The precommit handler procedure must conform to the following restrictions:</p> <ul style="list-style-type: none"> <li>■ Any work that commits must be an autonomous transaction.</li> <li>■ Any rollback must be to a named savepoint created in the procedure.</li> </ul> <p>If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue.</p> <p>See "Usage Notes" on page 21-22 in the <a href="#">CREATE_APPLY Procedure</a> for more information about a precommit handler procedure.</p>
remove_precommit_handler	<p>If TRUE, then the procedure removes the precommit handler for the specified apply component.</p> <p>If FALSE, then the procedure retains any precommit handler for the specified apply component.</p> <p>If the precommit_handler parameter is non-NULL, then this parameter should be set to FALSE.</p>

**Table 21–3 (Cont.) ALTER\_APPLY Procedure Parameters**

Parameter	Description
<code>negative_rule_set_name</code>	<p>The name of the negative rule set for the apply component. The negative rule set contains the rules that instruct the apply component to discard messages.</p> <p>If you want to use a negative rule set for the apply component, then you must specify an existing rule set in the form <code>[schema_name.]rule_set_name</code>. For example, to specify a negative rule set in the <code>hr</code> schema named <code>neg_apply_rules</code>, enter <code>hr.neg_apply_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify <code>NULL</code> and the <code>remove_negative_rule_set</code> parameter is set to <code>FALSE</code>, then the procedure retains any existing negative rule set. If you specify <code>NULL</code> and the <code>remove_negative_rule_set</code> parameter is set to <code>TRUE</code>, then the procedure removes any existing negative rule set.</p> <p>If you specify both a positive and a negative rule set for an apply component, then the negative rule set is always evaluated first.</p>
<code>remove_negative_rule_set</code>	<p>If <code>TRUE</code>, then the procedure removes the negative rule set for the specified apply component. If you remove the negative rule set for an apply component, and the apply component does not have a positive rule set, then the apply component dequeues all messages in its queue.</p> <p>If you remove the negative rule set for an apply component, and a positive rule set exists for the apply component, then the apply component dequeues all messages in its queue that are not discarded by the positive rule set.</p> <p>If <code>FALSE</code>, then the procedure retains the negative rule set for the specified apply component.</p> <p>If the <code>negative_rule_set_name</code> parameter is non-<code>NULL</code>, then this parameter should be set to <code>FALSE</code>.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [Automatic Restart of Apply Components](#)
- [The ALTER\\_APPLY Procedure and XStream Outbound Servers](#)
- [The ALTER\\_APPLY Procedure and XStream Inbound Servers](#)

### Automatic Restart of Apply Components

An apply component is stopped and restarted automatically when you change the value of one or more of the following `ALTER_APPLY` procedure parameters:

- `message_handler`
- `ddl_handler`
- `apply_user`
- `apply_tag`
- `precommit_handler`

**The ALTER\_APPLY Procedure and XStream Outbound Servers**

The following usage notes apply to this procedure and XStream outbound servers:

- The `apply_user` parameter can change the connect user for an outbound server.
- You cannot specify an apply handler for an outbound server. An outbound server ignores the settings for the following parameters: `message_handler`, `ddl_handler`, and `precommit_handler`.

The client application can perform custom processing of the LCRs instead if necessary. However, if apply processes are configured in the same database as the outbound server, then you can specify apply handlers for these apply processes. In addition, you can configure general apply handlers for the database. An outbound server ignores general apply handlers.

- An outbound server cannot set an apply tag for the changes it processes. An outbound server ignores the setting for the `apply_tag` parameter.

**The ALTER\_APPLY Procedure and XStream Inbound Servers**

Inbound servers can use apply handlers. However, inbound servers only process LCRs. Therefore, inbound servers ignore message handlers specified in the `message_handler` parameter.

## COMPARE\_OLD\_VALUES Procedure

This procedure specifies whether to compare the old values of one or more columns in a row logical change record (row LCR) with the current values of the corresponding columns at the destination site during apply. This procedure is relevant only for UPDATE and DELETE operations because only these operations result in old column values in row LCRs. The default is to compare old values for all columns.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about conflict detection and resolution in an Oracle Streams environment

### Syntax

```
DBMS_APPLY_ADM.COMPARE_OLD_VALUES (
  object_name      IN VARCHAR2,
  column_list      IN VARCHAR2,
  operation        IN VARCHAR2 DEFAULT 'UPDATE',
  compare          IN BOOLEAN  DEFAULT TRUE,
  apply_database_link IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_APPLY_ADM.COMPARE_OLD_VALUES (
  object_name      IN VARCHAR2,
  column_table     IN DBMS_UTILITY.LNAME_ARRAY,
  operation        IN VARCHAR2 DEFAULT 'UPDATE',
  compare          IN BOOLEAN  DEFAULT TRUE,
  apply_database_link IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 21–4 COMPARE\_OLD\_VALUES Procedure Parameters**

Parameter	Description
<code>object_name</code>	The name of the source table specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default.
<code>column_list</code>	A comma-delimited list of column names in the table. There must be no spaces between entries. Specify <code>*</code> to include all nonkey columns.
<code>column_table</code>	A PL/SQL associative array of type <code>DBMS_UTILITY.LNAME_ARRAY</code> that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be NULL terminated.
<code>operation</code>	The name of the operation, which can be specified as: <ul style="list-style-type: none"> <li>■ UPDATE for UPDATE operations</li> <li>■ DELETE for DELETE operations</li> <li>■ * for both UPDATE and DELETE operations</li> </ul>
<code>compare</code>	If <code>compare</code> is <code>TRUE</code> , the old values of the specified columns are compared during apply. If <code>compare</code> is <code>FALSE</code> , the old values of the specified columns are not compared during apply.

**Table 21–4 (Cont.) COMPARE\_OLD\_VALUES Procedure Parameters**

Parameter	Description
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

## Usage Notes

The following usage notes apply to this procedure:

- [Conflict Detection](#)
- [The COMPARE\\_OLD\\_VALUES Procedure and XStream Outbound Servers](#)
- [The COMPARE\\_OLD\\_VALUES Procedure and XStream Inbound Servers](#)

### Conflict Detection

By default, an apply component uses the old column values in a row LCR to detect conflicts. You can choose not to compare old column values to avoid conflict detection for specific tables. For example, if you do not want to compare the old values for a set of columns during apply, then, using the COMPARE\_OLD\_VALUES procedure, specify the set of columns in the `column_list` or `column_table` parameter, and set the `compare` parameter to `FALSE`.

In addition, when the `compare_key_only` apply component parameter is set to `Y`, automatic conflict detection is disabled, and the apply component only uses primary key and unique key columns to identify the table row for a row LCR. When the `compare_key_only` apply component parameter is set to `N`, automatic conflict detection is enabled, and the apply component uses all of the old values in a row LCR to identify the table row for a row LCR.

---



---

#### Note:

- An apply component compares old values for non-key columns when they are present in a row LCR and when the apply component parameter `compare_key_only` is set to `N`.
  - This procedure raises an error if a key column is specified in `column_list` or `column_table` and the `compare` parameter is set to `FALSE`.
- 
- 

**See Also:** [SET\\_PARAMETER Procedure](#) on page 21-56 for more information about the `compare_key_only` apply component parameter

### The COMPARE\_OLD\_VALUES Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

### The COMPARE\_OLD\_VALUES Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

## CREATE\_APPLY Procedure

This procedure creates an apply component.

### Syntax

```
DBMS_APPLY_ADM.CREATE_APPLY(
  queue_name          IN  VARCHAR2,
  apply_name          IN  VARCHAR2,
  rule_set_name       IN  VARCHAR2  DEFAULT NULL,
  message_handler     IN  VARCHAR2  DEFAULT NULL,
  ddl_handler         IN  VARCHAR2  DEFAULT NULL,
  apply_user          IN  VARCHAR2  DEFAULT NULL,
  apply_database_link IN  VARCHAR2  DEFAULT NULL,
  apply_tag           IN  RAW        DEFAULT '00',
  apply_captured      IN  BOOLEAN   DEFAULT FALSE,
  precommit_handler  IN  VARCHAR2  DEFAULT NULL,
  negative_rule_set_name IN VARCHAR2  DEFAULT NULL,
  source_database     IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 21–5 CREATE\_APPLY Procedure Parameters**

Parameter	Description
queue_name	<p>The name of the queue from which the apply component dequeues messages. You must specify an existing queue in the form <code>[ schema_name . ] queue_name</code>. For example, to specify a queue in the <code>hr</code> schema named <code>streams_queue</code>, enter <code>hr.streams_queue</code>. If the schema is not specified, then the current user is the default.</p> <p><b>Note:</b> The <code>queue_name</code> setting cannot be altered after the apply component is created.</p>
apply_name	<p>The name of the apply component being created. A <code>NULL</code> specification is not allowed. Do not specify an owner.</p> <p>The specified name must not match the name of an existing apply component or messaging client.</p> <p><b>Note:</b> The <code>apply_name</code> setting cannot be altered after the apply component is created.</p>
rule_set_name	<p>The name of the positive rule set for the apply component. The positive rule set contains the rules that instruct the apply component to apply messages.</p> <p>If you want to use a positive rule set for the apply component, then you must specify an existing rule set in the form <code>[ schema_name . ] rule_set_name</code>. For example, to specify a positive rule set in the <code>hr</code> schema named <code>job_apply_rules</code>, enter <code>hr.job_apply_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>If you specify <code>NULL</code>, and no negative rule set is specified, then the apply component applies either all captured messages or all messages in the persistent queue, depending on the setting of the <code>apply_captured</code> parameter.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p>

**Table 21-5 (Cont.) CREATE\_APPLY Procedure Parameters**

Parameter	Description
message_handler	<p>A user-defined procedure that processes non-LCR messages in the queue for the apply component.</p> <p>See "Usage Notes" on page 21-22 for more information about a message handler procedure.</p>
ddl_handler	<p>A user-defined procedure that processes DDL logical change record (DDL LCRs) in the queue for the apply component.</p> <p>All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the EXECUTE member procedure of a DDL LCR, then a commit is performed automatically.</p> <p>See "Usage Notes" on page 21-22 for more information about a DDL handler procedure.</p>
apply_user	<p>The user who applies all DML and DDL changes that satisfy the apply component rule sets and who runs user-defined apply handlers. If NULL, then the user who runs the CREATE_APPLY procedure is used.</p> <p>The apply user is the user in whose security domain an apply component dequeues messages that satisfy its rule sets, applies messages directly to database objects, runs custom rule-based transformations configured for apply component rules, and runs apply handlers configured for the apply component. This user must have the necessary privileges to apply changes. This procedure grants the apply user dequeue privilege on the queue used by the apply component and configures the user as a secure queue user of the queue.</p> <p>In addition to the privileges granted by this procedure, you also should grant the following privileges to the apply user:</p> <ul style="list-style-type: none"> <li>■ The necessary privileges to perform DML and DDL changes on the apply objects</li> <li>■ EXECUTE privilege on the rule sets used by the apply component</li> <li>■ EXECUTE privilege on all rule-based transformation functions used in the rule set</li> <li>■ EXECUTE privilege on all apply handler procedures</li> </ul> <p>These privileges can be granted directly to the apply user, or they can be granted through roles.</p> <p>In addition, the apply user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply component. These privileges must be granted directly to the apply user. They cannot be granted through roles.</p> <p><b>Note:</b> If the apply user for an apply component is dropped using DROP USER . . . CASCADE, then the apply component is also dropped automatically.</p> <p>See "Usage Notes" on page 21-22 for more information about this parameter.</p>

**Table 21–5 (Cont.) CREATE\_APPLY Procedure Parameters**

Parameter	Description
apply_database_link	<p>The database at which the apply component applies messages. This parameter is used by an apply component when applying changes from Oracle to non-Oracle systems, such as Sybase. Set this parameter to NULL to specify that the apply component applies messages at the local database.</p> <p><b>Note:</b> The <code>apply_database_link</code> setting cannot be altered after the apply component is created.</p>
apply_tag	<p>A binary tag that is added to redo entries generated by the specified apply component. The tag is a binary value that can be used to track LCRs.</p> <p>The tag is relevant only if a capture process at the database where the apply component is running captures changes made by the apply component. If so, then the captured changes include the tag specified by this parameter.</p> <p>By default, the tag for an apply component is the hexadecimal equivalent of '00' (double zero).</p> <p>The following is an example of a tag with a hexadecimal value of 17:</p> <pre>HEXTORAW('17')</pre> <p>If NULL, then the apply component generates redo entries with NULL tags.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>
apply_captured	<p>Either TRUE or FALSE.</p> <p>If TRUE, then the apply component applies only the captured LCRs in the queue. Captured LCRs are LCRs that were captured by an Oracle Streams capture process.</p> <p>If FALSE, then the apply component applies only the messages in a persistent queue. These are messages that were not captured by an Oracle Streams capture process, such as persistent LCRs or user messages.</p> <p>To apply both captured LCRs and messages in a persistent queue, you must create at least two apply components.</p> <p><b>Note:</b> The <code>apply_captured</code> setting cannot be altered after the apply component is created.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about processing messages with an apply component</p>



**Table 21–5 (Cont.) CREATE\_APPLY Procedure Parameters**

Parameter	Description
precommit_handler	<p>A user-defined procedure that can receive internal commit directives in the queue for the apply component before they are processed by the apply component. Typically, precommit handlers are used for auditing commit information for transactions processed by an apply component.</p> <p>An internal commit directive is enqueued in the following ways:</p> <ul style="list-style-type: none"> <li>■ When a capture process captures row LCRs, the capture process enqueues the commit directive for the transaction that contains the row LCRs.</li> <li>■ When a synchronous capture captures row LCRs, the persistent LCRs that were enqueued by the synchronous capture are organized into a message group. The synchronous capture records the transaction identifier in each persistent LCR in a transaction.</li> <li>■ When a user or application enqueues messages and then issues a COMMIT statement, the commit directive is enqueued automatically.</li> </ul> <p>For a row LCR captured by a capture process or synchronous capture, a commit directive contains the commit SCN of the transaction from the source database. For a message enqueued by a user or application, the commit SCN is generated by the apply component.</p> <p>The precommit handler procedure must conform to the following restrictions:</p> <ul style="list-style-type: none"> <li>■ Any work that commits must be an autonomous transaction.</li> <li>■ Any rollback must be to a named savepoint created in the procedure.</li> </ul> <p>If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue.</p> <p>See "Usage Notes" on page 21-22 for more information about a precommit handler procedure.</p>
negative_rule_set_name	<p>The name of the negative rule set for the apply component. The negative rule set contains the rules that instruct the apply component to discard messages.</p> <p>If you want to use a negative rule set for the apply component, then you must specify an existing rule set in the form <code>[schema_name.]rule_set_name</code>. For example, to specify a negative rule set in the hr schema named neg_apply_rules, enter <code>hr.neg_apply_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>If you specify NULL, and no positive rule set is specified, then the apply component applies either all captured LCRs or all of the messages in the persistent queue, depending on the setting of the apply_captured parameter.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</p> <p>If you specify both a positive and a negative rule set for an apply component, then the negative rule set is always evaluated first.</p>

**Table 21–5 (Cont.) CREATE\_APPLY Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database for the changes that will be applied by the apply component. The source database is the database where the changes originated. If an apply component applies captured messages, then the apply component can apply messages from only one capture process at one source database.</p> <p>If NULL, then the source database name of the first LCR received by the apply component is used for the source database.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1 . NET automatically.</p> <p>The rules in the apply component rule sets determine which messages are dequeued by the apply component. If the apply component dequeues an LCR with a source database that is different than the source database for the apply component, then an error is raised. You can determine the source database for an apply component by querying the DBA_APPLY_PROGRESS data dictionary view.</p>

## Usage Notes

The following sections describe usage notes for this procedure:

- [DBA Role Requirement](#)
- [Handler Procedure Names](#)
- [Message Handler and DDL Handler Procedure](#)
- [Precommit Handler Procedure](#)
- [The CREATE\\_APPLY Procedure and XStream Outbound Servers](#)
- [The CREATE\\_APPLY Procedure and XStream Inbound Servers](#)

### DBA Role Requirement

If the user who invokes this procedure is different from the user specified in the `apply_user` parameter, then the invoking user must be granted the DBA role. If the user who invokes this procedure is the same as the user specified in the `apply_user` parameter, then the DBA role is not required for the invoking user. Only the SYS user can set the `apply_user` to SYS.

### Handler Procedure Names

For the `message_handler`, `ddl_handler`, and `precommit_handler` parameters, specify an existing procedure in one of the following forms:

- `[schema_name.]procedure_name`
- `[schema_name.]package_name.procedure_name`

If the procedure is in a package, then the `package_name` must be specified. For example, to specify a procedure in the `apply_pkg` package in the `hr` schema named `process_ddls`, enter `hr.apply_pkg.process_ddls`. An error is returned if the specified procedure does not exist.

The user who invokes the `CREATE_APPLY` procedure must have `EXECUTE` privilege on a specified handler procedure. Also, if the `schema_name` is not specified, then the user who invokes the `CREATE_APPLY` procedure is the default.

### Message Handler and DDL Handler Procedure

The procedure specified in both the `message_handler` parameter and the `ddl_handler` parameter must have the following signature:

```
PROCEDURE handler_procedure (
    parameter_name IN ANYDATA);
```

Here, `handler_procedure` stands for the name of the procedure and `parameter_name` stands for the name of the parameter passed to the procedure. For the message handler, the parameter passed to the procedure is a `ANYDATA` encapsulation of a user message. For the DDL handler procedure, the parameter passed to the procedure is a `ANYDATA` encapsulation of a DDL LCR.

**See Also:** [Chapter 248, "Logical Change Record TYPES"](#) for information about DDL LCRs

### Precommit Handler Procedure

The procedure specified in the `precommit_handler` parameter must have the following signature:

```
PROCEDURE handler_procedure (
    parameter_name IN NUMBER);
```

Here, `handler_procedure` stands for the name of the procedure and `parameter_name` stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is the commit SCN of a commit directive.

### The CREATE\_APPLY Procedure and XStream Outbound Servers

This procedure cannot create an XStream outbound servers. To create an XStream outbound server, use the `DBMS_XSTREAM_ADM` package.

### The CREATE\_APPLY Procedure and XStream Inbound Servers

The following usage notes apply to this procedure and XStream inbound servers:

- The `CREATE_APPLY` procedure always creates an apply process. The apply process remains an apply process if it receives messages from a source other than an XStream client application, such as a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives messages from any other source. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.
- When creating an inbound server using the `CREATE_APPLY` procedure, set the `apply_captured` parameter to `TRUE`. Inbound servers only process LCRs captured by a capture process.
- Inbound servers can use apply handlers. However, inbound servers only process LCRs. Therefore, inbound servers ignore message handlers specified in the `message_handler` parameter.

## CREATE\_OBJECT\_DEPENDENCY Procedure

This procedure creates an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

An apply component schedules execution of transactions that involve the child object after all transactions with a lower commit system change number (commit SCN) that involve the parent object have been committed. An apply component uses the object identifier of the objects in the logical change records (LCRs) to detect dependencies. The apply component does not use column values in the LCRs to detect dependencies.

---

**Note:** An error is raised if NULL is specified for either of the procedure parameters.

---

### See Also:

- [DROP\\_OBJECT\\_DEPENDENCY Procedure](#) on page 21-29
- *Oracle Streams Concepts and Administration*

## Syntax

```
DBMS_APPLY_ADM.CREATE_OBJECT_DEPENDENCY (
  object_name          IN  VARCHAR2,
  parent_object_name  IN  VARCHAR2);
```

## Parameters

**Table 21–6 CREATE\_OBJECT\_DEPENDENCY Procedure Parameters**

Parameter	Description
object_name	The name of the child database object, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.
parent_object_name	The name of the parent database object, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, hr.departments. If the schema is not specified, then the current user is the default.

## Usage Notes

The following usage notes apply to this procedure:

- [The CREATE\\_OBJECT\\_DEPENDENCY Procedure and XStream Outbound Servers](#)
- [The CREATE\\_OBJECT\\_DEPENDENCY Procedure and XStream Inbound Servers](#)

### The CREATE\_OBJECT\_DEPENDENCY Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

### The CREATE\_OBJECT\_DEPENDENCY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

## DELETE\_ALL\_ERRORS Procedure

This procedure deletes all the error transactions for the specified apply component.

### Syntax

```
DBMS_APPLY_ADM.DELETE_ALL_ERRORS (
  apply_name IN VARCHAR2 DEFAULT NULL);
```

### Parameter

**Table 21–7** *DELETE\_ALL\_ERRORS Procedure Parameter*

Parameter	Description
apply_name	The name of the apply component that raised the errors while processing the transactions. Do not specify an owner.  If NULL, then all error transactions for all apply components are deleted.

### Usage Notes

The following usage notes apply to this procedure:

- [The DELETE\\_ALL\\_ERRORS Procedure and XStream Outbound Servers](#)
- [The DELETE\\_ALL\\_ERRORS Procedure and XStream Inbound Servers](#)

#### **The DELETE\_ALL\_ERRORS Procedure and XStream Outbound Servers**

Outbound servers do not enqueue error transactions into an error queue. This procedure has no effect on XStream outbound servers.

#### **The DELETE\_ALL\_ERRORS Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## DELETE\_ERROR Procedure

This procedure deletes the specified error transaction.

### Syntax

```
DBMS_APPLY_ADM.DELETE_ERROR(  
    local_transaction_id IN VARCHAR2);
```

### Parameter

**Table 21–8** *DELETE\_ERROR Procedure Parameter*

Parameter	Description
local_transaction_id	The identification number of the error transaction to delete. If the specified transaction does not exist in the error queue, then an error is raised.

### Usage Notes

The following usage notes apply to this procedure:

#### **The DELETE\_ERROR Procedure and XStream Outbound Servers**

Outbound servers do not enqueue error transactions into an error queue. This procedure has no effect on XStream outbound servers.

#### **The DELETE\_ERROR Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## DROP\_APPLY Procedure

This procedure drops an apply component.

### Syntax

```
DBMS_APPLY_ADM.DROP_APPLY(
    apply_name          IN  VARCHAR2,
    drop_unused_rule_sets IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 21–9 DROP\_APPLY Procedure Parameters**

Parameter	Description
apply_name	The name of the apply component being dropped. You must specify an existing apply component name. Do not specify an owner.
drop_unused_rule_sets	<p>If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified apply component if these rule sets are not used by any other Oracle Streams or XStream component. These components include capture processes, propagations, apply processes, inbound servers, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set.</p> <p>If FALSE, then the procedure does not drop the rule sets used by the specified apply component, and the rule sets retain their rules.</p>

### Usage Notes

The following usage notes apply to this procedure:

- [The DROP\\_APPLY Procedure and Rules](#)
- [The DROP\\_APPLY Procedure and XStream Outbound Servers](#)
- [The DROP\\_APPLY Procedure and XStream Inbound Servers](#)

#### The DROP\_APPLY Procedure and Rules

When you use this procedure to drop an apply component, information about rules created for the apply component using the DBMS\_STREAMS\_ADM package is removed from the data dictionary views for rules. Information about such a rule is removed even if the rule is not in either the positive or negative rule set for the apply component. The following are the data dictionary views for rules:

- ALL\_STREAMS\_GLOBAL\_RULES
- DBA\_STREAMS\_GLOBAL\_RULES
- ALL\_STREAMS\_MESSAGE\_RULES
- DBA\_STREAMS\_MESSAGE\_RULES
- ALL\_STREAMS\_SCHEMA\_RULES
- DBA\_STREAMS\_SCHEMA\_RULES
- ALL\_STREAMS\_TABLE\_RULES

- DBA\_STREAMS\_TABLE\_RULES

**See Also:** *Oracle Streams Concepts and Administration* for more information about Oracle Streams data dictionary views

**The DROP\_APPLY Procedure and XStream Outbound Servers**

When the `DROP_APPLY` procedure is executed on an outbound server, it runs the `DROP_OUTBOUND` procedure in the `DBMS_XSTREAM_ADM` package. Therefore, it might also drop the outbound server's capture process and queue.

**The DROP\_APPLY Procedure and XStream Inbound Servers**

When the `DROP_APPLY` procedure is executed on an inbound server, it runs the `DROP_INBOUND` procedure in the `DBMS_XSTREAM_ADM` package. Therefore, it might also drop the inbound server's queue.



## DROP\_OBJECT\_DEPENDENCY Procedure

This procedure drops an object dependency. An object dependency is a virtual dependency definition that defines a parent-child relationship between two objects at a destination database.

---



---

### Note:

- An error is raised if an object dependency does not exist for the specified database objects.
  - An error is raised if NULL is specified for either of the procedure parameters.
- 
- 

### See Also:

- [CREATE\\_OBJECT\\_DEPENDENCY Procedure](#) on page 21-24
- *Oracle Streams Concepts and Administration*

## Syntax

```
DBMS_APPLY_ADM.DROP_OBJECT_DEPENDENCY (
  object_name          IN  VARCHAR2,
  parent_object_name  IN  VARCHAR2);
```

## Parameters

**Table 21–10 DROP\_OBJECT\_DEPENDENCY Procedure Parameters**

Parameter	Description
object_name	The name of the child database object, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.
parent_object_name	The name of the parent database object, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, hr.departments. If the schema is not specified, then the current user is the default.

## Usage Notes

The following usage notes apply to this procedure:

### The DROP\_OBJECT\_DEPENDENCY Procedure and XStream Outbound Servers

This procedure has no effect on XStream outbound servers.

### The DROP\_OBJECT\_DEPENDENCY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

## EXECUTE\_ALL\_ERRORS Procedure

This procedure reexecutes the error transactions in the error queue for the specified apply component.

The transactions are reexecuted in commit SCN order. Error reexecution stops if an error is raised.

**See Also:** *Oracle Streams Concepts and Administration* for more information about the error queue

### Syntax

```
DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS (
    apply_name      IN  VARCHAR2  DEFAULT NULL,
    execute_as_user IN  BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 21–11 EXECUTE\_ALL\_ERRORS Procedure Parameters**

Parameter	Description
apply_name	The name of the apply component that raised the errors while processing the transactions. Do not specify an owner.  If NULL, then all error transactions for all apply components are reexecuted.
execute_as_user	If TRUE, then the procedure reexecutes the transactions in the security context of the current user.  If FALSE, then the procedure reexecutes each transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The DBA_APPLY_ERROR data dictionary view lists the original receiver for each error transaction.  The user who executes the transactions must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply component.

### Usage Notes

The following usage notes apply to this procedure:

- [The EXECUTE\\_ALL\\_ERRORS Procedure and XStream Outbound Servers](#)
- [The EXECUTE\\_ALL\\_ERRORS Procedure and XStream Inbound Servers](#)

#### **The EXECUTE\_ALL\_ERRORS Procedure and XStream Outbound Servers**

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

#### **The EXECUTE\_ALL\_ERRORS Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## EXECUTE\_ERROR Procedure

This procedure reexecutes the specified error transaction in the error queue.

**See Also:** *Oracle Streams Concepts and Administration* for more information about the error queue

### Syntax

```
DBMS_APPLY_ADM.EXECUTE_ERROR(
  local_transaction_id IN VARCHAR2,
  execute_as_user      IN BOOLEAN   DEFAULT FALSE,
  user_procedure       IN VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 21–12 EXECUTE\_ERROR Procedure Parameters**

Parameter	Description
local_transaction_id	The identification number of the error transaction to execute. If the specified transaction does not exist in the error queue, then an error is raised.
execute_as_user	If TRUE, then the procedure reexecutes the transaction in the security context of the current user.  If FALSE, then the procedure reexecutes the transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised. The DBA_APPLY_ERROR data dictionary view lists the original receiver for each error transaction.  The user who executes the transaction must have privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. This user must also have dequeue privileges on the queue used by the apply component.
user_procedure	A user-defined procedure that modifies the error transaction so that it can be successfully executed.  Specify NULL to execute the error transaction without running a user procedure.  <b>See Also:</b> " <a href="#">Usage Notes</a> " on page 21-31 for more information about the user procedure

### Usage Notes

The following usage notes apply to this procedure:

- [The User Procedure](#)
- [The EXECUTE\\_ERROR Procedure and XStream Outbound Servers](#)
- [The EXECUTE\\_ERROR Procedure and XStream Inbound Servers](#)

#### The User Procedure

You must specify the full procedure name for the `user_procedure` parameter in one of the following forms:

- `[schema_name.]package_name.procedure_name`
- `[schema_name.]procedure_name`

If the procedure is in a package, then the *package\_name* must be specified. The user who invokes the EXECUTE\_ERROR procedure must have EXECUTE privilege on the specified procedure. Also, if the *schema\_name* is not specified, then the user who invokes the EXECUTE\_ERROR procedure is the default.

For example, suppose the *procedure\_name* has the following properties:

- *strmadmin* is the *schema\_name*.
- *fix\_errors* is the *package\_name*.
- *fix\_hr\_errors* is the *procedure\_name*.

In this case, specify the following:

```
strmadmin.fix_errors.fix_hr_errors
```

The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (
    in_anydata          IN      ANYDATA,
    error_record        IN      DBA_APPLY_ERROR%ROWTYPE,
    error_message_number IN      NUMBER,
    messaging_default_processing IN OUT BOOLEAN,
    out_anydata         OUT     ANYDATA);
```

The user procedure has the following parameters:

- *in\_anydata*: The ANYDATA encapsulation of a message that the apply component passes to the procedure. A single transaction can include multiple messages. A message can be a row logical change record (row LCR), a DDL logical change record (DDL LCR), or a user message.
- *error\_record*: The row in the DBA\_APPLY\_ERROR data dictionary view that identifies the transaction
- *error\_message\_number*: The message number of the ANYDATA object in the *in\_anydata* parameter, starting at 1
- *messaging\_default\_processing*: If TRUE, then the apply component continues processing the message in the *in\_anydata* parameter, which can include executing DML or DDL statements and invoking apply handlers.

If FALSE, then the apply component skips processing the message in the *in\_anydata* parameter and moves on to the next message in the *in\_anydata* parameter.

- *out\_anydata*: The ANYDATA object processed by the user procedure and used by the apply component if *messaging\_default\_processing* is TRUE.

If an LCR is executed using the EXECUTE LCR member procedure in the user procedure, then the LCR is executed directly, and the *messaging\_default\_processing* parameter should be set to FALSE. In this case, the LCR is not passed to any apply handlers.

Processing an error transaction with a user procedure results in one of the following outcomes:

- The user procedure modifies the transaction so that it can be executed successfully.
- The user procedure fails to make the necessary modifications, and an error is raised when transaction execution is attempted. In this case, the transaction is rolled back and remains in the error queue.

The following restrictions apply to the user procedure:

- Do not execute COMMIT or ROLLBACK statements. Doing so can endanger the consistency of the transaction.
- Do not modify LONG, LONG RAW or LOB column data in an LCR.
- If the ANYDATA object in the in\_anydata parameter is a row LCR, then the out\_anydata parameter must be row LCR if the messaging\_default\_processing parameter is set to TRUE.
- If the ANYDATA object in the in\_anydata parameter is a DDL LCR, then the out\_anydata parameter must be DDL LCR if the messaging\_default\_processing parameter is set to TRUE.
- The user who runs the user procedure must have SELECT privilege on the DBA\_APPLY\_ERROR data dictionary view.

---

**Note:** LCRs containing transactional directives, such as COMMIT and ROLLBACK, are not passed to the user procedure.

---

#### **The EXECUTE\_ERROR Procedure and XStream Outbound Servers**

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

#### **The EXECUTE\_ERROR Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## GET\_ERROR\_MESSAGE Function

This function returns the message payload from the error queue for the specified message number and transaction identifier. The message can be a logical change record (LCR) or a non-LCR message.

This function is overloaded. One version of this function contains two `OUT` parameters. These `OUT` parameters contain the destination queue into which the message should be enqueued, if one exists, and whether the message should be executed. The destination queue is specified using the `SET_ENQUEUE_DESTINATION` procedure, and the execution directive is specified using the `SET_EXECUTE` procedure.

### See Also:

- [SET\\_ENQUEUE\\_DESTINATION Procedure](#) on page 21-46
- [SET\\_EXECUTE Procedure](#) on page 21-48

## Syntax

```
DBMS_APPLY_ADM.GET_ERROR_MESSAGE (
    message_number          IN    NUMBER,
    local_transaction_id    IN    VARCHAR2,
    destination_queue_name  OUT   VARCHAR2,
    execute                 OUT   BOOLEAN)
RETURN ANYDATA;
```

```
DBMS_APPLY_ADM.GET_ERROR_MESSAGE (
    message_number          IN    NUMBER,
    local_transaction_id    IN    VARCHAR2)
RETURN ANYDATA;
```

## Parameters

**Table 21–13** GET\_ERROR\_MESSAGE Function Parameters

Parameter	Description
message_number	The identification number of the message. This number identifies the position of the message in the transaction. Query the <code>DBA_APPLY_ERROR</code> data dictionary view to view the message number of each apply error.
local_transaction_id	Identifier of the error transaction for which to return a message
destination_queue_name	Contains the name of the queue into which the message should be enqueued. If the message should not be enqueued into a queue, then this parameter contains <code>NULL</code> .
execute	Contains <code>TRUE</code> if the message should be executed Contains <code>FALSE</code> if the message should not be executed

## Usage Notes

The following usage notes apply to this procedure:

- [The GET\\_ERROR\\_MESSAGE Procedure and XStream Outbound Servers](#)
- [The GET\\_ERROR\\_MESSAGE Procedure and XStream Inbound Servers](#)

**The GET\_ERROR\_MESSAGE Procedure and XStream Outbound Servers**

Outbound servers do not enqueue error transactions into an error queue. This procedure cannot be used with XStream outbound servers.

**The GET\_ERROR\_MESSAGE Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## REMOVE\_STMT\_HANDLER

This procedure removes a statement DML handler for a specified operation on a specified database object from a single apply component or from all apply components in the database.

**See Also:**

- [Chapter 147, "DBMS\\_STREAMS\\_HANDLER\\_ADM"](#)
- *Oracle Streams Concepts and Administration*

### Syntax

```
DBMS_APPLY_ADM.REMOVE_STMT_HANDLER(
  object_name      IN  VARCHAR2,
  operation_name   IN  VARCHAR2,
  handler_name     IN  VARCHAR2,
  apply_name       IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 21–14 REMOVE\_STMT\_HANDLER Procedure Parameters**

Parameter	Description
object_name	The name of the source object specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, <i>hr.employees</i> . If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure.  If NULL, then the procedure raises an error.
operation_name	The name of the operation, which can be specified as: <ul style="list-style-type: none"> <li>▪ INSERT</li> <li>▪ UPDATE</li> <li>▪ DELETE</li> </ul> If NULL, then the procedure raises an error.
handler_name	The name of the statement DML handler.  If NULL, then the procedure raises an error.
apply_name	The name of the apply component that uses the statement DML handler.  If NULL, then the procedure removes the statement DML handler from all apply components in the database.

### Usage Notes

The following usage notes apply to this procedure:

- [The REMOVE\\_STMT\\_HANDLER Procedure and XStream Outbound Servers](#)
- [The REMOVE\\_STMT\\_HANDLER Procedure and XStream Inbound Servers](#)

**The REMOVE\_STMT\_HANDLER Procedure and XStream Outbound Servers**

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.



**The REMOVE\_STMT\_HANDLER Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## SET\_CHANGE\_HANDLER Procedure

This procedure sets or unsets a change handler that tracks changes for a specified operation on a specified database object for a single apply component.

A change handler is a special type of statement DML handler that tracks table changes and was created by either the `DBMS_STREAMS_ADM.MAINTAIN_CHANGE_TABLE` procedure or this `SET_CHANGE_HANDLER` procedure. Information about change handlers is stored in the `ALL_APPLY_CHANGE_HANDLERS` and `DBA_APPLY_CHANGE_HANDLERS` views.

This procedure automatically generates the statement that is added to the change handler based on values specified in the procedure parameters. You should only run this procedure when a configuration that tracks database changes exists.

---



---

**Note:** Use the [MAINTAIN\\_CHANGE\\_TABLE Procedure](#) to configure an environment that tracks table changes

---



---

### Syntax

```
DBMS_APPLY_ADM.SET_CHANGE_HANDLER (
  change_table_name  IN  VARCHAR2,
  source_table_name  IN  VARCHAR2,
  capture_values     IN  VARCHAR2,
  apply_name         IN  VARCHAR2,
  operation_name     IN  VARCHAR2,
  change_handler_name IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 21–15** SET\_CHANGE\_HANDLER Procedure Parameters

Parameter	Description
<code>change_table_name</code>	The table that records changes to the source table. Specify the table as <code>[schema_name.] table_name</code> . For example, <code>hr.jobs_change_table</code> . If the schema is not specified, then the current user is the default. If <code>NULL</code> , then the procedure raises an error.
<code>source_table_name</code>	The table at the source database for which changes are recorded. Specify the table as <code>[schema_name.] table_name</code> . For example, <code>hr.jobs</code> . If the schema is not specified, then the current user is the default. If <code>NULL</code> , then the procedure raises an error.

**Table 21–15 (Cont.) SET\_CHANGE\_HANDLER Procedure Parameters**

Parameter	Description
capture_values	<p>Specify which values to record in the change table when update operations are performed on the source table:</p> <ul style="list-style-type: none"> <li>■ old - To record the original values for an updated column in the source table</li> <li>■ new - To record the new values for an updated column in the source table</li> <li>■ * - To record both the original and the new values for an updated column in the source table</li> </ul> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> For insert operations, only new column values are captured. For delete operations, only old column values are captured.</p>
apply_name	<p>The name of the apply component that applies changes to the change table. Do not specify an owner.</p> <p>If NULL, then the procedure raises an error.</p>
operation_name	<p>The name of the operation, which can be specified as:</p> <ul style="list-style-type: none"> <li>■ INSERT</li> <li>■ UPDATE</li> <li>■ DELETE</li> </ul> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> Change handlers cannot be specified for LOB operations.</p>
change_handler_name	<p>The name of the change handler.</p> <p>If the specified change handler exists, then a statement is added to the existing handler. Ensure that the existing change handler is for the same operation on the same table as the settings for the <code>operation_name</code> and <code>source_table_name</code> parameters, respectively. If the existing handler is for a different operation or table, then an apply error results when the handler is invoked.</p> <p>If non-NULL and the specified change handler does not exist, then this procedure creates the change handler.</p> <p>If NULL and a change handler exists for the same operation on the same table as the settings for the <code>operation_name</code> and <code>source_table_name</code> parameters, respectively, then the existing change handler is removed.</p> <p>If NULL and the specified change handler does not exist, then the procedure raises an error.</p> <p><b>See Also:</b> "Usage Notes" on page 21-39 for more information about this parameter.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [Checking for an Existing Change Handler](#)
- [The SET\\_CHANGE\\_HANDLER Procedure and XStream Outbound Servers](#)
- [The SET\\_CHANGE\\_HANDLER Procedure and XStream Inbound Servers](#)

**Checking for an Existing Change Handler**

To check for an existing change handler for a specific operation on a specific source table, run the following query:

```
SELECT HANDLER_NAME, APPLY_NAME FROM DBA_APPLY_CHANGE_HANDLERS
WHERE operation_name      = 'operation'
   AND source_table_owner = 'source_table_owner'
   AND source_table_name  = 'source_table_name'
   AND change_table_owner = 'change_table_owner'
   AND change_table_name  = 'change_table_name';
```

where:

- *operation* is operation specified for the new handler, either INSERT, UPDATE, or DELETE
- *source\_table\_owner* is the owner of the source table
- *source\_table\_name* is the name of the source table
- *change\_table\_owner* is the owner of the change table
- *change\_table\_name* is the name of the change table

**The SET\_CHANGE\_HANDLER Procedure and XStream Outbound Servers**

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.

**The SET\_CHANGE\_HANDLER Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## SET\_DML\_HANDLER Procedure

This procedure sets or unsets a user procedure as a procedure DML handler for a specified operation on a specified database object for a single apply component or for all apply components in the database. The user procedure alters the apply behavior for the specified operation on the specified object.

### Syntax

```
DBMS_APPLY_ADM.SET_DML_HANDLER (
  object_name      IN  VARCHAR2,
  object_type      IN  VARCHAR2,
  operation_name   IN  VARCHAR2,
  error_handler    IN  BOOLEAN   DEFAULT FALSE,
  user_procedure   IN  VARCHAR2,
  apply_database_link IN  VARCHAR2 DEFAULT NULL,
  apply_name       IN  VARCHAR2 DEFAULT NULL,
  assemble_lob    IN  BOOLEAN   DEFAULT TRUE);
```

### Parameters

**Table 21–16 SET\_DML\_HANDLER Procedure Parameters**

Parameter	Description
object_name	The name of the source object specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, <i>hr.employees</i> . If the schema is not specified, then the current user is the default. The specified object does not need to exist when you run this procedure.
object_type	The type of the source object. Currently, <i>TABLE</i> is the only possible source object type.
operation_name	The name of the operation, which can be specified as: <ul style="list-style-type: none"> <li>▪ <i>INSERT</i></li> <li>▪ <i>UPDATE</i></li> <li>▪ <i>DELETE</i></li> <li>▪ <i>LOB_UPDATE</i></li> <li>▪ <i>DEFAULT</i></li> </ul> <p>For example, suppose you run this procedure twice for the <i>hr.employees</i> table. In one call, you set <i>operation_name</i> to <i>UPDATE</i> and <i>user_procedure</i> to <i>employees_update</i>. In another call, you set <i>operation_name</i> to <i>INSERT</i> and <i>user_procedure</i> to <i>employees_insert</i>. Both times, you set <i>error_handler</i> to <i>FALSE</i>. In this case, the <i>employees_update</i> procedure is run for <i>UPDATE</i> operations on the <i>hr.employees</i> table, and the <i>employees_insert</i> procedure is run for <i>INSERT</i> operations on the <i>hr.employees</i> table.</p> <p>Specify <i>DEFAULT</i> to set the procedure as the default procedure DML handler for the database object. In this case, the procedure DML handler is used for any <i>INSERT</i>, <i>UPDATE</i>, <i>DELETE</i>, and <i>LOB_WRITE</i> on the database object, if another procedure DML handler is not specifically set for the operation on the database object.</p>

**Table 21–16 (Cont.) SET\_DML\_HANDLER Procedure Parameters**

Parameter	Description
error_handler	<p>If TRUE, then the specified user procedure is run when a row logical change record (row LCR) involving the specified operation on the specified object raises an apply error. You can code the user procedure to resolve possible error conditions, notify administrators of the error, log the error, or any combination of these actions.</p> <p>If FALSE, then the handler being set is run for all row LCRs involving the specified operation on the specified object.</p>
user_procedure	<p>A user-defined procedure that is invoked during apply for the specified operation on the specified object. If the procedure is a procedure DML handler, then it is invoked instead of the default apply performed by Oracle. If the procedure is an error handler, then it is invoked when an apply error is encountered.</p> <p>Specify NULL to unset a procedure DML handler that is set for the specified operation on the specified object.</p>
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.
apply_name	<p>The name of the apply component that uses the procedure DML handler or error handler.</p> <p>If NULL, then the procedure sets the procedure DML handler or error handler as a general handler for all apply components in the database.</p> <p>If the user_procedure parameter is set to NULL to unset a handler, and the handler being unset is set for a specific apply component, then use the apply_name parameter to specify the apply component to unset the handler.</p>
assemble_lob	<p>If TRUE, then LOB assembly is used for LOB columns in LCRs processed by the handler. LOB assembly combines multiple LCRs for a LOB column resulting from a single row change into one row LCR before passing the LCR to the handler. Database compatibility must be 10.2.0 or higher to use LOB assembly.</p> <p>If FALSE, then LOB assembly is not used for LOB columns in LCRs processed by the handler.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [Run the SET\\_DML\\_HANDLER Procedure at the Destination Database](#)
- [Procedure DML Handlers and Error Handlers](#)
- [The apply\\_name Parameter](#)
- [Signature of a DML Handler Procedure or Error Handler Procedure](#)
- [LOB Assembly](#)
- [The SET\\_DML\\_HANDLER Procedure and XStream Outbound Servers](#)
- [The SET\\_DML\\_HANDLER Procedure and XStream Inbound Servers](#)

### Run the SET\_DML\_HANDLER Procedure at the Destination Database

Run this procedure at the destination database. The SET\_DML\_HANDLER procedure provides a way for users to apply logical change records containing DML changes (row LCRs) using a customized apply.

### Procedure DML Handlers and Error Handlers

If the `error_handler` parameter is set to `TRUE`, then it specifies that the user procedure is an error handler. An error handler is invoked only when a row LCR raises an apply error. Such an error can result from a data conflict if no conflict handler is specified or if the update conflict handler cannot resolve the conflict. If the `error_handler` parameter is set to `FALSE`, then the user procedure is a procedure DML handler, not an error handler, and a procedure DML handler is always run instead of performing the specified operation on the specified object.

This procedure either sets a procedure DML handler or an error handler for a particular operation on an object. It cannot set both a procedure DML handler and an error handler for the same object and operation.

---

**Note:** Currently, setting an error handler for an apply component that is applying changes to a non-Oracle database is not supported.

---

### The apply\_name Parameter

If the `apply_name` parameter is non-NULL, then the procedure DML handler or error handler is set for the specified apply component. In this case, this handler is not invoked for other apply components at the local destination database. If the `apply_name` parameter is NULL, the default, then the handler is set as a general handler for all apply components at the destination database. When a handler is set for a specific apply component, then this handler takes precedence over any general handlers. For example, consider the following scenario:

- A procedure DML handler named `handler_hr` is specified for an apply component named `apply_hr` for UPDATE operations on the `hr.employees` table.
- A general procedure DML handler named `handler_gen` also exists for UPDATE operations on the `hr.employees` table.

In this case, the `apply_hr` apply component uses the `handler_hr` procedure DML handler for UPDATE operations on the `hr.employees` table.

At the source database, you must specify an unconditional supplemental log group for the columns needed by a DML or error handler.

### Signature of a DML Handler Procedure or Error Handler Procedure

You can use the SET\_DML\_HANDLER procedure to set either a procedure DML handler or an error handler for row LCRs that perform a specified operation on a specified object. The signatures of a DML handler procedure and of an error handler procedure are described following this section.

In either case, you must specify the full procedure name for the `user_procedure` parameter in one of the following forms:

- `[schema_name.]package_name.procedure_name`
- `[schema_name.]procedure_name`

If the procedure is in a package, then the *package\_name* must be specified. The user who invokes the SET\_DML\_HANDLER procedure must have EXECUTE privilege on the specified procedure. Also, if the *schema\_name* is not specified, then the user who invokes the SET\_DML\_HANDLER procedure is the default.

For example, suppose the *procedure\_name* has the following properties:

- *hr* is the *schema\_name*.
- *apply\_pkg* is the *package\_name*.
- *employees\_default* is the *procedure\_name*.

In this case, specify the following:

```
hr.apply_pkg.employees_default
```

The following restrictions apply to the user procedure:

- Do not execute COMMIT or ROLLBACK statements. Doing so can endanger the consistency of the transaction that contains the LCR.
- If you are manipulating a row using the EXECUTE member procedure for the row LCR, then do not attempt to manipulate more than one row in a row operation. You must construct and execute manually any DML statements that manipulate more than one row.
- If the command type is UPDATE or DELETE, then row operations resubmitted using the EXECUTE member procedure for the LCR must include the entire key in the list of old values. The key is the primary key or the smallest unique index that has at least one NOT NULL column, unless a substitute key has been specified by the SET\_KEY\_COLUMNS procedure. If there is no specified key, then the key consists of all non LOB, non LONG, and non LONG RAW columns.
- If the command type is INSERT, then row operations resubmitted using the EXECUTE member procedure for the LCR should include the entire key in the list of new values. Otherwise, duplicate rows are possible. The key is the primary key or the smallest unique index that has at least one NOT NULL column, unless a substitute key has been specified by the SET\_KEY\_COLUMNS procedure. If there is no specified key, then the key consists of all of the table columns, except for columns of the following data types: LOB, LONG, LONG RAW, user-defined types (including object types, REFs, varrays, nested tables), and Oracle-supplied types (including Any types, XML types, spatial types, and media types).

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about and restrictions regarding procedure DML handlers and LOB, LONG, and LONG RAW data types

The procedure specified in the *user\_procedure* parameter must have the following signature:

```
PROCEDURE user_procedure (  
    parameter_name IN ANYDATA);
```

Here, *user\_procedure* stands for the name of the procedure and *parameter\_name* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is a ANYDATA encapsulation of a row LCR.

**See Also:** [Chapter 248, "Logical Change Record TYPES"](#) for more information about LCRs



The procedure you create for error handling must have the following signature:

```
PROCEDURE user_procedure (  
    message          IN  ANYDATA,  
    error_stack_depth IN  NUMBER,  
    error_numbers    IN  DBMS_UTILITY.NUMBER_ARRAY,  
    error_messages   IN  emsg_array);
```

If you want to retry the DML operation within the error handler, then have the error handler procedure run the EXECUTE member procedure for the LCR. The last error raised is on top of the error stack. To specify the error message at the top of the error stack, use `error_numbers(1)` and `error_messages(1)`.

---

---

**Note:**

- Each parameter is required and must have the specified data type. However, you can change the names of the parameters.
  - The `emsg_array` value must be a user-defined array that is a table of type `VARCHAR2` with at least 76 characters.
- 
- 

Running an error handler results in one of the following outcomes:

- The error handler successfully resolves the error and returns control to the apply component.
- The error handler fails to resolve the error, and the error is raised. The raised error causes the transaction to be rolled back and placed in the error queue.

**LOB Assembly**

Do not modify `LONG`, `LONG RAW`, or nonassembled LOB column data in an LCR with procedure DML handlers, error handlers, or custom rule-based transformation functions. Procedure DML handlers and error handlers can modify LOB columns in row LCRs that have been constructed by LOB assembly.

**The SET\_DML\_HANDLER Procedure and XStream Outbound Servers**

Outbound servers ignore all apply handlers. This procedure has no effect on XStream outbound servers.

**The SET\_DML\_HANDLER Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## SET\_ENQUEUE\_DESTINATION Procedure

This procedure sets the queue where the apply component automatically enqueues a message that satisfies the specified rule.

This procedure modifies the specified rule's action context to specify the queue. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. In this case, the client of the rules engine is an apply component. The information in an action context is an object of type `SYS.RE$NV_LIST`, which consists of a list of name-value pairs.

A queue destination specified by this procedure always consists of the following name-value pair in an action context:

- The name is `APPLY$_ENQUEUE`.
- The value is an ANYDATA instance containing the queue name specified as a VARCHAR2.

### Syntax

```
DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION(
    rule_name           IN VARCHAR2,
    destination_queue_name IN VARCHAR2);
```

### Parameters

**Table 21–17 SET\_ENQUEUE\_DESTINATION Procedure Parameters**

Parameter	Description
<code>rule_name</code>	The name of the rule, specified as <code>[schema_name.]rule_name</code> . For example, to specify a rule named <code>hr5</code> in the <code>hr</code> schema, enter <code>hr.hr5</code> for this parameter. If the schema is not specified, then the current user is the default.
<code>destination_queue_name</code>	<p>The name of the queue into which the apply component enqueues the message. Specify the queue in the form <code>[schema_name.]queue_name</code>. Only local queues can be specified.</p> <p>For example, to specify a queue in the <code>hr</code> schema named <code>streams_queue</code>, enter <code>hr.streams_queue</code>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then an existing name-value pair with the name <code>APPLY\$_ENQUEUE</code> is removed. If no name-value pair exists with the name <code>APPLY\$_ENQUEUE</code> for the rule, then no action is taken.</p> <p>If non-NULL and a name-value pair exists for the rule with the name <code>APPLY\$_ENQUEUE</code>, then it is removed, and a new name-value pair with the value specified by this parameter is added.</p>

### Usage Notes

The following usage notes apply to this procedure:

- [The SET\\_ENQUEUE\\_DESTINATION Procedure and Apply Handlers](#)
- [Considerations for the SET\\_ENQUEUE\\_DESTINATION Procedure](#)
- [The SET\\_ENQUEUE\\_DESTINATION Procedure and XStream Outbound Servers](#)

- [The SET\\_ENQUEUE\\_DESTINATION Procedure and XStream Inbound Servers](#)

### **The SET\_ENQUEUE\_DESTINATION Procedure and Apply Handlers**

If an apply handler, such as a procedure DML handler, DDL handler, or message handler, processes a message that also is enqueued into a destination queue, then the apply handler processes the message before it is enqueued.

### **Considerations for the SET\_ENQUEUE\_DESTINATION Procedure**

The following are considerations for using this procedure:

- This procedure does not verify that the specified queue exists. If the queue does not exist, then an error is raised when an apply component tries to enqueue a message into it.
- Oracle Streams capture processes, propagations, and messaging clients ignore the action context created by this procedure.
- The apply user of the apply component using the specified rule must have the necessary privileges to enqueue messages into the specified queue. If the queue is a secure queue, then the apply user must be a secure queue user of the queue.
- The specified rule must be in the positive rule set for an apply component. If the rule is in the negative rule set for an apply component, then the apply component does not enqueue the message into the destination queue.
- If the commit SCN for a message is less than or equal to the relevant instantiation SCN for the message, then the message is not enqueued into the destination queue, even if the message satisfies the apply component rule sets.

### **The SET\_ENQUEUE\_DESTINATION Procedure and XStream Outbound Servers**

This procedure has no effect on XStream outbound servers.

### **The SET\_ENQUEUE\_DESTINATION Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## SET\_EXECUTE Procedure

This procedure specifies whether a message that satisfies the specified rule is executed by an apply component.

This procedure modifies the specified rule's action context to specify message execution. A rule action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to `TRUE` for a message. In this case, the client of the rules engine is an apply component. The information in an action context is an object of type `SYS.RE$NV_LIST`, which consists of a list of name-value pairs.

A message execution directive specified by this procedure always consists of the following name-value pair in an action context:

- The name is `APPLY$_EXECUTE`.
- The value is an `ANYDATA` instance that contains `NO` as a `VARCHAR2`. When the value is `NO`, an apply component does not execute the message and does not send the message to any apply handler.

### Syntax

```
DBMS_APPLY_ADM.SET_EXECUTE(
  rule_name IN VARCHAR2,
  execute   IN BOOLEAN);
```

### Parameters

**Table 21–18 SET\_EXECUTE Procedure Parameters**

Parameter	Description
<code>rule_name</code>	The name of the rule, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . For example, to specify a rule named <code>hr5</code> in the <code>hr</code> schema, enter <code>hr.hr5</code> for this parameter. If the schema is not specified, then the current user is the default.
<code>execute</code>	<p>If <code>TRUE</code>, then the procedure removes the name-value pair with the name <code>APPLY\$_EXECUTE</code> for the specified rule. Removing the name-value pair means that the apply component executes messages that satisfy the rule. If no name-value pair with name <code>APPLY\$_EXECUTE</code> exists for the rule, then no action is taken.</p> <p>If <code>FALSE</code>, then the procedure adds a name-value pair to the rule's action context. The name is <code>APPLY\$_EXECUTE</code> and the value is <code>NO</code>. An apply component does not execute a message that satisfies the rule and does not send the message to any apply handler. If a name-value pair exists for the rule with the name <code>APPLY\$_EXECUTE</code>, then it is removed, and a new one with the value <code>NO</code> is added.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p>

### Usage Notes

The following usage notes apply to this procedure:

- [Considerations for the SET\\_EXECUTE Procedure](#)
- [The SET\\_EXECUTE Procedure and XStream Outbound Servers](#)
- [The SET\\_EXECUTE Procedure and XStream Inbound Servers](#)

**Considerations for the SET\_EXECUTE Procedure**

The following are considerations for using this procedure:

- If the message is a logical change record (LCR) and the message is not executed, then the change encapsulated in the LCR is not made to the relevant local database object. Also, if the message is not executed, then it is not sent to any apply handler.
- Oracle Streams capture processes, propagations, and messaging clients ignore the action context created by this procedure.
- The specified rule must be in the positive rule set for an apply component for the apply component to follow the execution directive. If the rule is in the negative rule set for an apply component, then the apply component ignores the execution directive for the rule.

**The SET\_EXECUTE Procedure and XStream Outbound Servers**

This procedure has no effect on XStream outbound servers.

**The SET\_EXECUTE Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## SET\_GLOBAL\_INSTANTIATION\_SCN Procedure

This procedure records the specified instantiation SCN for the specified source database and, optionally, for the schemas at the source database and the tables owned by these schemas. This procedure overwrites any existing instantiation SCN for the database, and, if it sets the instantiation SCN for a schema or a table, then it overwrites any existing instantiation SCN for the schema or table.

This procedure gives you precise control over which DDL logical change records (DDL LCRs) from a source database are ignored and which DDL LCRs are applied by an apply component.

### Syntax

```
DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
  source_database_name  IN  VARCHAR2,
  instantiation_scn     IN  NUMBER,
  apply_database_link   IN  VARCHAR2  DEFAULT NULL,
  recursive             IN  BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 21–19 SET\_GLOBAL\_INSTANTIATION\_SCN Procedure Parameters**

Parameter	Description
source_database_name	The global name of the source database. For example, DBS1.NET.  If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1.NET automatically.
instantiation_scn	The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source database from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.
recursive	If TRUE, then the procedure sets the instantiation SCN for the source database, all schemas in the source database, and all tables owned by the schemas in the source database. This procedure selects the schemas and tables from the ALL_USERS and ALL_TABLES data dictionary views, respectively, at the source database under the security context of the current user.  If FALSE, then the procedure sets the global instantiation SCN for the source database, but does not set the instantiation SCN for any schemas or tables.  <b>Note:</b> If recursive is set to TRUE, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the ALL_TABLES or DBA_TABLES data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.

## Usage Notes

The following usage notes apply to this procedure:

- [Instantiation SCNs and DDL LCRs](#)
- [The recursive Parameter](#)
- [Considerations for the SET\\_GLOBAL\\_INSTANTIATION\\_SCN Procedure](#)
- [The SET\\_GLOBAL\\_INSTANTIATION\\_SCN Procedure and XStream Outbound Servers](#)
- [The SET\\_GLOBAL\\_INSTANTIATION\\_SCN Procedure and XStream Inbound Servers](#)

### See Also:

- [SET\\_SCHEMA\\_INSTANTIATION\\_SCN Procedure](#) on page 21-66
- [SET\\_TABLE\\_INSTANTIATION\\_SCN Procedure](#) on page 21-69
- [LCR\\$\\_DDL\\_RECORD Type](#) on page 248-6 for more information about DDL LCRs
- *Oracle Streams Replication Administrator's Guide*

### Instantiation SCNs and DDL LCRs

If the commit SCN of a DDL LCR for a database object from a source database is less than or equal to the instantiation SCN for that source database at a destination database, then the apply component at the destination database disregards the DDL LCR. Otherwise, the apply component applies the DDL LCR.

The global instantiation SCN specified by this procedure is used for a DDL LCR only if the DDL LCR does not have `object_owner`, `base_table_owner`, and `base_table_name` specified. For example, the global instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of CREATE USER.

### The recursive Parameter

If the `recursive` parameter is set to TRUE, then this procedure sets the instantiation SCN for each schema at a source database and for the tables owned by these schemas. This procedure uses the `SET_SCHEMA_INSTANTIATION_SCN` procedure to set the instantiation SCN for each schema, and it uses the `SET_TABLE_INSTANTIATION_SCN` procedure to set the instantiation SCN for each table. Each schema instantiation SCN is used for DDL LCRs on the schema, and each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the `recursive` parameter is set to FALSE, then this procedure does not set the instantiation SCN for any schemas or tables.

### Considerations for the SET\_GLOBAL\_INSTANTIATION\_SCN Procedure

The following are considerations for using this procedure:

- Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.
- The instantiation SCN is not set for the SYS or SYSTEM schemas.

**The SET\_GLOBAL\_INSTANTIATION\_SCN Procedure and XStream Outbound Servers**

Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

**See Also:** *Oracle Database XStream Guide* for more information about outbound servers and instantiation SCNs

**The SET\_GLOBAL\_INSTANTIATION\_SCN Procedure and XStream Inbound Servers**

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.



## SET\_KEY\_COLUMNS Procedures

This procedure records the set of columns to be used as the substitute primary key for apply purposes and removes existing substitute primary key columns for the specified object if they exist.

This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

### Syntax

```
DBMS_APPLY_ADM.SET_KEY_COLUMNS (
  object_name          IN  VARCHAR2,
  column_list         IN  VARCHAR2,
  apply_database_link IN  VARCHAR2 DEFAULT NULL);

DBMS_APPLY_ADM.SET_KEY_COLUMNS (
  object_name          IN  VARCHAR2,
  column_table        IN  DBMS_UTILITY.NAME_ARRAY,
  apply_database_link IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 21–20 SET\_KEY\_COLUMNS Procedure Parameters**

Parameter	Description
<code>object_name</code>	The name of the table specified as <code>[schema_name.] object_name</code> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default. If the apply component is applying changes to a non-Oracle database in a heterogeneous environment, then the object name is not verified.
<code>column_list</code>	A comma-delimited list of the columns in the table to use as the substitute primary key, with no spaces between the column names.  If the <code>column_list</code> parameter is empty or NULL, then the current set of key columns is removed.
<code>column_table</code>	A PL/SQL associative array of type <code>DBMS_UTILITY.NAME_ARRAY</code> of the columns in the table to use as the substitute primary key. The index for <code>column_table</code> must be 1-based, increasing, dense, and terminated by a NULL.  If the <code>column_table</code> parameter is empty or NULL, then the current set of key columns is removed.
<code>apply_database_link</code>	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.

### Usage Notes

The following usage notes apply to this procedure:

- [Considerations for the SET\\_KEY\\_COLUMNS Procedure](#)
- [Duplicate Rows and Substitute Primary Key Columns](#)
- [The SET\\_KEY\\_COLUMNS Procedure and XStream Outbound Servers](#)
- [The SET\\_KEY\\_COLUMNS Procedure and XStream Inbound Servers](#)

### Considerations for the SET\_KEY\_COLUMNS Procedure

The following are considerations for using this procedure:

- When not empty, the specified set of columns takes precedence over any primary key for the specified object. Do not specify substitute key columns if the object has primary key columns and you want to use those primary key columns as the key.
- Run this procedure at the destination database. At the source database, you must specify an unconditional supplemental log group for the substitute key columns.
- Unlike true primary keys, columns specified as substitute key column columns can contain NULLs. However, Oracle recommends that each column you specify as a substitute key column be a NOT NULL column. You also should create a single index that includes all of the columns in a substitute key. Following these guidelines improves performance for updates, deletes, and piecewise updates to LOBs because Oracle can locate the relevant row more efficiently.
- Do not permit applications to update the primary key or substitute key columns of a table. This ensures that Oracle can identify rows and preserve the integrity of the data.
- If there is neither a primary key, nor a unique index that has at least one NOT NULL column, nor a substitute key for a table, then the key consists of all of the table columns, except for columns of the following data types: LOB, LONG, LONG RAW, user-defined types (including object types, REFS, varrays, nested tables), and Oracle-supplied types (including Any types, XML types, spatial types, and media types).

### Duplicate Rows and Substitute Primary Key Columns

A table has duplicate rows when all of the column values are identical for two or more rows in the table, excluding LOB, LONG, and LONG RAW columns. You can specify substitute primary key columns for a table at a destination database using by the SET\_KEY\_COLUMNS procedure. When substitute primary key columns are specified for a table with duplicate rows at a destination database, and the `allow_duplicate_rows` apply component parameter is set to Y, meet the following requirements to keep the table data synchronized at the source and destination databases:

- Ensure that supplemental logging is specified at source database for the columns specified as substitute key columns at the destination database. The substitute key columns must be in an unconditional log group at the source database.
- Ensure that the substitute key columns uniquely identify each row in the table at the destination database.

The rest of this section provides more details about these requirements.

When there is no key for a table and the `allow_duplicate_rows` apply component parameter is set to Y, a single row LCR with an UPDATE or DELETE command type only is applied to one of the duplicate rows. In this case, if the table at the source database and the table at the destination database have corresponding duplicate rows, then a change that changes all of the duplicate rows at the source database also changes all the duplicate rows at the destination database when the row LCRs resulting from the change are applied.

For example, suppose a table at a source database has two duplicate rows. An update is performed on the duplicate rows, resulting in two row LCRs. At the destination database, one row LCR is applied to one of the duplicate rows. At this point, the rows are no longer duplicate at the destination database because one of the rows has changed. When the second row LCR is applied at the destination database, the rows are duplicate again. Similarly, if a delete is performed on these duplicate rows at the

source database, then both rows are deleted at the destination database when the row LCRs resulting from the source change are applied.

When substitute primary key columns are specified for a table, row LCRs are identified with rows in the table during apply using the substitute primary key columns. If substitute primary key columns are specified for a table with duplicate rows at a destination database, and the `allow_duplicate_rows` apply component parameter is set to `Y`, then an update performed on duplicate rows at the source database can result in different changes when the row LCRs are applied at the destination database. Specifically, if the update does not change one of the columns specified as a substitute primary key column, then the same duplicate row can be updated multiple times at the destination database, while other duplicate rows might not be updated.

Also, if the substitute key columns do not identify each row in the table at the destination database uniquely, then a row LCR identified with multiple rows can update any one of the rows. In this case, the update in the row LCR might not be applied to the correct row in the table at the destination database.

An apply component ignores substitute primary key columns when it determines whether rows in a table are duplicates. An apply component determines that rows are duplicates only if all of the column values in the rows are identical (excluding LOB, LONG, and LONG RAW columns). Therefore, an apply component always raises an error if a single update or delete changes two or more nonduplicate rows in a table.

For example, consider a table with columns `c1`, `c2`, and `c3` on which the `SET_KEY_COLUMNS` procedure is used to designate column `c1` as the substitute primary key. If two rows have the same key value for the `c1` column, but different value for the `c2` or `c3` columns, then an apply component does not treat the rows as duplicates. If an update or delete modifies more than one row because the `c1` values in the rows are the same, then the apply component raises an error regardless of the setting for the `allow_duplicate_rows` apply component parameter.

**See Also:** [SET\\_PARAMETER Procedure](#) on page 21-56 for more information about the `allow_duplicate_rows` apply component parameter

### **The SET\_KEY\_COLUMNS Procedure and XStream Outbound Servers**

This procedure has no effect on XStream outbound servers.

### **The SET\_KEY\_COLUMNS Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

## SET\_PARAMETER Procedure

This procedure sets an apply parameter to the specified value.

### Syntax

```
DBMS_APPLY_ADM.SET_PARAMETER (
    apply_name IN VARCHAR2,
    parameter  IN VARCHAR2,
    value      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 21–21 SET\_PARAMETER Procedure Parameters**

Parameter	Description
apply_name	The apply component name. Do not specify an owner.
parameter	The name of the parameter you are setting. See " <a href="#">Apply Component Parameters</a> " on page 21-56 for a list of these parameters.
value	The value to which the parameter is set. If NULL, then the parameter is set to its default value.

### Apply Component Parameters

The following table lists the parameters for an apply component.

---



---

**Note:** Starting with Oracle Database 11g Release 2 (11.2.0.2), this subprogram includes the following new parameters: apply\_sequence\_nextval, compare\_key\_only, grouptransops, ignore\_transaction, and max\_sga\_size.

---



---

**Table 21–22 Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
allow_duplicate_rows	Y or N	N	<p>If Y and more than one row is changed by a single row logical change record (row LCR) with an UPDATE or DELETE command type, then the apply component only updates or deletes one of the rows.</p> <p>If N, then the apply component raises an error when it encounters a single row LCR with an UPDATE or DELETE command type that changes more than one row in a table.</p> <p><b>Note:</b> Regardless of the setting for this parameter, apply components do not allow changes to duplicate rows for tables with LOB, LONG, or LONG RAW columns.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64 and "Duplicate Rows and Substitute Primary Key Columns" on page 21-54</p>
apply_sequence_nextval	Y or N	N for apply processes Y for XStream outbound servers and XStream inbound servers	<p>Controls whether the apply component checks and adjusts sequence values.</p> <p>If Y, then the apply component checks and adjusts sequence values.</p> <p>For ascending sequences, setting this parameter to Y ensures that the destination sequence values are equal to or greater than the source sequence values.</p> <p>For descending sequences, setting this parameter to Y ensures that the destination sequence values are equal to or less than the source sequence values.</p> <p>If N, then the apply component does not check or adjust sequence values.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not set this parameter to Y for an apply process in an Oracle Streams replication environment unless XStream capabilities are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See <i>Oracle Database XStream Guide</i> for information about enabling XStream capabilities.</p> <p><b>See Also:</b> <a href="#">SET_PARAMETER Procedure</a> on page 32-39 for information about the capture_sequence_nextval capture process parameter</p>

**Table 21–22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
commit_serialization	DEPENDENT_TRANSACTION or FULL	DEPENDENT_TRANSACTION	<p>The order in which applied transactions are committed.</p> <p>Apply servers can apply nondependent transactions at the destination database in an order that is different from the commit order at the source database. Dependent transactions are always applied at the destination database in the same order as they were committed at the source database.</p> <p>You control whether the apply servers can apply nondependent transactions in a different order at the destination database using the <code>commit_serialization</code> apply parameter. This parameter has the following settings:</p> <ul style="list-style-type: none"> <li>▪ <code>DEPENDENT_TRANSACTION</code> - The apply component can commit nondependent transactions in any order. Performance is best if you specify <code>DEPENDENT_TRANSACTION</code>.</li> <li>▪ <code>FULL</code> - The apply component commits applied transactions in the order in which they were committed at the source database.</li> </ul> <p>Regardless of the specification, applied transactions can execute in parallel subject to data dependencies and constraint dependencies.</p> <p>If you specify <code>DEPENDENT_TRANSACTION</code>, then a destination database might commit changes in a different order than the source database. For example, suppose two nondependent transactions are committed at the source database in the following order:</p> <ol style="list-style-type: none"> <li>1. Transaction A</li> <li>2. Transaction B</li> </ol> <p>At the destination database, these transactions might be committed in the opposite order:</p> <ol style="list-style-type: none"> <li>1. Transaction B</li> <li>2. Transaction A</li> </ol> <p>If you specify <code>DEPENDENT_TRANSACTION</code> and there are application constraints that are not enforced by the database, then use virtual dependency definitions or add <code>RELY</code> constraints to account for the application constraints. See <i>Oracle Streams Concepts and Administration</i> for information about virtual dependency definitions and <i>Oracle Database Data Warehousing Guide</i> for information about <code>RELY</code> constraints.</p> <p><b>Note:</b> The <code>NONE</code> value is deprecated for this parameter. It is replaced by the <code>DEPENDENT_TRANSACTION</code> value.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>

**Table 21–22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
compare_key_only	Y or N	N for apply processes Y for XStream inbound servers	<p>If Y, then disables automatic conflict detection and only uses primary and unique key columns to identify the table row for a row LCR.</p> <p>If N, then enables automatic conflict detection and uses all of the old values in a row LCR to identify the table row for a row LCR.</p> <p><b>Note:</b> The COMPARE_OLD_VALUES procedure in this package can disable comparison of old values for specified columns during apply. See <a href="#">COMPARE_OLD_VALUES Procedure</a> on page 21-16.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64 and <i>Oracle Streams Replication Administrator's Guide</i> for information about automatic conflict detection</p>
disable_on_error	Y or N	Y	<p>If Y, then the apply component is disabled on the first unresolved error, even if the error is not irrecoverable.</p> <p>If N, then the apply component continues regardless of unresolved errors.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
disable_on_limit	Y or N	N	<p>If Y, then the apply component is disabled if the apply component terminates because it reached a value specified by the time_limit parameter or transaction_limit parameter.</p> <p>If N, then the apply component is restarted immediately after stopping because it reached a limit.</p> <p>When an apply component is restarted, it gets a new session identifier, and the processes associated with the apply component also get new session identifiers. However, the coordinator process number (APnn) remains the same.</p>
grouptransops	A positive integer from 1 to 10000	250 for apply processes and XStream inbound servers 10000 for XStream outbound servers	<p>The minimum number of LCRs that can be grouped into a single transaction. The commit LCR for a transaction is not included in the LCR count for the transaction.</p> <p>This parameter enables an apply component to group LCRs from multiple transactions into a single transaction. The apply component groups only LCRs that are part of committed transactions.</p> <p>If a transaction has more LCRs than the setting for this parameter, then the transaction is applied as a single transaction. The apply component does not split a transaction into separate transactions.</p> <p>This parameter only takes effect if the parallelism parameter setting is 1. The grouptransops parameter is ignored if the parallelism parameter setting is greater than 1.</p> <p><b>Note:</b> This parameter is intended for XStream outbound servers and inbound servers. An Oracle Streams apply process ignores this parameter unless XStream capabilities are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See <i>Oracle Database XStream Guide</i> for information about enabling XStream capabilities.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>

**Table 21–22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
ignore_transaction	A valid source transaction ID or NULL	NULL	<p>Instructs the apply component to ignore the specified transaction from the source database, effective immediately.</p> <p>Use caution when setting this parameter because ignoring a transaction might lead to data divergence between the source database and destination database.</p> <p>To ignore multiple transactions, specify each transaction in a separate call to the SET_PARAMETER procedure. The DBA_APPLY_PARAMETERS view displays a comma-delimited list of all transactions to be ignored. To clear the list of ignored transactions, run the SET_PARAMETER procedure and specify NULL for the ignore_transaction parameter.</p> <p>If NULL, then the apply component ignores this parameter.</p> <p><b>Note:</b> An apply component ignores this parameter for transactions that were not captured by an Oracle Streams capture process.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
max_sga_size	A positive integer	INFINITE	<p>Controls the amount of system global area (SGA) memory allocated specifically to the apply component, in megabytes.</p> <p>The memory is allocated for the duration of the apply component's session and is released when the apply component becomes disabled.</p> <p><b>Note:</b> The sum of SGA memory allocated for all components on a database must be less than the value set for the STREAMS_POOL_SIZE initialization parameter.</p> <p>If NULL, then the apply component uses the original default value. A NULL value has the same effect as resetting the parameter to its default value.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS procedure. See <i>Oracle Database XStream Guide</i> for information about enabling XStream capabilities.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
maximum_scn	A valid SCN or INFINITE	INFINITE	<p>The apply component is disabled before applying a transaction with a commit SCN greater than or equal to the value specified.</p> <p>If INFINITE, then the apply component runs regardless of the SCN value.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>



**Table 21–22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
parallelism	A positive integer	4	<p>The number of apply servers that can concurrently apply transactions.</p> <p>The reader server and the apply server process names are <i>ASnn</i>, where <i>nn</i> can include letters and numbers. The total number of <i>ASnn</i> processes is the value of the <code>parallelism</code> parameter plus one.</p> <p>For example, if <code>parallelism</code> is set to 4, then an apply component uses a total of five <i>ASnn</i> processes. In this case, there is one reader server and four apply servers.</p> <p>Setting the <code>parallelism</code> parameter to a number higher than the number of available operating system user processes can disable the apply component. Make sure the <code>PROCESSES</code> initialization parameter is set appropriately when you set the <code>parallelism</code> parameter.</p> <p><b>Note:</b> When the value of this parameter is changed from 1 to a higher value for a running apply component, the apply component is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied. When the value of this parameter is greater than 1, and the parameter value is decreased or increased, the apply component does not restart.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
preserve_encryption	Y or N	Y	<p>Whether to preserve encryption for columns encrypted using transparent data encryption.</p> <p>If <code>Y</code>, then columns in tables at the destination database must be encrypted when corresponding columns in row LCRs are encrypted. If columns are encrypted in row LCRs but the corresponding columns are not encrypted in the tables at the destination database, then an error is raised when the apply component tries to apply the row LCRs.</p> <p>If <code>N</code>, then columns in tables at the destination database do not need to be encrypted when corresponding columns in row LCRs are encrypted. If columns are encrypted in row LCRs but the corresponding columns are not encrypted in the tables at the destination database, then the apply component applies the changes in the row LCRs.</p> <p><b>Note:</b> When the value of this parameter is changed for a running apply component, the apply component is stopped and restarted automatically. This can take some time depending on the size of the transactions currently being applied.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>

**Table 21–22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
rtrim_on_implicit_conversion	Y or N	Y	<p>Whether to remove blank padding from the right end of a column when automatic data type conversion is performed during apply.</p> <p>If Y, then blank padding is removed when a CHAR or NCHAR source column in a row LCR is converted to a VARCHAR2, NVARCHAR2, or CLOB column in a table.</p> <p>If N, then blank padding is preserved in the column.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64 and <i>Oracle Streams Concepts and Administration</i> for information about automatic data type conversion during apply</p>
startup_seconds	0, a positive integer, or INFINITE	0	<p>The maximum number of seconds to wait for another instantiation of the same apply component to finish. If the other instantiation of the same apply component does not finish within this time, then the apply component does not start.</p> <p>If INFINITE, then an apply component does not start until another instantiation of the same apply component finishes.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
time_limit	A positive integer or INFINITE	INFINITE	<p>The apply component stops as soon as possible after the specified number of seconds since it started.</p> <p>If INFINITE, then the apply component continues to run until it is stopped explicitly.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
trace_level	0 or a positive integer	0	<p>Set this parameter only under the guidance of Oracle Support Services.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
transaction_limit	A positive integer or INFINITE	INFINITE	<p>The apply component stops after applying the specified number of transactions.</p> <p>If INFINITE, then the apply component continues to run regardless of the number of transactions applied.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>

**Table 21-22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
txn_age_spill_threshold	A positive integer or INFINITE	900	<p>The apply component begins to spill messages from memory to hard disk for a particular transaction when the amount of time that any message in the transaction has been in memory exceeds the specified number. The parameter specifies the age in seconds.</p> <p>When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table.</p> <p>Message spilling occurs at the transaction level. For example, if this parameter is set to 900, and the reader server of an apply component detects that one message in a transaction has been in memory longer than 900 seconds, then all of the messages in the transaction spill from memory to hard disk.</p> <p>If INFINITE, then the apply component does not spill messages to the hard disk based on the age of the messages.</p> <p>Query the DBA_APPLY_SPILL_TXN data dictionary view for information about transactions spilled by an apply component.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>

**Table 21–22 (Cont.) Apply Component Parameters**

Parameter Name	Possible Values	Default	Description
txn_lcr_spill_threshold	A positive integer or INFINITE	10000	<p>The apply component begins to spill messages from memory to hard disk for a particular transaction when the number of messages in memory for the transaction exceeds the specified number. The number of messages in first chunk of messages spilled from memory equals the number specified for this parameter, and the number of messages spilled in future chunks is either 100 or the number specified for this parameter, whichever is less.</p> <p>If the reader server of an apply component has the specified number of messages in memory for a particular transaction, then when it detects the next message for this transaction, it spills the messages that are in memory to the hard disk. For example, if this parameter is set to 10000, and a transaction has 10,200 messages, then the reader server handles the transaction in the following way:</p> <ol style="list-style-type: none"> <li>1. Reads the first 10,000 messages in the transaction into memory</li> <li>2. Spills messages 1 - 10,000 to hard disk when it detects message 10,000</li> <li>3. Reads the next 100 messages in the transaction into memory</li> <li>4. Spills messages 10,001 - 10,100 to hard disk when it detects message 10,100</li> <li>5. Reads the next 100 messages in the transaction into memory</li> </ol> <p>The apply component applies the first 10,100 messages from the hard disk and the last 100 messages from memory.</p> <p>When the reader server spills messages from memory, the messages are stored in a database table on the hard disk. These messages are not spilled from memory to a queue table.</p> <p>Message spilling occurs at the transaction level. For example, if this parameter is set to 10000, and the reader server of an apply component is assembling two transactions, one with 7,500 messages and another with 8,000 messages, then it does not spill any messages.</p> <p>If INFINITE, then the apply component does not spill messages to the hard disk based on the number of messages in a transaction.</p> <p>Query the DBA_APPLY_SPILL_TXN data dictionary view for information about transactions spilled by an apply component.</p> <p><b>See Also:</b> "Usage Notes" on page 21-64</p>
write_alert_log	Y or N	Y	<p>If Y, then the apply component writes a message to the alert log on exit.</p> <p>If N, then the apply component does not write a message to the alert log on exit.</p> <p>The message specifies the reason why the apply component stopped.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [Delays Are Possible Before New Parameter Settings Take Effect](#)

- [Parameters Interpreted as Positive Integers](#)
- [Parameters with a System Change Number \(SCN\) Setting](#)
- [The SET\\_PARAMETER Procedure and XStream Outbound Servers](#)
- [The SET\\_PARAMETER Procedure and XStream Inbound Servers](#)

### **Delays Are Possible Before New Parameter Settings Take Effect**

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

### **Parameters Interpreted as Positive Integers**

For all parameters that are interpreted as positive integers, the maximum possible value is 4,294,967,295. Where applicable, specify `INFINITE` for larger values.

### **Parameters with a System Change Number (SCN) Setting**

For parameters that require an SCN setting, any valid SCN value can be specified.

### **The SET\_PARAMETER Procedure and XStream Outbound Servers**

Outbound servers ignore the settings for the following apply parameters:

- `allow_duplicate_rows`
- `commit_serialization`
- `compare_key_only`
- `disable_on_error`
- `parallelism`
- `preserve_encryption`
- `rtrim_on_implicit_conversion`

The `commit_serialization` parameter is always set to `FULL` for an outbound server, and the `parallelism` parameter is always set to 1 for an outbound server.

You can use the other apply parameters with outbound servers.

---

---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

---

### **The SET\_PARAMETER Procedure and XStream Inbound Servers**

Inbound servers ignore the settings for the `ignore_transaction` and `maximum_scn` apply component parameters. You can use all of the other apply component parameters with inbound servers.

The default setting for the `compare_key_only` parameter for an inbound server is `Y`.

The default setting for the `parallelism` parameter for an inbound server is 4.

---

---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

---

## SET\_SCHEMA\_INSTANTIATION\_SCN Procedure

This procedure records the specified instantiation SCN for the specified schema in the specified source database and, optionally, for the tables owned by the schema at the source database. This procedure overwrites any existing instantiation SCN for the schema, and, if it sets the instantiation SCN for a table, it overwrites any existing instantiation SCN for the table.

This procedure gives you precise control over which DDL logical change records (LCRs) for a schema are ignored and which DDL LCRs are applied by an apply component.

### Syntax

```
DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN(
  source_schema_name  IN  VARCHAR2,
  source_database_name IN  VARCHAR2,
  instantiation_scn   IN  NUMBER,
  apply_database_link IN  VARCHAR2  DEFAULT NULL,
  recursive           IN  BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 21–23 SET\_SCHEMA\_INSTANTIATION\_SCN Procedure Parameters**

Parameter	Description
source_schema_name	The name of the source schema. For example, hr.  When setting an instantiation SCN for schema, always specify the name of the schema at the source database, even if a rule-based transformation or apply handler is configured to change the schema name.
source_database_name	The global name of the source database. For example, DBS1.NET.  If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1.NET automatically.
instantiation_scn	The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source schema from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.

**Table 21-23 (Cont.) SET\_SCHEMA\_INSTANTIATION\_SCN Procedure Parameters**

Parameter	Description
<code>recursive</code>	<p>If <code>TRUE</code>, then the procedure sets the instantiation SCN for the specified schema and all tables owned by the schema in the source database. This procedure selects the tables owned by the specified schema from the <code>ALL_TABLES</code> data dictionary view at the source database under the security context of the current user.</p> <p>If <code>FALSE</code>, then the procedure sets the instantiation SCN for specified schema, but does not set the instantiation SCN for any tables</p> <p><b>Note:</b> If <code>recursive</code> is set to <code>TRUE</code>, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the current user. Also, a table must be accessible to the current user in either the <code>ALL_TABLES</code> or <code>DBA_TABLES</code> data dictionary view at the source database for this procedure to set the instantiation SCN for the table at the destination database.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [The SET\\_SCHEMA\\_INSTANTIATION\\_SCN Procedure and LCRs](#)
- [Instantiation SCNs and DDL LCRs](#)
- [The recursive Parameter](#)
- [The SET\\_SCHEMA\\_INSTANTIATION\\_SCN Procedure and XStream Outbound Servers](#)
- [The SET\\_SCHEMA\\_INSTANTIATION\\_SCN Procedure and XStream Inbound Servers](#)

### See Also:

- [SET\\_GLOBAL\\_INSTANTIATION\\_SCN Procedure](#) on page 21-50
- [SET\\_TABLE\\_INSTANTIATION\\_SCN Procedure](#) on page 21-69
- [LCR\\$\\_DDL\\_RECORD Type](#) on page 248-6 for more information about DDL LCRs
- *Oracle Streams Replication Administrator's Guide*

### The SET\_SCHEMA\_INSTANTIATION\_SCN Procedure and LCRs

Any instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

### Instantiation SCNs and DDL LCRs

If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at a destination database, then the apply component at the destination database disregards the DDL LCR. Otherwise, the apply component applies the DDL LCR.

The schema instantiation SCN specified by this procedure is used on the following types of DDL LCRs:

- DDL LCRs with a `command_type` of `CREATE TABLE`
- DDL LCRs with a non-NULL `object_owner` specified and neither `base_table_owner` nor `base_table_name` specified.

For example, the schema instantiation SCN set by this procedure is used for a DDL LCR with a `command_type` of `CREATE TABLE` and `ALTER USER`.

The schema instantiation SCN specified by this procedure is not used for DDL LCRs with a `command_type` of `CREATE USER`. A global instantiation SCN is needed for such DDL LCRs.

### The recursive Parameter

If the `recursive` parameter is set to `TRUE`, then this procedure sets the table instantiation SCN for each table at the source database owned by the schema. This procedure uses the `SET_TABLE_INSTANTIATION_SCN` procedure to set the instantiation SCN for each table. Each table instantiation SCN is used for DDL LCRs and row LCRs on the table.

If the `recursive` parameter is set to `FALSE`, then this procedure does not set the instantiation SCN for any tables.

### The SET\_SCHEMA\_INSTANTIATION\_SCN Procedure and XStream Outbound Servers

Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

**See Also:** *Oracle Database XStream Guide* for more information about outbound servers and instantiation SCNs

### The SET\_SCHEMA\_INSTANTIATION\_SCN Procedure and XStream Inbound Servers

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.



## SET\_TABLE\_INSTANTIATION\_SCN Procedure

This procedure records the specified instantiation SCN for the specified table in the specified source database. This procedure overwrites any existing instantiation SCN for the particular table.

This procedure gives you precise control over which logical change records (LCRs) for a table are ignored and which LCRs are applied by an apply component.

### Syntax

```
DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
    source_object_name    IN  VARCHAR2,
    source_database_name  IN  VARCHAR2,
    instantiation_scn     IN  NUMBER,
    apply_database_link   IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 21–24 SET\_TABLE\_INSTANTIATION\_SCN Procedure Parameters**

Parameter	Description
source_object_name	The name of the source object specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.  When setting an instantiation SCN for a database object, always specify the name of the schema and database object at the source database, even if a rule-based transformation or apply handler is configured to change the schema name or database object name.
source_database_name	The global name of the source database. For example, DBS1.NET.  If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is NET, then the procedure specifies DBS1.NET automatically.
instantiation_scn	The instantiation SCN. Specify NULL to remove the instantiation SCN metadata for the source table from the data dictionary.
apply_database_link	The name of the database link to a non-Oracle database. This parameter should be set only when the destination database of a local apply component is a non-Oracle database.

### Usage Notes

The following usage notes apply to this procedure:

- [Instantiation SCNs and LCRs](#)
- [The SET\\_TABLE\\_INSTANTIATION\\_SCN Procedure and XStream Outbound Servers](#)
- [The SET\\_TABLE\\_INSTANTIATION\\_SCN Procedure and XStream Inbound Servers](#)

**Instantiation SCNs and LCRs**

If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at some destination database, then the apply component at the destination database disregards the LCR. Otherwise, the apply component applies the LCR.

The table instantiation SCN specified by this procedure is used on the following types of LCRs:

- Row LCRs for the table
- DDL LCRs that have a non-NULL `base_table_owner` and `base_table_name` specified, except for DDL LCRs with a `command_type` of `CREATE TABLE`

For example, the table instantiation SCN set by this procedure is used for DDL LCRs with a `command_type` of `ALTER TABLE` or `CREATE TRIGGER`.

---

---

**Note:** The instantiation SCN specified by this procedure is used only for LCRs captured by a capture process. It is not used for user-created LCRs.

---

---

**See Also:**

- [SET\\_GLOBAL\\_INSTANTIATION\\_SCN Procedure](#) on page 21-50
- [SET\\_SCHEMA\\_INSTANTIATION\\_SCN Procedure](#) on page 21-66
- [LCR\\$\\_ROW\\_RECORD Type](#) on page 248-15 for more information about row LCRs
- [LCR\\$\\_DDL\\_RECORD Type](#) on page 248-6 for more information about DDL LCRs
- *Oracle Streams Replication Administrator's Guide*

**The SET\_TABLE\_INSTANTIATION\_SCN Procedure and XStream Outbound Servers**

Instantiation SCNs are not required for database objects processed by an outbound server. If an instantiation SCN is set for a database object, then the outbound server only sends the LCRs for the database object with SCN values that are greater than the instantiation SCN value. If a database object does not have an instantiation SCN set, then the outbound server skips the instantiation SCN check and sends all LCRs for that database object. In both cases, the outbound server only sends LCRs that satisfy its rule sets.

The `apply_database_link` parameter must be set to `NULL` or to the local database for this procedure to set an instantiation SCN for an outbound server.

**See Also:** *Oracle Database XStream Guide* for more information about outbound servers and instantiation SCNs

**The SET\_TABLE\_INSTANTIATION\_SCN Procedure and XStream Inbound Servers**

Inbound servers ignore instantiation SCNs. This procedure has no effect on XStream inbound servers.

## SET\_UPDATE\_CONFLICT\_HANDLER Procedure

This procedure adds, modifies, or removes a prebuilt update conflict handler for the specified object.

### Syntax

```
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER (
  object_name      IN  VARCHAR2,
  method_name      IN  VARCHAR2,
  resolution_column IN  VARCHAR2,
  column_list      IN  DBMS_UTILITY.NAME_ARRAY,
  apply_database_link IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 21–25 SET\_UPDATE\_CONFLICT\_HANDLER Procedure Parameters**

Parameter	Description
object_name	<p>The schema and name of the table, specified as [<i>schema_name</i>.] <i>object_name</i>, for which an update conflict handler is being added, modified, or removed.</p> <p>For example, if an update conflict handler is being added for table <code>employees</code> owned by user <code>hr</code>, then specify <code>hr.employees</code>. If the schema is not specified, then the current user is the default.</p>
method_name	<p>Type of update conflict handler to create.</p> <p>You can specify one of the prebuilt handlers, which determine whether the column list from the source database is applied for the row or whether the values in the row at the destination database are retained:</p> <ul style="list-style-type: none"> <li>■ <b>MAXIMUM:</b> Applies the column list from the source database if it has the greater value for the resolution column. Otherwise, retains the values at the destination database.</li> <li>■ <b>MINIMUM:</b> Applies the column list from the source database if it has the lesser value for the resolution column. Otherwise, retains the values at the destination database.</li> <li>■ <b>OVERWRITE:</b> Applies the column list from the source database, overwriting the column values at the destination database.</li> <li>■ <b>DISCARD:</b> Retains the column list from the destination database, discarding the column list from the source database.</li> </ul> <p>If <code>NULL</code>, then the procedure removes any existing update conflict handler with the same <code>object_name</code>, <code>resolution_column</code>, and <code>column_list</code>. If non-<code>NULL</code>, then the procedure replaces any existing update conflict handler with the same <code>object_name</code> and <code>resolution_column</code>.</p>

**Table 21–25 (Cont.) SET\_UPDATE\_CONFLICT\_HANDLER Procedure Parameters**

Parameter	Description
resolution_column	<p>Name of the column used to uniquely identify an update conflict handler. For the <code>MAXIMUM</code> and <code>MINIMUM</code> prebuilt methods, the resolution column is also used to resolve the conflict. The resolution column must be one of the columns listed in the <code>column_list</code> parameter.</p> <p><code>NULL</code> is not allowed for this parameter. For the <code>OVERWRITE</code> and <code>DISCARD</code> prebuilt methods, you can specify any column in the column list.</p>
column_list	<p>List of columns for which the conflict handler is called.</p> <p>The same column cannot be in more than one column list.</p> <p>If a conflict occurs for one or more of the columns in the list when an apply component tries to apply a row logical change record (row LCR), then the conflict handler is called to resolve the conflict. The conflict handler is not called if a conflict occurs only for columns that are not in the list.</p> <p><b>Note:</b> Prebuilt update conflict handlers do not support LOB, LONG, LONG RAW, user-defined type, and Oracle-supplied type columns. Therefore, you should not include these types of columns in the <code>column_list</code> parameter.</p>
apply_database_link	<p>The name of the database link to a non-Oracle database. This parameter should be set only when the destination database is a non-Oracle database.</p> <p><b>Note:</b> Currently, conflict handlers are not supported when applying changes to a non-Oracle database.</p>

## Usage Notes

The following usage notes apply to this procedure:

- [Modifying an Existing Update Conflict Handler](#)
- [Removing an Existing Update Conflict Handler](#)
- [Series of Actions for Conflicts](#)
- [Procedure DML Handlers for Conflicts](#)
- [A Column Can Be in Only One Column List](#)
- [Update Conflict Handlers and Non-Oracle Databases](#)
- [The SET\\_UPDATE\\_CONFLICT\\_HANDLER Procedure and XStream Outbound Servers](#)
- [The SET\\_UPDATE\\_CONFLICT\\_HANDLER Procedure and XStream Inbound Servers](#)

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about prebuilt and custom update conflict handlers

### Modifying an Existing Update Conflict Handler

If you want to modify an existing update conflict handler, then you specify the table and resolution column of an the existing update conflict handler. You can modify the prebuilt method or the column list.

**Removing an Existing Update Conflict Handler**

If you want to remove an existing update conflict handler, then specify `NULL` for the prebuilt method and specify the table, column list, and resolution column of the existing update conflict handler.

**Series of Actions for Conflicts**

If an update conflict occurs, then Oracle completes the following series of actions:

1. Calls the appropriate update conflict handler to resolve the conflict
2. If no update conflict handler is specified or if the update conflict handler cannot resolve the conflict, then calls the appropriate error handler for the apply component, table, and operation to handle the error
3. If no error handler is specified or if the error handler cannot resolve the error, then raises an error and moves the transaction containing the row LCR that caused the error to the error queue

**See Also:** ["Signature of a DML Handler Procedure or Error Handler Procedure"](#) on page 21-43 for information about setting an error handler

**Procedure DML Handlers for Conflicts**

If you cannot use a prebuilt update conflict handler to meet your requirements, then you can create a PL/SQL procedure to use as a custom conflict handler. You use the `SET_DML_HANDLER` procedure to designate one or more custom conflict handlers for a particular table. In addition, a custom conflict handler can process LOB columns and use LOB assembly.

**See Also:** [SET\\_DML\\_HANDLER Procedure](#) on page 21-41

**A Column Can Be in Only One Column List**

When a column is in a column list, and you try to add the same column to another column list, this procedure returns the following error:

```
ORA-00001: UNIQUE CONSTRAINT (SYS.APPLY$_CONF_HDLR_COLUMNS_UNQ1) VIOLATED
```

**Update Conflict Handlers and Non-Oracle Databases**

Setting an update conflict handler for an apply component that is applying to a non-Oracle database is not supported.

**The SET\_UPDATE\_CONFLICT\_HANDLER Procedure and XStream Outbound Servers**

This procedure has no effect on XStream outbound servers.

**The SET\_UPDATE\_CONFLICT\_HANDLER Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.

**Examples**

The following is an example for setting an update conflict handler for the `employees` table in the `hr` schema:

```
DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
```

```
BEGIN
  cols(1) := 'salary';
  cols(2) := 'commission_pct';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name      => 'hr.employees',
    method_name      => 'MAXIMUM',
    resolution_column => 'salary',
    column_list      => cols);
END;
/
```

This example sets a conflict handler that is called if a conflict occurs for the `salary` or `commission_pct` column in the `hr.employees` table. If such a conflict occurs, then the `salary` column is evaluated to resolve the conflict. If a conflict occurs only for a column that is not in the column list, such as the `job_id` column, then this conflict handler is not called.

## SET\_VALUE\_DEPENDENCY Procedure

This procedure sets or removes a value dependency. A value dependency is a virtual dependency definition that defines a relationship between the columns of two or more tables.

An apply component uses the name of a value dependencies to detect dependencies between row logical change records (row LCRs) that contain the columns defined in the value dependency. Value dependencies can define virtual foreign key relationships between tables, but, unlike foreign key relationships, value dependencies can involve more than two database objects.

This procedure is overloaded. The `attribute_list` and `attribute_table` parameters are mutually exclusive.

**See Also:** *Oracle Streams Concepts and Administration*

### Syntax

```
DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY (
  dependency_name IN VARCHAR2,
  object_name     IN VARCHAR2,
  attribute_list  IN VARCHAR2);
```

```
DBMS_APPLY_ADM.SET_VALUE_DEPENDENCY (
  dependency_name IN VARCHAR2,
  object_name     IN VARCHAR2,
  attribute_table IN DBMS_UTILITY.NAME_ARRAY);
```

### Parameters

**Table 21–26 SET\_VALUE\_DEPENDENCY Procedure Parameters**

Parameter	Description
<code>dependency_name</code>	The name of the value dependency.  If a dependency with the specified name does not exist, then it is created.  If a dependency with the specified name exists, then the specified object and attributes are added to the dependency.  If NULL, an error is raised.
<code>object_name</code>	The name of the table, specified as [ <i>schema_name</i> .] <i>table_name</i> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default.  If NULL and the specified dependency exists, then the dependency is removed. If NULL and the specified dependency does not exist, then an error is raised.  If NULL, then <code>attribute_list</code> and <code>attribute_table</code> also must be NULL.
<code>attribute_list</code>	A comma-delimited list of column names in the table. There must be no spaces between entries.
<code>attribute_table</code>	A PL/SQL associative array of type <code>DBMS_UTILITY.NAME_ARRAY</code> that contains names of columns in the table. The first column name should be at position 1, the second at position 2, and so on. The table does not need to be NULL terminated.

## Usage Notes

The following usage notes apply to this procedure:

- [The SET\\_VALUE\\_DEPENDENCY Procedure and XStream Outbound Servers](#)
- [The SET\\_VALUE\\_DEPENDENCY Procedure and XStream Inbound Servers](#)

### **The SET\_VALUE\_DEPENDENCY Procedure and XStream Outbound Servers**

This procedure has no effect on XStream outbound servers.

### **The SET\_VALUE\_DEPENDENCY Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.



## START\_APPLY Procedure

This procedure directs the apply component to start applying messages.

### Syntax

```
DBMS_APPLY_ADM.START_APPLY(
    apply_name IN VARCHAR2);
```

### Parameter

**Table 21–27 START\_APPLY Procedure Parameter**

Parameter	Description
apply_name	The apply component name. A NULL setting is not allowed. Do not specify an owner.

### Usage Notes

The following usage notes apply to this procedure:

- [Apply Component Status](#)
- [The START\\_APPLY Procedure and XStream Outbound Servers](#)
- [The START\\_APPLY Procedure and XStream Inbound Servers](#)

#### Apply Component Status

The apply component status is persistently recorded. Hence, if the status is `ENABLED`, then the apply component is started upon database instance startup. An apply component (`anmn`) is an Oracle background process. The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the start status of an apply component.

#### The START\_APPLY Procedure and XStream Outbound Servers

This procedure functions the same way for apply processes and outbound servers.

#### The START\_APPLY Procedure and XStream Inbound Servers

This procedure functions the same way for apply processes and inbound servers.

## STOP\_APPLY Procedure

This procedure stops the apply component from applying messages and rolls back any unfinished transactions being applied.

### Syntax

```
DBMS_APPLY_ADM.STOP_APPLY (
  apply_name IN VARCHAR2,
  force      IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 21–28 STOP\_APPLY Procedure Parameters**

Parameter	Description
apply_name	The apply component name. A NULL setting is not allowed. Do not specify an owner.
force	<p>If TRUE, then the procedure stops the apply component as soon as possible.</p> <p>If FALSE, then the procedure stops the apply component after ensuring that there are no gaps in the set of applied transactions.</p> <p>The behavior of the apply component depends on the setting specified for the <code>force</code> parameter and the setting specified for the <code>commit_serialization</code> apply component parameter. See "Usage Notes" for more information.</p>

### Usage Notes

The following usage notes apply to this procedure:

- [Apply Component Status](#)
- [Queue Subprograms Have No Effect on Apply Component Status](#)
- [The STOP\\_APPLY force Parameter and the commit\\_serialization Apply Parameter](#)
- [The STOP\\_APPLY Procedure and XStream Outbound Servers](#)
- [The STOP\\_APPLY Procedure and XStream Inbound Servers](#)

#### Apply Component Status

The apply component status is persistently recorded. Hence, if the status is `DISABLED` or `ABORTED`, then the apply component is not started upon database instance startup.

#### Queue Subprograms Have No Effect on Apply Component Status

The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the `STOP` status of an apply component.

#### The STOP\_APPLY force Parameter and the commit\_serialization Apply Parameter

The following table describes apply component behavior for each setting of the `force` parameter in the `STOP_APPLY` procedure and the `commit_serialization` apply component parameter. In all cases, the apply component rolls back any unfinished transactions when it stops.

<b>force</b>	<b>commit_serialization</b>	<b>Apply Component Behavior</b>
TRUE	FULL	The apply component stops immediately and does not apply any unfinished transactions.
TRUE	DEPENDENT_TRANSACTION	When the apply component stops, some transactions that have been applied locally might have committed at the source database at a later point in time than some transactions that have not been applied locally.
FALSE	FULL	The apply component stops after applying the next uncommitted transaction in the commit order, if any such transaction is in progress.
FALSE	DEPENDENT_TRANSACTION	Before stopping, the apply component applies all of the transactions that have a commit time that is earlier than the applied transaction with the most recent commit time.

For example, assume that the `commit_serialization` apply component parameter is set to `DEPENDENT_TRANSACTION` and there are three transactions: transaction 1 has the earliest commit time, transaction 2 is committed after transaction 1, and transaction 3 has the latest commit time. Also assume that an apply component has applied transaction 1 and transaction 3 and is in the process of applying transaction 2 when the `STOP_APPLY` procedure is run. Given this scenario, if the `force` parameter is set to `TRUE`, then transaction 2 is not applied, and the apply component stops (transaction 2 is rolled back). If, however, the `force` parameter is set to `FALSE`, then transaction 2 is applied before the apply component stops.

A different scenario would result if the `commit_serialization` apply component parameter is set to `FULL`. For example, assume that the `commit_serialization` apply component parameter is set to `FULL` and there are three transactions: transaction A has the earliest commit time, transaction B is committed after transaction A, and transaction C has the latest commit time. In this case, the apply component has applied transaction A and is in the process of applying transactions B and C when the `STOP_APPLY` procedure is run. Given this scenario, if the `force` parameter is set to `TRUE`, then transactions B and C are not applied, and the apply component stops (transactions B and C are rolled back). If, however, the `force` parameter is set to `FALSE`, then transaction B is applied before the apply component stops, and transaction C is rolled back.

**See Also:** [SET\\_PARAMETER Procedure](#) on page 21-56 for more information about the `commit_serialization` apply component parameter

### **The STOP\_APPLY Procedure and XStream Outbound Servers**

This procedure functions the same way for apply processes and outbound servers.

### **The STOP\_APPLY Procedure and XStream Inbound Servers**

This procedure functions the same way for apply processes and inbound servers.



The `DBMS_AQ` package provides an interface to Oracle Streams Advanced Queuing (AQ).

**See Also:**

- *Oracle Streams Advanced Queuing User's Guide*
- [Oracle Streams AQ TYPES](#) for information about TYPES to use with `DBMS_AQ`.

This chapter contains the following topics:

- [Using DBMS\\_AQ](#)
  - Constants
  - Data Structures
  - Operational Notes
- [Summary of DBMS\\_AQ Subprograms](#)

## Using DBMS\_AQ

- [Constants](#)
- [Data Structures](#)
- [Operational Notes](#)

## Constants

The DBMS\_AQ package uses the constants shown in [Table 22–1](#).

When using enumerated constants such as BROWSE, LOCKED, or REMOVE, the PL/SQL constants must be specified with the scope of the packages defining it. All types associated with the operational interfaces have to be prepended with DBMS\_AQ. For example: DBMS\_AQ.BROWSE.

---

**Note:** The `sequence_deviation` attribute has no effect in releases prior to Oracle Streams AQ 10g Release 1 (10.1) if `message_grouping` parameter of DBMS\_AQADM subprograms is set to TRANSACTIONAL. The sequence deviation feature is deprecated in Oracle Streams AQ 10g Release 2 (10.2).

---

**Table 22–1 Enumerated Constants**

Parameter	Options	Type	Description
VISIBILITY	IMMEDIATE		
.	ON_COMMIT		
DEQUEUE_MODE	BROWSE		
.	LOCKED		
.	REMOVE		
.	REMOVE_NODATA		
NAVIGATION	FIRST_MESSAGE		
.	NEXT_MESSAGE		
STATE	WAITING		
.	READY		
.	PROCESSED		
.	EXPIRED		
SEQUENCE_DEVIATION	BEFORE		
.	TOP		
WAIT	FOREVER	BINARY_INTEGER	
.	NO_WAIT	BINARY_INTEGER	
DELAY	NO_DELAY		
EXPIRATION	NEVER		
NAMESPACE	NAMESPACE_AQ		
.	NAMESPACE_ANONYMOUS		
NTFN_GROUPING_CLASS	NTFN_GROUPING_CLASS_TIME	NUMBER	
NTFN_GROUPING_TYPE	NTFN_GROUPING_TYPE_SUMMARY	NUMBER	

**Table 22–1 (Cont.) Enumerated Constants**

<b>Parameter</b>	<b>Options</b>	<b>Type</b>	<b>Description</b>
.	NTFN_GROUPING_TYPE_ LAST	NUMBER	
NTFN_GROUPING_ REPEAT_COUNT	NTFN_GROUPING_ FOREVER	NUMBER	



## Data Structures

**Table 22–2 DBMS\_AQ Data Structures**

Data Structures	Description
<a href="#">Object Name</a> on page 22-5	Names database objects
<a href="#">Type Name</a> on page 22-5	Defines queue types
<a href="#">Oracle Streams AQ PL/SQL Callback</a> on page 22-6	Specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification

### Object Name

The `object_name` data structure names database objects. It applies to queues, queue tables, agent names, and object types.

#### Syntax

```
object_name := VARCHAR2;
object_name := [schema_name.]name;
```

#### Usage Notes

Names for objects are specified by an optional schema name and a name. If the schema name is not specified, the current schema is assumed. The name must follow object name guidelines in *Oracle Database SQL Language Reference* with regard to reserved characters. Schema names, agent names, and object type names can be up to 30 bytes long. Queue names and queue table names can be up to 24 bytes long.

### Type Name

The `type_name` data structure defines queue types.

#### Syntax

```
type_name := VARCHAR2;
type_name := object_type | "RAW";
```

#### Attributes

**Table 22–3 Type Name Attributes**

Attribute	Description
<code>object_type</code>	Maximum number of attributes in the object type is limited to 900.

**Table 22–3 (Cont.) Type Name Attributes**

Attribute	Description
"RAW"	<p>To store payload of type RAW, Oracle Streams AQ creates a queue table with a LOB column as the payload repository. The theoretical maximum size of the message payload is the maximum amount of data that can be stored in a LOB column. However, the maximum size of the payload is determined by which programmatic environment you use to access Oracle Streams AQ. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G. Because the PL/SQL enqueue and dequeue interfaces accept RAW buffers as the payload parameters you will be limited to 32K bytes. In OCI, the maximum size of your RAW data will be limited to the maximum amount of contiguous memory (as an OCIRaw is simply an array of bytes) that the OCI Object Cache can allocate. Typically, this will be at least 32K bytes and much larger in many cases.</p> <p>Because LOB columns are used for storing RAW payload, the Oracle Streams AQ administrator can choose the LOB tablespace and configure the LOB storage by constructing a LOB storage string in the <code>storage_clause</code> parameter during queue table creation time.</p>

## Oracle Streams AQ PL/SQL Callback

The `plsqllcallback` data structure specifies the user-defined PL/SQL procedure, defined in the database to be invoked on message notification.

### Syntax

If a notification message is expected for a RAW payload enqueue, then the PL/SQL callback must have the following signature:

```
procedure plsqllcallback(
  context IN RAW,
  reginfo IN SYS.AQ$_REG_INFO,
  descr   IN SYS.AQ$_DESCRIPTOR,
  payload IN RAW,
  payloadl IN NUMBER);
```

### Attributes

**Table 22–4 Oracle Streams AQ PL/SQL Callback Attributes**

Attribute	Description
<code>context</code>	Specifies the context for the callback function that was passed by <code>dbms_aq.register</code> . See <a href="#">AQ\$_REG_INFO Type</a> on page 243-12.
<code>reginfo</code>	See <a href="#">AQ\$_REG_INFO Type</a> on page 243-12.
<code>descr</code>	See <a href="#">AQ\$_DESCRIPTOR Type</a> on page 243-5
<code>payload</code>	If a notification message is expected for a raw payload enqueue then this contains the raw payload that was enqueued into a non persistent queue. In case of a persistent queue with raw payload this parameter will be null.
<code>payloadl</code>	Specifies the length of <code>payload</code> . If <code>payload</code> is null, <code>payloadl = 0</code> .

If the notification message is expected for an ADT payload enqueue, the PL/SQL callback must have the following signature:

```
procedure plsqllcallback(
  context IN RAW,
  reginfo IN SYS.AQ$_REG_INFO,
  descr   IN SYS.AQ$_DESCRIPTOR,
```

```
payload IN VARCHAR2,  
payload1 IN NUMBER);
```

## Operational Notes

- [DBMS\\_AQ and DBMS\\_AQADM Java Classes](#)

### **DBMS\_AQ and DBMS\_AQADM Java Classes**

Java interfaces are available for DBMS\_AQ and DBMS\_AQADM. The Java interfaces are provided in the `$ORACLE_HOME/rdbms/jlib/aqapi.jar`. Users are required to have EXECUTE privileges on the DBMS\_AQIN package to use these interfaces.

---

## Summary of DBMS\_AQ Subprograms

**Table 22-5 DBMS\_AQ Package Subprograms**

Subprograms	Description
<a href="#">BIND_AGENT Procedure</a> on page 22-10	Creates an entry for an Oracle Streams AQ agent in the LDAP directory
<a href="#">DEQUEUE Procedure</a> on page 22-11	Dequeues a message from the specified queue
<a href="#">DEQUEUE_ARRAY Function</a> on page 22-14	Dequeues an array of messages from the specified queue
<a href="#">ENQUEUE Procedure</a> on page 22-16	Adds a message to the specified queue
<a href="#">ENQUEUE_ARRAY Function</a> on page 22-18	Adds an array of messages to the specified queue
<a href="#">LISTEN Procedures</a> on page 22-19	Listen to one or more queues on behalf of a list of agents
<a href="#">POST Procedure</a> on page 22-21	Posts to a anonymous subscription which allows all clients who are registered for the subscription to get notifications
<a href="#">REGISTER Procedure</a> on page 22-22	Registers for message notifications
<a href="#">UNBIND_AGENT Procedure</a> on page 22-23	Removes an entry for an Oracle Streams AQ agent from the LDAP directory
<a href="#">UNREGISTER Procedure</a> on page 22-24	Unregisters a subscription which turns off notification

---

**Note:** DBMS\_AQ does not have a purity level defined; therefore, you cannot call any procedure in this package from other procedures that have RNDS, WNDS, RNPS or WNPS constraints defined.

---

## BIND\_AGENT Procedure

This procedure creates an entry for an Oracle Streams AQ agent in the LDAP server.

### Syntax

```
DBMS_AQ.BIND_AGENT(  
  agent      IN SYS.AQ$_AGENT,  
  certificate IN VARCHAR2 default NULL);
```

### Parameters

**Table 22–6** *BIND\_AGENT Procedure Parameters*

Parameter	Description
agent	Agent that is to be registered in LDAP server.
certificate	Location (LDAP distinguished name) of the "organizationalperson" entry in LDAP whose digital certificate (attribute <code>usercertificate</code> ) is to be used for this agent. Example: "cn=OE, cn=ACME, cn=com" is a distinguished name for a <code>OrganizationalPerson</code> OE whose certificate will be used with the specified agent.

### Usage Notes

In the LDAP server, digital certificates are stored as an attribute (`usercertificate`) of the `OrganizationalPerson` entity. The distinguished name for this `OrganizationalPerson` must be specified when binding the agent.

## DEQUEUE Procedure

This procedure dequeues a message from the specified queue.

### Syntax

```
DBMS_AQ.DEQUEUE (
  queue_name          IN      VARCHAR2,
  dequeue_options    IN      dequeue_options_t,
  message_properties OUT    message_properties_t,
  payload            OUT    "<ADT_1>"
  msgid             OUT    RAW);
```

### Parameters

**Table 22-7 DEQUEUE Procedure Parameters**

Parameter	Description
queue_name	Specifies the name of the queue.
dequeue_options	See <a href="#">DEQUEUE_OPTIONS_T Type</a> on page 243-17.
message_properties	See <a href="#">MESSAGE_PROPERTIES_T Type</a> on page 243-24.
payload	Not interpreted by Oracle Streams AQ. The payload must be specified according to the specification in the associated queue table. For the definition of <i>type_name</i> refer to <a href="#">Type Name</a> on page 22-5.
msgid	System generated identification of the message.

### Usage Notes

The search criteria for messages to be dequeued is determined by the following parameters in `dequeue_options`:

- `consumer_name`
- `msgid`

`Msgid` uniquely identifies the message to be dequeued. Only messages in the `READY` state are dequeued unless `msgid` is specified.

- `correlation`

Correlation identifiers are application-defined identifiers that are not interpreted by Oracle Streams AQ.

- `deq_condition`

Dequeue condition is an expression based on the message properties, the message data properties and PL/SQL functions. A `deq_condition` is specified as a Boolean expression using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the `where` clause of a SQL query). Message properties include `priority`, `corrid` and other columns in the queue table.

To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with `tab.user_data` as a qualifier to indicate the specific column of the queue table that stores the payload.

Example: `tab.user_data.orderstatus='EXPRESS'`

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the `msgid` and correlation ID in `dequeue_options`.

The database-consistent read mechanism is applicable for queue operations. For example, a `BROWSE` call may not see a message that is enqueued after the beginning of the browsing transaction.

The default `NAVIGATION` parameter during dequeue is `NEXT_MESSAGE`. This means that subsequent dequeues will retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the first dequeue command will be processed only after processing all the remaining messages in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering. However, applications must use the `FIRST_MESSAGE` navigation option when the first message in the queue needs to be processed by every dequeue command. This usually becomes necessary when a higher priority message arrives in the queue while messages already-enqueued are being processed.

---

---

**Note:** It may be more efficient to use the `FIRST_MESSAGE` navigation option when messages are concurrently enqueued. If the `FIRST_MESSAGE` option is not specified, Oracle Streams AQ continually generates the snapshot as of the first dequeue command, leading to poor performance. If the `FIRST_MESSAGE` option is specified, then Oracle Streams AQ uses a new snapshot for every dequeue command.

---

---

Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, then this will effectively form a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in `LOCKED` or `REMOVE` mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use the `NEXT_TRANSACTION` to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue will time out after the specified `WAIT` period.

### Using Secure Queues

For secure queues, you must specify `consumer_name` in the `dequeue_options` parameter. See [DEQUEUE\\_OPTIONS\\_T Type](#) on page 243-17 for more information about `consumer_name`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Streams AQ agent using `DBMS_AQADM.CREATE_AQ_AGENT`. See [CREATE\\_AQ\\_AGENT Procedure](#) on page 23-22.
- You must map the Oracle Streams AQ agent to a database user with dequeue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this. See [ENABLE\\_DB\\_ACCESS Procedure](#) on page 23-35.



**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

## DEQUEUE\_ARRAY Function

This function dequeues an array of messages and returns them in the form of an array of payloads, an array of message properties and an array of message IDs. This function returns the number of messages successfully dequeued.

### Syntax

```
DBMS_AQ.DEQUEUE_ARRAY (
    queue_name           IN   VARCHAR2,
    dequeue_options     IN   dequeue_options_t,
    array_size          IN   pls_integer,
    message_properties_array OUT message_properties_array_t,
    payload_array       OUT   "<COLLECTION_1>",
    msgid_array         OUT   msgid_array_t,
    error_array         OUT   error_array_t)
RETURN pls_integer;
```

### Parameters

**Table 22–8 DEQUEUE\_ARRAY Function Parameters**

Parameter	Description
queue_name	The queue name from which messages are dequeued (same as single-row dequeue).
dequeue_options	The set of options which will be applied to all messages in the array (same as single-row dequeue).
array_size	The number of elements to dequeue.
message_properties_array	A record containing an array corresponding to each message property. Each payload element has a corresponding set of message properties. See <a href="#">MESSAGE_PROPERTIES_ARRAY_T Type</a> on page 243-28.
payload_array	An array of dequeued payload data. "<COLLECTION_1>" can be an associative array, varray or nested table in its PL/SQL representation.
msgid_array	An array of message IDs of the dequeued messages. See <a href="#">MSGID_ARRAY_T Type</a> on page 243-29.
error_array	Currently not implemented

### Usage Notes

A nonzero wait time, as specified in `dequeue_options`, is recognized only when there are no messages in the queue. If the queue contains messages that are eligible for dequeue, then the `DEQUEUE_ARRAY` function will dequeue up to `array_size` messages and return immediately.

Dequeue by `message_id` is not supported. See [DEQUEUE Procedure](#) on page 22-11 for more information on the `navigation` parameter. Existing `NAVIGATION` modes are supported. In addition, two new `NAVIGATION` modes are supported for queues enabled for message grouping:

- `FIRST_MESSAGE_MULTI_GROUP`
- `NEXT_MESSAGE_MULTI_GROUP`

**See Also:** [ENQUEUE\\_OPTIONS\\_T Type](#) on page 243-20

For transaction grouped queues and `ONE_GROUP` navigation, messages are dequeued from a single transaction group only, subject to the `array_size` limit. In `MULTI_GROUP` navigation, messages are dequeued across multiple transaction groups, still subject to the `array_size` limit. `ORA-25235` is returned to indicate the end of a transaction group.

`DEQUEUE_ARRAY` is not supported for buffered messages, but you can still use this procedure on individual buffered messages by setting `array_size` to one message.

## ENQUEUE Procedure

This procedure adds a message to the specified queue.

### Syntax

```
DBMS_AQ.ENQUEUE (
  queue_name          IN      VARCHAR2,
  enqueue_options    IN      enqueue_options_t,
  message_properties IN      message_properties_t,
  payload            IN      "<ADT_1>",
  msgid              OUT     RAW);
```

### Parameters

**Table 22–9 ENQUEUE Procedure Parameters**

Parameter	Description
queue_name	Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue.
enqueue_options	See <a href="#">ENQUEUE_OPTIONS_T Type</a> on page 243-20.
message_properties	See <a href="#">MESSAGE_PROPERTIES_T Type</a> on page 243-24.
payload	Not interpreted by Oracle Streams AQ. The payload must be specified according to the specification in the associated queue table. NULL is an acceptable parameter. For the definition of <i>type_name</i> refer to <a href="#">Type Name</a> on page 22-5.
msgid	System generated identification of the message. This is a globally unique identifier that can be used to identify the message at dequeue time.

### Usage Notes

The `sequence_deviation` parameter in `enqueue_options` can be used to change the order of processing between two messages. The identity of the other message, if any, is specified by the `enqueue_options` parameter `relative_msgid`. The relationship is identified by the `sequence_deviation` parameter.

Specifying `sequence_deviation` for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

---

**Note:** The `sequence_deviation` attribute has no effect in releases prior to Oracle Streams AQ 10g Release 1 (10.1) if `message_grouping` parameter of `DBMS_AQADM` subprograms is set to `TRANSACTIONAL`. The sequence deviation feature is deprecated in Oracle Streams AQ 10g Release 2 (10.2).

---

If a message is enqueued to a multiconsumer queue with no recipient, and if the queue has no subscribers (or rule-based subscribers that match this message), then Oracle error `ORA_24033` is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

### Using Secure Queues

For secure queues, you must specify the `sender_id` in the `messages_properties` parameter. See [MESSAGE\\_PROPERTIES\\_T Type](#) on page 243-24 for more information about `sender_id`.

When you use secure queues, the following are required:

- You must have created a valid Oracle Streams AQ agent using `DBMS_AQADM.CREATE_AQ_AGENT`. See [CREATE\\_AQ\\_AGENT Procedure](#) on page 23-22.
- You must map `sender_id` to a database user with enqueue privileges on the secure queue. Use `DBMS_AQADM.ENABLE_DB_ACCESS` to do this. See [ENABLE\\_DB\\_ACCESS Procedure](#) on page 23-35.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

## ENQUEUE\_ARRAY Function

This function enqueues an array of payloads using a corresponding array of message properties. The output will be an array of message IDs of the enqueued messages.

### Syntax

```
DBMS_AQ.ENQUEUE_ARRAY (
  queue_name          IN   VARCHAR2,
  enqueue_options    IN   enqueue_options_t,
  array_size         IN   pls_integer,
  message_properties_array IN message_properties_array_t,
  payload_array      IN   "<COLLECTION_1>",
  msgid_array        OUT  msgid_array_t,
  error_array        OUT  error_array_t)
RETURN pls_integer;
```

### Parameters

**Table 22-10 ENQUEUE\_ARRAY Function Parameters**

Parameter	Description
queue_name	The queue name in which messages are enqueued (same as single-row enqueue).
enqueue_options	See <a href="#">ENQUEUE_OPTIONS_T Type</a> on page 243-20.
array_size	The number of elements to enqueue.
message_properties_array	A record containing an array corresponding to each message property. For each property, the user must allocate <code>array_size</code> elements. See <a href="#">MESSAGE_PROPERTIES_ARRAY_T Type</a> on page 243-28.
payload_array	An array of payload data. "<COLLECTION_1>" can be an associative array, <code>VARRAY</code> , or nested table in its PL/SQL representation.
msgid_array	An array of message IDs for the enqueued messages. If an error occurs for a particular message, then its corresponding message ID is null. See <a href="#">MSGID_ARRAY_T Type</a> on page 243-29.
error_array	Currently not implemented

### Usage Notes

`ENQUEUE_ARRAY` is not supported for buffered messages, but you can still use this procedure on individual buffered messages by setting `array_size` to one message.

## LISTEN Procedures

This procedure listens on one or more queues on behalf of a list of agents. The address field of the agent indicates the queue the agent wants to monitor. Only local queues are supported as addresses. Protocol is reserved for future use.

### Syntax

```
DBMS_AQ.LISTEN (
  agent_list      IN    AQ$_AGENT_LIST_T,
  wait            IN    BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER,
  agent           OUT   SYS.AQ$_AGENT);
```

```
DBMS_AQ.LISTEN (
  agent_list      IN    AQ$_AGENT_LIST_T,
  wait            IN    BINARY_INTEGER DEFAULT FOREVER,
  listen_delivery_mode IN PLS_INTEGER DEFAULT DBMS_AQ.PERSISTENT,
  agent           OUT   SYS.AQ$_AGENT,
  message_delivery_mode OUT PLS_INTEGER);
```

```
TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
```

### Parameters

**Table 22–11 LISTEN Procedure Parameters**

Parameter	Description
agent_list	List of agents to listen for
wait	Time out for the listen call in seconds. By default, the call will block forever.
listen_delivery_mode	The caller specifies whether it is interested in persistent, buffered messages or both types of messages, specifying a delivery mode of DBMS_AQ.PERSISTENT or DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT_OR_BUFFERED
agent	Agent with a message available for consumption
message_delivery_mode	Returns the message type along with the queue and consumer for which there is a message

### Usage Notes

If agent-address is a multiconsumer queue, then agent-name is mandatory. For single-consumer queues, agent-name must not be specified.

This procedure takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses. Protocol is reserved for future use.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, only the first agent listed is returned. If there are no messages found when the wait time expires, an error is raised.

A successful return from the `LISTEN` call is only an indication that there is a message for one of the listed agents in one the specified queues. The interested agent must still dequeue the relevant message.

---

---

**Note:** You cannot call `LISTEN` on nonpersistent queues.

---

---



## POST Procedure

This procedure posts to a list of anonymous subscriptions that allows all clients who are registered for the subscriptions to get notifications.

### Syntax

```
DBMS_AQ.POST (
  post_list      IN SYS.AQ$_POST_INFO_LIST,
  post_count     IN NUMBER);
```

### Parameters

**Table 22–12** *POST Procedure Parameters*

Parameter	Description
post_list	Specifies the list of anonymous subscriptions to which you want to post. It is a list of <a href="#">AQ\$_POST_INFO_LIST</a> Type.
post_count	Specifies the number of entries in the post_list.

### Usage Notes

This procedure is used to post to anonymous subscriptions which allows all clients who are registered for the subscriptions to get notifications. Several subscriptions can be posted to at one time.

## REGISTER Procedure

This procedure registers an e-mail address, user-defined PL/SQL procedure, or HTTP URL for message notification.

### Syntax

```
DBMS_AQ.REGISTER (
  reg_list      IN  SYS.AQ$_REG_INFO_LIST,
  count         IN  NUMBER);
```

### Parameters

**Table 22–13 REGISTER Procedure Parameters**

Parameter	Description
reg_list	Specifies the list of subscriptions to which you want to register for message notifications. It is a list of <a href="#">AQ\$_REG_INFO Type</a> .
count	Specifies the number of entries in the reg_list.

### Usage Notes

- This procedure is used to register for notifications. You can specify an e-mail address to which message notifications are sent, register a procedure to be invoked on a notification, or register an HTTP URL to which the notification is posted. Interest in several subscriptions can be registered at one time.
- The procedure can also be used to register for grouping notifications using five grouping attributes:
  - Class – grouping criterion (currently only `TIME` criterion is supported)
  - Value – the value of the grouping criterion (currently only time in seconds for criterion `TIME`)
  - Type – summary or last, also contains count of notifications received in group (for `AQ` namespace only, not for `DBCHANGE` namespace)
  - Repeat count – how many times to perform grouping (Default is `FOREVER`)
  - Start time – when to start grouping (Default is current time)
- If you register for e-mail notifications, you should set the host name and port name for the SMTP server that will be used by the database to send e-mail notifications. If required, you should set the send-from e-mail address, which is set by the database as the `sent from` field. You need a Java-enabled database to use this feature.
- If you register for HTTP notifications, you may want to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

**See Also:** [Chapter 24, "DBMS\\_AQELM"](#) for more information on e-mail and HTTP notifications

## UNBIND\_AGENT Procedure

This procedure removes the entry for an Oracle Streams AQ agent from the LDAP server.

### Syntax

```
DBMS_AQ.UNBIND_AGENT (  
    agent      IN SYS.AQ$_AGENT);
```

### Parameters

**Table 22–14** *BIND\_AGENT Procedure Parameters*

Parameter	Description
agent	Agent that is to be removed from the LDAP server

## UNREGISTER Procedure

This procedure unregisters a subscription which turns off notifications.

### Syntax

```
DBMS_AQ.UNREGISTER (  
    reg_list      IN  SYS.AQ$_REG_INFO_LIST,  
    reg_count     IN  NUMBER);
```

### Parameters

**Table 22–15 UNREGISTER Procedure Parameters**

Parameter	Description
reg_list	Specifies the list of subscriptions to which you want to register for message notifications. It is a list of <a href="#">AQ\$_REG_INFO Type</a> .
reg_count	Specifies the number of entries in the reg_list.

### Usage Notes

This procedure is used to unregister a subscription which turns off notifications. Several subscriptions can be unregistered from at one time.

The DBMS\_AQADM package provides procedures to manage Oracle Streams Advanced Queuing (AQ) configuration and administration information.

**See Also:**

- *Oracle Streams Advanced Queuing User's Guide*
- [Chapter 243, "Oracle Streams AQ TYPES"](#) for information about the TYPES to use with DBMS\_AQADM

This chapter contains the following topics:

- [Using DBMS\\_AQADM](#)
  - Constants
- [Subprogram Groups](#)
  - Queue Table Subprograms
  - Privilege Subprograms
  - Queue Subprograms
  - Subscriber Subprograms
  - Notification Subprograms
  - Propagation Subprograms
  - Oracle Streams AQ Agent Subprograms
  - Alias Subprograms
- [Summary of DBMS\\_AQADM Subprograms](#)

---

## Using DBMS\_AQADM

This section contains the following topics.

- [Constants](#)

## Constants

When using enumerated constants, such as INFINITE, TRANSACTIONAL, or NORMAL\_QUEUE, the symbol must be specified with the scope of the packages defining it. All types associated with the administrative interfaces must be prepended with DBMS\_AQADM. For example: DBMS\_AQADM.NORMAL\_QUEUE.

**Table 23–1** *Enumerated Types in the Administrative Interface*

Parameter	Options
retention	0, 1, 2...INFINITE
message_grouping	TRANSACTIONAL, NONE
queue_type	NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE

**See Also:** For more information on the Java classes and data structures used in both DBMS\_AQ and DBMS\_AQADM, see the [DBMS\\_AQ](#) package.

## Subprogram Groups

This DBMS\_AQADM package is made up of the following subprogram groups:

- [Queue Table Subprograms](#) on page 23-5
- [Privilege Subprograms](#) on page 23-6
- [Queue Subprograms](#) on page 23-7
- [Subscriber Subprograms](#) on page 23-8
- [Notification Subprograms](#) on page 23-9
- [Propagation Subprograms](#) on page 23-10
- [Oracle Streams AQ Agent Subprograms](#) on page 23-11
- [Alias Subprograms](#) on page 23-12



## Queue Table Subprograms

**Table 23–2** *Queue Table Subprograms*

<b>Subprograms</b>	<b>Description</b>
<a href="#">ALTER_QUEUE_TABLE Procedure</a> on page 23-20	Alters the existing properties of a queue table
<a href="#">CREATE_QUEUE_TABLE Procedure</a> on page 23-26	Creates a queue table for messages of a predefined type
<a href="#">DROP_QUEUE_TABLE Procedure</a> on page 23-34	Drops an existing queue table
<a href="#">ENABLE_JMS_TYPES Procedure</a> on page 23-36	A precondition for the enqueue of JMS types and XML types
<a href="#">MIGRATE_QUEUE_TABLE Procedure</a> on page 23-41	Upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table
<a href="#">PURGE_QUEUE_TABLE Procedure</a> on page 23-42	Purges messages from queue tables

## Privilege Subprograms

**Table 23–3** *Privilege Subprograms*

<b>Subprograms</b>	<b>Description</b>
<a href="#">GRANT_QUEUE_PRIVILEGE Procedure</a> on page 23-39	Grants privileges on a queue to users and roles
<a href="#">GRANT_SYSTEM_PRIVILEGE Procedure</a> on page 23-40	Grants Oracle Streams AQ system privileges to users and roles
<a href="#">REVOKE_QUEUE_PRIVILEGE Procedure</a> on page 23-46	Revokes privileges on a queue from users and roles
<a href="#">REVOKE_SYSTEM_PRIVILEGE Procedure</a> on page 23-47	Revokes Oracle Streams AQ system privileges from users and roles

## Queue Subprograms

**Table 23–4 Queue Subprograms**

<b>Subprograms</b>	<b>Description</b>
<a href="#">ALTER_QUEUE Procedure</a> on page 23-19	Alters existing properties of a queue
<a href="#">CREATE_NP_QUEUE Procedure</a> on page 23-23	Creates a nonpersistent RAW queue
<a href="#">CREATE_QUEUE Procedure</a> on page 23-24	Creates a queue in the specified queue table
<a href="#">DROP_QUEUE Procedure</a> on page 23-33	Drops an existing queue
<a href="#">QUEUE_SUBSCRIBERS Function</a> on page 23-44	Returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type <code>DBMS_AQADM.AQ\$_subscriber_list_t</code>
<a href="#">START_QUEUE Procedure</a> on page 23-51	Enables the specified queue for enqueueing or dequeuing
<a href="#">STOP_QUEUE Procedure</a> on page 23-52	Disables enqueueing or dequeuing on the specified queue

## Subscriber Subprograms

**Table 23–5** *Subscriber Subprograms*

<b>Subprograms</b>	<b>Description</b>
<a href="#">ADD_SUBSCRIBER Procedure</a> on page 23-16	Adds a default subscriber to a queue
<a href="#">ALTER_SUBSCRIBER Procedure</a> on page 23-21	Alters existing properties of a subscriber to a specified queue
<a href="#">REMOVE_SUBSCRIBER Procedure</a> on page 23-45	Removes a default subscriber from a queue

## Notification Subprograms

**Table 23–6 Notification Subprograms**

<b>Subprograms</b>	<b>Description</b>
<a href="#">GET_WATERMARK Procedure</a> on page 23-38	Retrieves the value of watermark set by the SET_WATERMARK Procedure
<a href="#">SET_WATERMARK Procedure</a> on page 23-50	Used for Oracle Streams AQ notification to specify and limit memory use

## Propagation Subprograms

**Table 23–7 Propagation Subprograms**

<b>Subprograms</b>	<b>Description</b>
<a href="#">ALTER_PROPAGATION_SCHEDULE Procedure</a> on page 23-18	Alters parameters for a propagation schedule
<a href="#">DISABLE_PROPAGATION_SCHEDULE Procedure</a> on page 23-31	Disables a propagation schedule
<a href="#">ENABLE_PROPAGATION_SCHEDULE Procedure</a> on page 23-37	Enables a previously disabled propagation schedule
<a href="#">SCHEDULE_PROPAGATION Procedure</a> on page 23-48	Schedules propagation of messages from a queue to a destination identified by a specific database link
<a href="#">UNSCHEDULE_PROPAGATION Procedure</a> on page 23-53	Unscheduled previously scheduled propagation of messages from a queue to a destination identified by a specific database link
<a href="#">VERIFY_QUEUE_TYPES Procedure</a> on page 23-54	Verifies that the source and destination queues have identical types

## Oracle Streams AQ Agent Subprograms

**Table 23–8 Oracle Streams AQ Agent Subprograms**

<b>Subprograms</b>	<b>Description</b>
<a href="#">ALTER_AQ_AGENT Procedure</a> on page 23-17	Alters an agent registered for Oracle Streams AQ Internet access, and an Oracle Streams AQ agent that accesses secure queues
<a href="#">CREATE_AQ_AGENT Procedure</a> on page 23-22	Registers an agent for Oracle Streams AQ Internet access using HTTP/SMTP protocols, and creates an Oracle Streams AQ agent to access secure queues
<a href="#">DISABLE_DB_ACCESS Procedure</a> on page 23-30	Revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent
<a href="#">DROP_AQ_AGENT Procedure</a> on page 23-32	Drops an agent that was previously registered for Oracle Streams AQ Internet access
<a href="#">ENABLE_DB_ACCESS Procedure</a> on page 23-35	Grants an Oracle Streams AQ Internet agent the privileges of a specific database user

## Alias Subprograms

**Table 23–9** *Alias Subprograms*

<b>Subprograms</b>	<b>Description</b>
<a href="#">ADD_ALIAS_TO_LDAP Procedure</a> on page 23-15	Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP
<a href="#">DEL_ALIAS_FROM_LDAP Procedure</a> on page 23-29	Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP



## Summary of DBMS\_AQADM Subprograms

**Table 23–10 DBMS\_AQADM Package Subprograms**

Subprograms	Description
<a href="#">ADD_ALIAS_TO_LDAP Procedure</a> on page 23-15	Creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP
<a href="#">ADD_SUBSCRIBER Procedure</a> on page 23-16	Adds a default subscriber to a queue
<a href="#">ALTER_AQ_AGENT Procedure</a> on page 23-17	Alters an agent registered for Oracle Streams AQ Internet access, and an Oracle Streams AQ agent that accesses secure queues
<a href="#">ALTER_PROPAGATION_SCHEDULE Procedure</a> on page 23-18	Alters parameters for a propagation schedule
<a href="#">ALTER_QUEUE Procedure</a> on page 23-19	Alters existing properties of a queue
<a href="#">ALTER_QUEUE_TABLE Procedure</a> on page 23-20	Alters the existing properties of a queue table
<a href="#">ALTER_SUBSCRIBER Procedure</a> on page 23-21	Alters existing properties of a subscriber to a specified queue
<a href="#">CREATE_AQ_AGENT Procedure</a> on page 23-22	Registers an agent for Oracle Streams AQ Internet access using HTTP/SMTP protocols, and creates an Oracle Streams AQ agent to access secure queues
<a href="#">CREATE_NP_QUEUE Procedure</a> on page 23-23	Creates a nonpersistent RAW queue
<a href="#">CREATE_QUEUE Procedure</a> on page 23-24	Creates a queue in the specified queue table
<a href="#">CREATE_QUEUE_TABLE Procedure</a> on page 23-26	Creates a queue table for messages of a predefined type
<a href="#">DEL_ALIAS_FROM_LDAP Procedure</a> on page 23-29	Drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP
<a href="#">DISABLE_DB_ACCESS Procedure</a> on page 23-30	Revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent
<a href="#">DISABLE_PROPAGATION_SCHEDULE Procedure</a> on page 23-31	Disables a propagation schedule
<a href="#">DROP_AQ_AGENT Procedure</a> on page 23-32	Drops an agent that was previously registered for Oracle Streams AQ Internet access
<a href="#">DROP_QUEUE Procedure</a> on page 23-33	Drops an existing queue
<a href="#">DROP_QUEUE_TABLE Procedure</a> on page 23-34	Drops an existing queue table
<a href="#">ENABLE_DB_ACCESS Procedure</a> on page 23-35	Grants an Oracle Streams AQ Internet agent the privileges of a specific database user
<a href="#">ENABLE_JMS_TYPES Procedure</a> on page 23-36	A precondition for the enqueue of JMS types and XML types

**Table 23–10 (Cont.) DBMS\_AQADM Package Subprograms**

<b>Subprograms</b>	<b>Description</b>
<a href="#">ENABLE_PROPAGATION_SCHEDULE Procedure</a> on page 23-37	Enables a previously disabled propagation schedule
<a href="#">GET_WATERMARK Procedure</a> on page 23-38	Retrieves the value of watermark set by the <a href="#">SET_WATERMARK Procedure</a>
<a href="#">GRANT_QUEUE_PRIVILEGE Procedure</a> on page 23-39	Grants privileges on a queue to users and roles
<a href="#">GRANT_SYSTEM_PRIVILEGE Procedure</a> on page 23-40	Grants Oracle Streams AQ system privileges to users and roles
<a href="#">MIGRATE_QUEUE_TABLE Procedure</a> on page 23-41	Upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table
<a href="#">PURGE_QUEUE_TABLE Procedure</a> on page 23-42	Purges messages from queue tables
<a href="#">QUEUE_SUBSCRIBERS Function</a> on page 23-44	Returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type <code>DBMS_AQADM.AQ\$_subscriber_list_t</code>
<a href="#">REMOVE_SUBSCRIBER Procedure</a> on page 23-45	Removes a default subscriber from a queue
<a href="#">REVOKE_QUEUE_PRIVILEGE Procedure</a> on page 23-46	Revokes privileges on a queue from users and roles
<a href="#">REVOKE_SYSTEM_PRIVILEGE Procedure</a> on page 23-47	Revokes Oracle Streams AQ system privileges from users and roles
<a href="#">SCHEDULE_PROPAGATION Procedure</a> on page 23-48	Schedules propagation of messages from a queue to a destination identified by a specific database link
<a href="#">SET_WATERMARK Procedure</a> on page 23-50	Used for Oracle Streams AQ notification to specify and limit memory use
<a href="#">START_QUEUE Procedure</a> on page 23-51	Enables the specified queue for enqueueing or dequeuing
<a href="#">STOP_QUEUE Procedure</a> on page 23-52	Disables enqueueing or dequeuing on the specified queue
<a href="#">UNSCHEDULE_PROPAGATION Procedure</a> on page 23-53	Unscheduled previously scheduled propagation of messages from a queue to a destination identified by a specific database link
<a href="#">VERIFY_QUEUE_TYPES Procedure</a> on page 23-54	Verifies that the source and destination queues have identical types

## ADD\_ALIAS\_TO\_LDAP Procedure

This procedure creates an alias for a queue, agent, or a JMS ConnectionFactory in LDAP. The alias will be placed directly under the database server's distinguished name in LDAP hierarchy.

### Syntax

```
DBMS_AQADM.ADD_ALIAS_TO_LDAP (
  alias          IN VARCHAR2,
  obj_location  IN VARCHAR2);
```

### Parameters

**Table 23–11 ADD\_ALIAS\_TO\_LDAP Procedure Parameters**

Parameter	Description
alias	Name of the alias. Example: west_shipping.
obj_location	The distinguished name of the object (queue, agent or connection factory) to which alias refers.

### Usage Notes

This method can be used to create aliases for queues, agents, and JMS ConnectionFactory objects. These object must exist before the alias is created. These aliases can be used for JNDI lookup in JMS and Oracle Streams AQ Internet access.

## ADD\_SUBSCRIBER Procedure

This procedure adds a default subscriber to a queue.

### Syntax

```
DBMS_AQADM.ADD_SUBSCRIBER (
  queue_name      IN      VARCHAR2,
  subscriber      IN      sys.aq$_agent,
  rule            IN      VARCHAR2 DEFAULT NULL,
  transformation  IN      VARCHAR2 DEFAULT NULL,
  queue_to_queue  IN      BOOLEAN DEFAULT FALSE,
  delivery_mode   IN      PLS_INTEGER DEFAULT DBMS_AQADM.PERSISTENT);
```

### Parameters

**Table 23–12 ADD\_SUBSCRIBER Procedure Parameters**

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being defined.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. A rule is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Currently supported message properties are <code>priority</code> and <code>corrid</code> .  To specify rules on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with <code>tab.user_data</code> as a qualifier to indicate the specific column of the queue table that stores the payload. The rule parameter cannot exceed 4000 characters.
transformation	Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue.
queue_to_queue	If TRUE, propagation is from queue-to-queue.
delivery_mode	The administrator may specify one of <code>DBMS_AQADM.PERSISTENT</code> , <code>DBMS_AQADM.BUFFERED</code> , or <code>DBMS_AQADM.PERSISTENT_OR_BUFFERED</code> for the delivery mode of the messages the subscriber is interested in. This parameter will not be modifiable by <code>ALTER_SUBSCRIBER</code> .

### Usage Notes

A program can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation only succeeds on queues that allow multiple consumers. This operation takes effect immediately, and the containing transaction is committed. Enqueue requests that are executed after the completion of this call will reflect the new behavior.

Any string within the rule must be quoted:

```
rule => 'PRIORITY <= 3 AND CORRID = ''FROM JAPAN'''
```

Note that these are all single quotation marks.

## ALTER\_AQ\_AGENT Procedure

This procedure alters an agent registered for Oracle Streams AQ Internet access. It is also used to alter an Oracle Streams AQ agent that accesses secure queues.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

### Syntax

```
DBMS_AQADM.ALTER_AQ_AGENT (
  agent_name           IN VARCHAR2,
  certificate_location IN VARCHAR2 DEFAULT NULL,
  enable_http         IN BOOLEAN DEFAULT FALSE,
  enable_smtp         IN BOOLEAN DEFAULT FALSE,
  enable_anyp         IN BOOLEAN DEFAULT FALSE )
```

### Parameters

**Table 23–13 ALTER\_AQ\_AGENT Procedure Parameters**

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
certification_location	Agent's certificate location in LDAP (default is NULL). If the agent is allowed to access Oracle Streams AQ through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required.
enable_http	TRUE means the agent can access Oracle Streams AQ through HTTP. FALSE means the agent cannot access Oracle Streams AQ through HTTP.
enable_smtp	TRUE means the agent can access Oracle Streams AQ through SMTP (e-mail). FALSE means the agent cannot access Oracle Streams AQ through SMTP.
enable_anyp	TRUE means the agent can access Oracle Streams AQ through any protocol (HTTP or SMTP).

## ALTER\_PROPAGATION\_SCHEDULE Procedure

This procedure alters parameters for a propagation schedule.

### Syntax

```
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE (
  queue_name          IN    VARCHAR2,
  destination         IN    VARCHAR2 DEFAULT NULL,
  duration            IN    NUMBER  DEFAULT NULL,
  next_time           IN    VARCHAR2 DEFAULT NULL,
  latency             IN    NUMBER  DEFAULT 60,
  destination_queue   IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 23–14 ALTER\_PROPAGATION\_SCHEDULE Procedure Parameters**

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as <code>SYSDATE + 1 - duration/86400</code> .
latency	<p>Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. The default value is 60. Caution: if latency is not specified for this call, then latency will over-write any existing value with the default value.</p> <p>For example, if the latency is 60 seconds and there are no messages to be propagated during the propagation window, then messages from that queue for the destination are not propagated for at least 60 more seconds. It will be at least 60 seconds before the queue will be checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue will not be checked for 10 minutes and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination and as soon as a message is enqueued it will be propagated.</p>
destination_queue	Name of the target queue to which messages are to be propagated in the form of a dblink

## ALTER\_QUEUE Procedure

This procedure alters existing properties of a queue. The parameters `max_retries`, `retention_time`, and `retry_delay` are not supported for nonpersistent queues.

### Syntax

```
DBMS_AQADM.ALTER_QUEUE (
    queue_name      IN      VARCHAR2,
    max_retries     IN      NUMBER   DEFAULT NULL,
    retry_delay     IN      NUMBER   DEFAULT NULL,
    retention_time  IN      NUMBER   DEFAULT NULL,
    auto_commit     IN      BOOLEAN  DEFAULT TRUE,
    comment         IN      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 23–15 ALTER\_QUEUE Procedure Parameters**

Parameter	Description
<code>queue_name</code>	Name of the queue that is to be altered
<code>max_retries</code>	Limits the number of times a dequeue with REMOVE mode can be attempted on a message. The maximum value of <code>max_retries</code> is $2^{31}-1$ .  A message is moved to an exception queue if <code>RETRY_COUNT</code> is greater than <code>MAX_RETRIES</code> . <code>RETRY_COUNT</code> is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including ALTER SYSTEM KILL SESSION) or SHUTDOWN ABORT on the instance, then <code>RETRY_COUNT</code> is not incremented.  Note that <code>max_retries</code> is supported for all single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
<code>retry_delay</code>	Delay time in seconds before this message is scheduled for processing again after an application rollback. The default is NULL, which means that the value will not be altered.  Note that <code>retry_delay</code> is supported for single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
<code>retention_time</code>	Retention time in seconds for which a message is retained in the queue table after being dequeued. The default is NULL, which means that the value will not be altered.
<code>auto_commit</code>	TRUE causes the current transaction, if any, to commit before the ALTER_QUEUE operation is carried out. The ALTER_QUEUE operation become persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.  Caution: This parameter has been deprecated.
<code>comment</code>	User-specified description of the queue. This user comment is added to the queue catalog. The default value is NULL, which means that the value will not be changed.

## ALTER\_QUEUE\_TABLE Procedure

This procedure alters the existing properties of a queue table.

### Syntax

```
DBMS_AQADM.ALTER_QUEUE_TABLE (
  queue_table      IN  VARCHAR2,
  comment          IN  VARCHAR2      DEFAULT NULL,
  primary_instance IN  BINARY_INTEGER DEFAULT NULL,
  secondary_instance IN BINARY_INTEGER DEFAULT NULL);
```

### Parameters

**Table 23–16 ALTER\_QUEUE\_TABLE Procedure Parameters**

Parameter	Description
queue_table	Name of a queue table to be created.
comment	Modifies the user-specified description of the queue table. This user comment is added to the queue catalog. The default value is NULL which means that the value will not be changed.
primary_instance	This is the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table will be done in this instance. The default value is NULL, which means that the current value will not be changed.
secondary_instance	The queue table fails over to the secondary instance if the primary instance is not available. The default value is NULL, which means that the current value will not be changed.



## ALTER\_SUBSCRIBER Procedure

This procedure alters existing properties of a subscriber to a specified queue. Only the rule can be altered.

### Syntax

```
DBMS_AQADM.ALTER_SUBSCRIBER (
  queue_name      IN      VARCHAR2,
  subscriber      IN      sys.aq$_agent,
  rule            IN      VARCHAR2
  transformation  IN      VARCHAR2);
```

### Parameters

**Table 23–17 ALTER\_SUBSCRIBER Procedure Parameters**

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being altered. See " <a href="#">AQ\$_AGENT Type</a> " on page 243-3.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. The rule parameter cannot exceed 4000 characters. To eliminate the rule, set the rule parameter to NULL.
transformation	Specifies a transformation that will be applied when this subscriber dequeues the message. The source type of the transformation must match the type of the queue. If the subscriber is remote, then the transformation is applied before propagation to the remote queue.

### Usage Notes

This procedure alters both the rule and the transformation for the subscriber. If you want to retain the existing value for either of them, you must specify its old value. The current values for rule and transformation for a subscriber can be obtained from the *schema.AQ\$queue\_table\_R* and *schema.AQ\$queue\_table\_S* views.

## CREATE\_AQ\_AGENT Procedure

This procedure registers an agent for Oracle Streams AQ Internet access using HTTP/SMTP protocols. It is also used to create an Oracle Streams AQ agent to access secure queues.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

### Syntax

```
DBMS_AQADM.CREATE_AQ_AGENT (
  agent_name          IN VARCHAR2,
  certificate_location IN VARCHAR2 DEFAULT NULL,
  enable_http        IN BOOLEAN DEFAULT FALSE,
  enable_smtp        IN BOOLEAN DEFAULT FALSE,
  enable_anyp        IN BOOLEAN DEFAULT FALSE )
```

### Parameters

**Table 23–18** CREATE\_AQ\_AGENT Procedure Parameters

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
certification_location	Agent's certificate location in LDAP (default is NULL). If the agent is allowed to access Oracle Streams AQ through SMTP, then its certificate must be registered in LDAP. For access through HTTP, the certificate location is not required.
enable_http	TRUE means the agent can access Oracle Streams AQ through HTTP. FALSE means the agent cannot access Oracle Streams AQ through HTTP.
enable_smtp	TRUE means the agent can access Oracle Streams AQ through SMTP (e-mail). FALSE means the agent cannot access Oracle Streams AQ through SMTP.
enable_anyp	TRUE means the agent can access Oracle Streams AQ through any protocol (HTTP or SMTP).

### Usage Notes

The `SYS.AQ$INTERNET_USERS` view has a list of all Oracle Streams AQ Internet agents.

## CREATE\_NP\_QUEUE Procedure

---

**Note:** nonpersistent queues are deprecated as of Release 10gR2. Oracle recommends using buffered messaging.

---

This procedure creates a nonpersistent RAW queue.

### Syntax

```
DBMS_AQADM.CREATE_NP_QUEUE (
  queue_name          IN          VARCHAR2,
  multiple_consumers IN          BOOLEAN DEFAULT FALSE,
  comment             IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 23–19 CREATE\_NP\_QUEUE Procedure Parameters**

Parameter	Description
queue_name	Name of the nonpersistent queue that is to be created. The name must be unique within a schema and must follow object name guidelines in <i>Oracle Database SQL Language Reference</i> .
multiple_consumers	FALSE means queues created in the table can only have one consumer for each message. This is the default. TRUE means queues created in the table can have multiple consumers for each message.  Note that this parameter is distinguished at the queue level, because a nonpersistent queue does not inherit this characteristic from any user-created queue table.
comment	User-specified description of the queue. This user comment is added to the queue catalog.

### Usage Notes

The queue may be either single-consumer or multiconsumer queue. All queue names must be unique within a schema. The queues are created in a 8.1-compatible or higher system-created queue table (AQ\$\_MEM\_SC or AQ\$\_MEM\_MC) in the same schema as that specified by the queue name.

If the queue name does not specify a schema name, the queue is created in the login user's schema. After a queue is created with `CREATE_NP_QUEUE`, it can be enabled by calling `START_QUEUE`. By default, the queue is created with both enqueue and dequeue disabled.

You cannot dequeue from a nonpersistent queue. The only way to retrieve a message from a nonpersistent queue is by using the OCI notification mechanism. You cannot invoke the `LISTEN` call on a nonpersistent queue.

## CREATE\_QUEUE Procedure

This procedure creates a queue in the specified queue table.

### Syntax

```
DBMS_AQADM.CREATE_QUEUE (
  queue_name      IN      VARCHAR2,
  queue_table     IN      VARCHAR2,
  queue_type      IN      BINARY_INTEGER DEFAULT NORMAL_QUEUE,
  max_retries     IN      NUMBER          DEFAULT NULL,
  retry_delay     IN      NUMBER          DEFAULT 0,
  retention_time  IN      NUMBER          DEFAULT 0,
  dependency_tracking IN    BOOLEAN        DEFAULT FALSE,
  comment         IN      VARCHAR2        DEFAULT NULL,
  auto_commit     IN      BOOLEAN        DEFAULT TRUE);
```

### Parameters

**Table 23–20 CREATE\_QUEUE Procedure Parameters**

Parameter	Description
queue_name	Name of the queue that is to be created. The name must be unique within a schema and must follow object name guidelines in <i>Oracle Database SQL Language Reference</i> with regard to reserved characters.
queue_table	Name of the queue table that will contain the queue.
queue_type	Specifies whether the queue being created is an exception queue or a normal queue. <code>NORMAL_QUEUE</code> means the queue is a normal queue. This is the default. <code>EXCEPTION_QUEUE</code> means it is an exception queue. Only the dequeue operation is allowed on the exception queue.
max_retries	Limits the number of times a dequeue with the <code>REMOVE</code> mode can be attempted on a message. The maximum value of <code>max_retries</code> is $2^{31} - 1$ .  A message is moved to an exception queue if <code>RETRY_COUNT</code> is greater than <code>MAX_RETRIES</code> . <code>RETRY_COUNT</code> is incremented when the application issues a rollback after executing the dequeue. If a dequeue transaction fails because the server process dies (including <code>ALTER SYSTEM KILL SESSION</code> ) or <code>SHUTDOWN ABORT</code> on the instance, then <code>RETRY_COUNT</code> is not incremented.  Note that <code>max_retries</code> is supported for all single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retry_delay	Delay time, in seconds, before this message is scheduled for processing again after an application rollback.  The default is 0, which means the message can be retried as soon as possible. This parameter has no effect if <code>max_retries</code> is set to 0. Note that <code>retry_delay</code> is supported for single consumer queues and 8.1-compatible or higher multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retention_time	Number of seconds for which a message is retained in the queue table after being dequeued from the queue. <code>INFINITE</code> means the message is retained forever. <code>NUMBER</code> is the number of seconds for which to retain the messages. The default is 0, no retention.

**Table 23–20 (Cont.) CREATE\_QUEUE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
dependency_tracking	Reserved for future use. FALSE is the default. TRUE is not permitted in this release.
comment	User-specified description of the queue. This user comment is added to the queue catalog.
auto_commit	TRUE causes the current transaction, if any, to commit before the CREATE_QUEUE operation is carried out. The CREATE_QUEUE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.  Caution: This parameter has been deprecated.

**Usage Notes**

All queue names must be unique within a schema. After a queue is created with CREATE\_QUEUE, it can be enabled by calling START\_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

## CREATE\_QUEUE\_TABLE Procedure

This procedure creates a queue table for messages of a predefined type.

### Syntax

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
  queue_table          IN      VARCHAR2,
  queue_payload_type  IN      VARCHAR2,
  storage_clause       IN      VARCHAR2          DEFAULT NULL,
  sort_list           IN      VARCHAR2          DEFAULT NULL,
  multiple_consumers  IN      BOOLEAN           DEFAULT FALSE,
  message_grouping    IN      BINARY_INTEGER   DEFAULT NONE,
  comment             IN      VARCHAR2          DEFAULT NULL,
  auto_commit         IN      BOOLEAN           DEFAULT TRUE,
  primary_instance    IN      BINARY_INTEGER   DEFAULT 0,
  secondary_instance  IN      BINARY_INTEGER   DEFAULT 0,
  compatible          IN      VARCHAR2          DEFAULT NULL,
  secure              IN      BOOLEAN           DEFAULT FALSE);
```

### Parameters

**Table 23–21 CREATE\_QUEUE\_TABLE Procedure Parameters**

Parameter	Description
queue_table	Name of a queue table to be created
queue_payload_type	Type of the user data stored. See <a href="#">Type Name</a> on page 22-5 for valid values for this parameter.
storage_clause	Storage parameter. The storage parameter is included in the CREATE TABLE statement when the queue table is created. The storage_clause argument can take any text that can be used in a standard CREATE TABLE storage_clause argument. The storage parameter can be made up of any combinations of the following parameters: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLESPACE, LOB, and a table storage clause.  If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See <i>Oracle Database SQL Language Reference</i> for the usage of these parameters.

**Table 23–21 (Cont.) CREATE\_QUEUE\_TABLE Procedure Parameters**

Parameter	Description
sort_list	<p>The columns to be used as the sort key in ascending order. This parameter has the following format:</p> <p><i>'sort_column_1, sort_column_2'</i></p> <p>The allowed column names are <code>priority</code> and <code>enq_time</code>. If both columns are specified, then <code>sort_column_1</code> defines the most significant order.</p> <p>After a queue table is created with a specific ordering mechanism, all queues in the queue table inherit the same defaults. The order of a queue table cannot be altered after the queue table has been created.</p> <p>If no sort list is specified, then all the queues in this queue table are sorted by the enqueue time in ascending order. This order is equivalent to FIFO order.</p> <p>Even with the default ordering defined, a dequeuer is allowed to choose a message to dequeue by specifying its <code>msgid</code> or <code>correlation.msgid</code>, <code>correlation</code>, and <code>sequence_deviation</code> take precedence over the default dequeuing order, if they are specified.</p>
multiple_consumers	<p><code>FALSE</code> means queues created in the table can only have one consumer for each message. This is the default. <code>TRUE</code> means queues created in the table can have multiple consumers for each message.</p>
message_grouping	<p>Message grouping behavior for queues created in the table. <code>NONE</code> means each message is treated individually. <code>TRANSACTIONAL</code> means messages enqueued as part of one transaction are considered part of the same group and can be dequeued as a group of related messages.</p>
comment	<p>User-specified description of the queue table. This user comment is added to the queue catalog.</p>
auto_commit	<p><code>TRUE</code> causes the current transaction, if any, to commit before the <code>CREATE_QUEUE_TABLE</code> operation is carried out. The <code>CREATE_QUEUE_TABLE</code> operation becomes persistent when the call returns. This is the default. <code>FALSE</code> means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.</p> <p>Note: This parameter has been deprecated.</p>
primary_instance	<p>The primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance.</p> <p>The default value for primary instance is 0, which means queue monitor scheduling and propagation will be done in any available instance.</p>
secondary_instance	<p>The queue table fails over to the secondary instance if the primary instance is not available. The default value is 0, which means that the queue table will fail over to any available instance.</p>
compatible	<p>The lowest database version with which the queue is compatible. Currently the possible values are either 8.0, 8.1, or 10.0. If the database is in 10.1-compatible mode, the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, the default value is 8.1. If the database is in 8.0 compatible mode, the default value is 8.0.</p>

**Table 23–21 (Cont.) CREATE\_QUEUE\_TABLE Procedure Parameters**

Parameter	Description
<code>secure</code>	This parameter must be set to <code>TRUE</code> if you want to use the queue table for secure queues. Secure queues are queues for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot perform queue operations on a secure queue, unless they are configured as secure queue users.

## Usage Notes

The sort keys for dequeue ordering, if any, must be defined at table creation time. The following objects are created at this time:

- `aq$_queue_table_name_e`, a default exception queue associated with the queue table
- `aq$queue_table_name`, a read-only view, which is used by Oracle Streams AQ applications for querying queue data
- `aq$_queue_table_name_t`, an index (or an index organized table (IOT) in the case of multiple consumer queues) for the queue monitor operations
- `aq$_queue_table_name_i`, an index (or an index organized table in the case of multiple consumer queues) for dequeue operations

For 8.1-compatible or higher queue tables, the following index-organized tables are created:

- `aq$_queue_table_name_s`, a table for storing information about the subscribers
- `aq$_queue_table_name_r`, a table for storing information about rules on subscriptions

`aq$_queue_table_name_h`, an index-organized table for storing the dequeue history data

CLOB, BLOB, and BFILE are valid attributes for Oracle Streams AQ object type payloads. However, only CLOB and BLOB can be propagated using Oracle Streams AQ propagation in Oracle8i release 8.1.5 or later. See the *Oracle Streams Advanced Queuing User's Guide* for more information.

The default value of the `compatible` parameter depends on the database compatibility mode in the `init.ora`. If the database is in 10.1-compatible mode, the default value is 10.0. If the database is in 8.1-compatible or 9.2-compatible mode, the default value is 8.1. If the database is in 8.0 compatible mode, the default value is 8.0

You can specify and modify the `primary_instance` and `secondary_instance` only in 8.1-compatible or higher mode. You cannot specify a secondary instance unless there is a primary instance.



## DEL\_ALIAS\_FROM\_LDAP Procedure

This procedure drops an alias for a queue, agent, or JMS ConnectionFactory in LDAP.

### Syntax

```
DBMS_AQ.DEL_ALIAS_FROM_LDAP(  
    alias IN VARCHAR2);
```

### Parameters

**Table 23–22** *DEL\_ALIAS\_FROM\_LDAP Procedure Parameters*

Parameter	Description
alias	The alias to be removed.

## DISABLE\_DB\_ACCESS Procedure

This procedure revokes the privileges of a specific database user from an Oracle Streams AQ Internet agent.

### Syntax

```
DBMS_AQADM.DISABLE_DB_ACCESS (  
  agent_name          IN VARCHAR2,  
  db_username         IN VARCHAR2)
```

### Parameters

**Table 23–23** *DISABLE\_DB\_ACCESS Procedure Parameters*

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
db_username	Specifies the database user whose privileges are to be revoked from the Oracle Streams AQ Internet agent.

### Usage Notes

The Oracle Streams AQ Internet agent should have been previously granted those privileges using the [ENABLE\\_DB\\_ACCESS Procedure](#).

## DISABLE\_PROPAGATION\_SCHEDULE Procedure

This procedure disables a propagation schedule.

### Syntax

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
  queue_name          IN  VARCHAR2,
  destination         IN  VARCHAR2 DEFAULT NULL,
  destination_queue   IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 23–24** *DISABLE\_PROPAGATION\_SCHEDULE Procedure Parameters*

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is <code>NULL</code> , then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a <code>dblink</code> .

## DROP\_AQ\_AGENT Procedure

This procedure drops an agent that was previously registered for Oracle Streams AQ Internet access.

### Syntax

```
DBMS_AQADM.DROP_AQ_AGENT (  
    agent_name          IN VARCHAR2)
```

### Parameters

**Table 23–25** *DROP\_AQ\_AGENT Procedure Parameters*

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent

## DROP\_QUEUE Procedure

This procedure drops an existing queue.

### Syntax

```
DBMS_AQADM.DROP_QUEUE (
  queue_name      IN    VARCHAR2,
  auto_commit     IN    BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 23–26 DROP\_QUEUE Procedure Parameters**

Parameter	Description
queue_name	Name of the queue that is to be dropped.
auto_commit	TRUE causes the current transaction, if any, to commit before the DROP_QUEUE operation is carried out. The DROP_QUEUE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.  Caution: This parameter has been deprecated.

### Usage Notes

DROP\_QUEUE is not allowed unless STOP\_QUEUE has been called to disable the queue for both enqueueing and dequeuing. All the queue data is deleted as part of the drop operation.

## DROP\_QUEUE\_TABLE Procedure

This procedure drops an existing queue table.

### Syntax

```
DBMS_AQADM.DROP_QUEUE_TABLE (  
    queue_table      IN    VARCHAR2,  
    force            IN    BOOLEAN DEFAULT FALSE,  
    auto_commit      IN    BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 23–27** DROP\_QUEUE\_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be dropped.
force	FALSE means the operation does not succeed if there are any queues in the table. This is the default. TRUE means all queues in the table are stopped and dropped automatically.
auto_commit	TRUE causes the current transaction, if any, to commit before the DROP_QUEUE_TABLE operation is carried out. The DROP_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE means the operation is part of the current transaction and becomes persistent only when the caller enters a commit.  Caution: This parameter has been deprecated.

### Usage Notes

All the queues in a queue table must be stopped and dropped before the queue table can be dropped. You must do this explicitly unless the `force` option is used, in which case this is done automatically.

## ENABLE\_DB\_ACCESS Procedure

This procedure grants an Oracle Streams AQ Internet agent the privileges of a specific database user.

### Syntax

```
DBMS_AQADM.ENABLE_DB_ACCESS (
  agent_name          IN VARCHAR2,
  db_username        IN VARCHAR2)
```

### Parameters

**Table 23–28** *ENABLE\_DB\_ACCESS Procedure Parameters*

Parameter	Description
agent_name	Specifies the username of the Oracle Streams AQ Internet agent.
db_username	Specified the database user whose privileges are to be granted to the Oracle Streams AQ Internet agent.

### Usage Notes

The Oracle Streams AQ Internet agent should have been previously created using the [CREATE\\_AQ\\_AGENT Procedure](#).

For secure queues, the sender and receiver agent of the message must be mapped to the database user performing the enqueue or dequeue operation.

**See Also:** *Oracle Streams Concepts and Administration* for information about secure queues

The `SYS.AQ$INTERNET_USERS` view has a list of all Oracle Streams AQ Internet agents and the names of the database users whose privileges are granted to them.

## ENABLE\_JMS\_TYPES Procedure

Enqueue of JMS types and XML types does not work with Oracle Streams Sys.Anydata queues unless you call this procedure after DBMS\_STREAMS\_ADM.SET\_UP\_QUEUE. Enabling an Oracle Streams queue for these types may affect import/export of the queue table.

### Syntax

```
DBMS_AQADM.ENABLE_JMS_TYPES (  
    queue_table IN VARCHAR2);
```

### Parameters

**Table 23–29** *ENABLE\_JMS\_TYPES Procedure Parameters*

Parameter	Description
queue_table	Specifies name of the queue table to be enabled for JMS and XML types.



## ENABLE\_PROPAGATION\_SCHEDULE Procedure

This procedure enables a previously disabled propagation schedule.

### Syntax

```
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE (
    queue_name          IN   VARCHAR2,
    destination         IN   VARCHAR2 DEFAULT NULL,
    destination_queue   IN   VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 23–30** *ENABLE\_PROPAGATION\_SCHEDULE Procedure Parameters*

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a dblink

## GET\_WATERMARK Procedure

This procedure retrieves the value of watermark set by SET\_WATERMARK.

### Syntax

```
DBMS_AQADM.GET_WATERMARK (  
    wmvalue      OUT      NUMBER);
```

### Parameters

**Table 23–31** GET\_WATERMARK Procedure Parameter

Parameter	Description
wmvalue	Watermark value in megabytes.

## GRANT\_QUEUE\_PRIVILEGE Procedure

This procedure grants privileges on a queue to users and roles. The privileges are ENQUEUE or DEQUEUE. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

### Syntax

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
  privilege      IN   VARCHAR2,
  queue_name     IN   VARCHAR2,
  grantee        IN   VARCHAR2,
  grant_option   IN   BOOLEAN := FALSE);
```

### Parameters

**Table 23–32 GRANT\_QUEUE\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The Oracle Streams AQ queue privilege to grant. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
grant_option	Specifies if the access privilege is granted with the GRANT option or not. If the privilege is granted with the GRANT option, then the grantee is allowed to use this procedure to grant the access privilege to other users or roles, regardless of the ownership of the queue table. The default is FALSE.

## GRANT\_SYSTEM\_PRIVILEGE Procedure

This procedure grants Oracle Streams AQ system privileges to users and roles. The privileges are `ENQUEUE_ANY`, `DEQUEUE_ANY`, and `MANAGE_ANY`. Initially, only `SYS` and `SYSTEM` can use this procedure successfully.

### Syntax

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
  privilege      IN   VARCHAR2,
  grantee        IN   VARCHAR2,
  admin_option   IN   BOOLEAN := FALSE);
```

### Parameters

**Table 23–33 GRANT\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
<code>privilege</code>	The Oracle Streams AQ system privilege to grant. The options are <code>ENQUEUE_ANY</code> , <code>DEQUEUE_ANY</code> , and <code>MANAGE_ANY</code> . <code>ENQUEUE_ANY</code> means users granted this privilege are allowed to enqueue messages to any queues in the database. <code>DEQUEUE_ANY</code> means users granted this privilege are allowed to dequeue messages from any queues in the database. <code>MANAGE_ANY</code> means users granted this privilege are allowed to run <code>DBMS_AQADM</code> calls on any schemas in the database.
<code>grantee</code>	Grantee(s). The grantee(s) can be a user, a role, or the <code>PUBLIC</code> role.
<code>admin_option</code>	Specifies if the system privilege is granted with the <code>ADMIN</code> option or not. If the privilege is granted with the <code>ADMIN</code> option, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles. The default is <code>FALSE</code> .

## MIGRATE\_QUEUE\_TABLE Procedure

This procedure upgrades an 8.0-compatible queue table to an 8.1-compatible or higher queue table, or downgrades an 8.1-compatible or higher queue table to an 8.0-compatible queue table.

### Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE (  
    queue_table IN VARCHAR2,  
    compatible  IN VARCHAR2);
```

### Parameters

**Table 23–34** *MIGRATE\_QUEUE\_TABLE Procedure Parameters*

Parameter	Description
queue_table	Specifies name of the queue table to be migrated.
compatible	Set this to 8.1 to upgrade an 8.0-compatible queue table, or set this to 8.0 to downgrade an 8.1-compatible queue table.

## PURGE\_QUEUE\_TABLE Procedure

This procedure purges messages from queue tables. You can perform various purge operations on both single-consumer and multiconsumer queue tables for persistent and buffered messages.

### Syntax

```
DBMS_AQADM.PURGE_QUEUE_TABLE (
  queue_table      IN  VARCHAR2,
  purge_condition  IN  VARCHAR2,
  purge_options    IN  aq$_purge_options_t);
```

where type `aq$_purge_options_t` is described in [Chapter 243, "Oracle Streams AQ TYPES"](#).

### Parameters

**Table 23–35** *PURGE\_QUEUE\_TABLE Procedure Parameters*

Parameter	Description
<code>queue_table</code>	Specifies the name of the queue table to be purged.
<code>purge_condition</code>	<p>Specifies the purge condition to use when purging the queue table. The purge condition must be in the format of a SQL <code>WHERE</code> clause, and it is case-sensitive. The condition is based on the columns of <code>aq\$queue_table_name</code> view.</p> <p>When specifying the <code>purge_condition</code>, qualify the column names in <code>aq\$queue_table_name</code> view with <code>qtview</code>.</p> <p>To purge all queues in a queue table, set <code>purge_condition</code> to either <code>NULL</code> (a bare null word, no quotes) or <code>' '</code> (two single quotes).</p>
<code>purge_options</code>	<p>Type <code>aq\$_purge_options_t</code> contains a <code>block</code> parameter and a <code>delivery_mode</code> parameter.</p> <ul style="list-style-type: none"> <li>■ If <code>block</code> is <code>TRUE</code>, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueueers and dequeuers to block while the queue table is purged. The purge call always succeeds if <code>block</code> is <code>TRUE</code>. The default for <code>block</code> is <code>FALSE</code>. This will not block enqueueers and dequeuers, but it can cause the purge to fail with an error during high concurrency times.</li> <li>■ <code>delivery_mode</code> is used to specify whether <code>DBMS_AQADM.PERSISTENT</code>, <code>DBMS_AQADM.BUFFERED</code> or <code>DBMS_AQADM.PERSISTENT_OR_BUFFERED</code> types of messages are to be purged. You cannot implement arbitrary purge conditions if buffered messages have to be purged.</li> </ul>

### Usage Notes

- You can purge selected messages from the queue table by specifying a `purge_condition`. [Table 23–35](#) describes these parameters. Messages can be enqueued and dequeued from the queue table while the queue table is being purged.

- A trace file is generated in the `udump` destination when you run this procedure. It details what the procedure is doing.
- This procedure commits batches of messages in autonomous transactions. Several such autonomous transactions may get executed as a part of one `purge_queue_table` call depending on the number of messages in the queue table.

## QUEUE\_SUBSCRIBERS Function

This function returns the subscribers to an 8.0-compatible multiconsumer queue in the PL/SQL index by table collection type `DBMS_AQADM.AQ$_subscriber_list_t`. Each element of the collection is of type `sys.aq$_agent`. This functionality is provided for 8.1-compatible queues by the `AQ$queue_table_name_S` view.

### Syntax

```
DBMS_AQADM.QUEUE_SUBSCRIBERS (
    queue_name          IN          VARCHAR2);
RETURN aq$_subscriber_list_t IS
```

### Parameters

**Table 23–36** *QUEUE\_SUBSCRIBERS Function Parameters*

Parameter	Description
queue_name	Specifies the queue whose subscribers are to be printed.



## REMOVE\_SUBSCRIBER Procedure

This procedure removes a default subscriber from a queue. This operation takes effect immediately, and the containing transaction is committed. All references to the subscriber in existing messages are removed as part of the operation.

### Syntax

```
DBMS_AQADM.REMOVE_SUBSCRIBER (  
    queue_name      IN      VARCHAR2,  
    subscriber      IN      sys.aq$_agent);
```

### Parameters

**Table 23–37 REMOVE\_SUBSCRIBER Procedure Parameters**

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent who is being removed. See <a href="#">AQ\$_AGENT Type</a> on page 243-3.

## REVOKE\_QUEUE\_PRIVILEGE Procedure

This procedure revokes privileges on a queue from users and roles. The privileges are ENQUEUE or DEQUEUE.

### Syntax

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (  
  privilege      IN      VARCHAR2,  
  queue_name     IN      VARCHAR2,  
  grantee        IN      VARCHAR2);
```

### Parameters

**Table 23–38** *REVOKE\_QUEUE\_PRIVILEGE Procedure Parameters*

Parameter	Description
privilege	The Oracle Streams AQ queue privilege to revoke. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role. If the privilege has been propagated by the grantee through the GRANT option, then the propagated privilege is also revoked.

### Usage Notes

To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the GRANT option are revoked if the grantor's privileges are revoked.

## REVOKE\_SYSTEM\_PRIVILEGE Procedure

This procedure revokes Oracle Streams AQ system privileges from users and roles. The privileges are ENQUEUE\_ANY, DEQUEUE\_ANY and MANAGE\_ANY. The ADMIN option for a system privilege cannot be selectively revoked.

### Syntax

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE (
  privilege      IN  VARCHAR2,
  grantee        IN  VARCHAR2);
```

### Parameters

**Table 23–39 REVOKE\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The Oracle Streams AQ system privilege to revoke. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.

## SCHEDULE\_PROPAGATION Procedure

This procedure schedules propagation of messages from a queue to a destination identified by a specific database link.

### Syntax

```
DBMS_AQADM.SCHEDULE_PROPAGATION (
  queue_name          IN   VARCHAR2,
  destination         IN   VARCHAR2 DEFAULT NULL,
  start_time         IN   DATE      DEFAULT SYSDATE,
  duration            IN   NUMBER   DEFAULT NULL,
  next_time          IN   VARCHAR2 DEFAULT NULL,
  latency            IN   NUMBER   DEFAULT 60,
  destination_queue  IN   VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 23–40 SCHEDULE\_PROPAGATION Procedure Parameters**

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
start_time	Initial start time for the propagation window for messages from the source queue to the destination.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as SYSDATE + 1 - duration/86400.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued.  For example, if the latency is 60 seconds and there are no messages to be propagated during the propagation window, then messages from that queue for the destination are not propagated for at least 60 more seconds.  It is at least 60 seconds before the queue is checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue is not checked for 10 minutes, and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination. As soon as a message is enqueued, it is propagated.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a dblink

### Usage Notes

Messages may also be propagated to other queues in the same database by specifying a NULL destination. If a message has multiple recipients at the same destination in

either the same or different queues, the message is propagated to all of them at the same time.

## SET\_WATERMARK Procedure

This procedure is used for Oracle Streams AQ notification to specify and limit memory use.

### Syntax

```
DBMS_AQADM.SET_WATERMARK (  
    wmvalue      IN      NUMBER);
```

### Parameters

**Table 23–41** SET\_WATERMARK Procedure Parameter

Parameter	Description
wmvalue	Watermark value in megabytes.

## START\_QUEUE Procedure

This procedure enables the specified queue for enqueueing or dequeuing.

### Syntax

```
DBMS_AQADM.START_QUEUE (
    queue_name      IN      VARCHAR2,
    enqueue         IN      BOOLEAN DEFAULT TRUE,
    dequeue         IN      BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 23–42** *START\_QUEUE Procedure Parameters*

Parameter	Description
queue_name	Name of the queue to be enabled
enqueue	Specifies whether ENQUEUE should be enabled on this queue. TRUE means enable ENQUEUE. This is the default. FALSE means do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be enabled on this queue. TRUE means enable DEQUEUE. This is the default. FALSE means do not alter the current setting.

### Usage Notes

After creating a queue, the administrator must use `START_QUEUE` to enable the queue. The default is to enable it for both `ENQUEUE` and `DEQUEUE`. Only `dequeue` operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

## STOP\_QUEUE Procedure

This procedure disables enqueueing or dequeuing on the specified queue.

### Syntax

```
DBMS_AQADM.STOP_QUEUE (
  queue_name      IN  VARCHAR2,
  enqueue         IN  BOOLEAN DEFAULT TRUE,
  dequeue         IN  BOOLEAN DEFAULT TRUE,
  wait            IN  BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 23–43 STOP\_QUEUE Procedure Parameters**

Parameter	Description
queue_name	Name of the queue to be disabled
enqueue	Specifies whether ENQUEUE should be disabled on this queue. TRUE means disable ENQUEUE. This is the default. FALSE means do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be disabled on this queue. TRUE means disable DEQUEUE. This is the default. FALSE means do not alter the current setting.
wait	Specifies whether to wait for the completion of outstanding transactions. TRUE means wait if there are any outstanding transactions. In this state no new transactions are allowed to enqueue to or dequeue from this queue. FALSE means return immediately either with a success or an error.

### Usage Notes

By default, this call disables both ENQUEUE and DEQUEUE. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.



## UNSCHEDULE\_PROPAGATION Procedure

This procedure unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific database link.

### Syntax

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
  queue_name      IN  VARCHAR2,
  destination     IN  VARCHAR2 DEFAULT NULL
  destination_queue IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 23–44 UNSCHEDULE\_PROPAGATION Procedure Parameters**

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is <code>NULL</code> , then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
destination_queue	Name of the target queue to which messages are to be propagated in the form of a <code>dblink</code> .

## VERIFY\_QUEUE\_TYPES Procedure

This procedure verifies that the source and destination queues have identical types. The result of the verification is stored in the table `sys.aq$_message_types`, overwriting all previous output of this command.

### Syntax

```
DBMS_AQADM.VERIFY_QUEUE_TYPES (
    src_queue_name    IN    VARCHAR2,
    dest_queue_name   IN    VARCHAR2,
    destination       IN    VARCHAR2 DEFAULT NULL,
    rc                OUT   BINARY_INTEGER);
```

### Parameters

**Table 23–45** VERIFY\_QUEUE\_TYPES Procedure Parameters

Parameter	Description
<code>src_queue_name</code>	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
<code>dest_queue_name</code>	Name of the destination queue where messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
<code>destination</code>	Destination database link. Messages in the source queue for recipients at this destination are propagated. If it is <code>NULL</code> , then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
<code>rc</code>	Return code for the result of the procedure. If there is no error, and if the source and destination queue types match, then the result is 1. If they do not match, then the result is 0. If an Oracle error is encountered, then it is returned in <code>rc</code> .

The DBMS\_AQELM package provides subprograms to manage the configuration of Oracle Streams Advanced Queuing (AQ) asynchronous notification by e-mail and HTTP.

**See Also:** *Oracle Streams Advanced Queuing User's Guide* for detailed information about DBMS\_AQELM

This chapter contains the following topic:

- [Summary of DBMS\\_AQELM Subprograms](#)

## Summary of DBMS\_AQELM Subprograms

**Table 24–1** *DBMS\_ALERT Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">SET_MAILHOST Procedure</a> on page 24-3	Sets the host name for the SMTP server that the database will use to send out e-mail notifications
<a href="#">SET_MAILPORT Procedure</a> on page 24-4	Sets the port number for the SMTP server
<a href="#">SET_SENDFROM Procedure</a> on page 24-5	Sets the sent-from e-mail address

## SET\_MAILHOST Procedure

This procedure sets the host name for the SMTP server. The database uses this SMTP server host name to send out e-mail notifications.

### Syntax

```
DBMS_AQELM.SET_MAILHOST (  
    mailhost IN VARCHAR2);
```

### Parameters

**Table 24–2** *SET\_MAILHOST Procedure Parameters*

Parameter	Description
mailhost	SMTP server host name.

### Usage Notes

As part of the configuration for e-mail notifications, a user with AQ\_ADMINISTRATOR\_ROLE or with EXECUTE permissions on the DBMS\_AQELM package needs to set the host name before registering for e-mail notifications.

## SET\_MAILPORT Procedure

This procedure sets the port number for the SMTP server.

### Syntax

```
DBMS_AQELM.SET_MAILPORT (  
    mailport IN NUMBER);
```

### Parameters

**Table 24–3** SET\_MAILPORT Procedure Parameters

Parameter	Description
mailport	SMTP server port number.

### Usage Notes

As part of the configuration for e-mail notifications, a user with AQ\_ADMINISTRATOR\_ROLE or with EXECUTE permissions on DBMS\_AQELM package needs to set the port number before registering for e-mail notifications. The database uses this SMTP server port number to send out e-mail notifications. If not set, the SMTP mailport defaults to 25

## SET\_SENDFROM Procedure

This procedure sets the sent-from e-mail address. This e-mail address is used in the sent-from field in all the e-mail notifications sent out by the database to the registered e-mail addresses.

### Syntax

```
DBMS_AQELM.SET_SENDFROM (  
    sendfrom IN VARCHAR2);
```

### Parameters

**Table 24–4 SET\_SENDFROM Procedure Parameters**

Parameter	Description
sendfrom	The sent-from e-mail address.

### Usage Notes

As part of the configuration for e-mail notifications, a user with AQ\_ADMINISTRATOR\_ROLE or with EXECUTE permissions on the DBMS\_AQELM package should set the sent-from address before registering for e-mail notifications





The DBMS\_AQIN package plays a part in providing secure access to the Oracle JMS interfaces.

**See Also:** *Oracle Streams Advanced Queuing User's Guide* for detailed information about DBMS\_AQIN

This chapter contains the following topic:

- [Using DBMS\\_AQIN](#)
  - Over view

---

## Using DBMS\_AQIN

This section contains topics which relate to using the DBMS\_AQIN package.

- [Overview](#)

## Overview

While you should not call any subprograms in the DBMS\_AQIN package directly, you must have the EXECUTE privilege on the DBMS\_AQIN and DBMS\_AQJMS packages to use the Oracle JMS interfaces. Use the following syntax to accomplish this with regard to the DBMS\_AQIN package:

```
GRANT EXECUTE ON DBMS_AQIN to user;
```

Note that you can also acquire these rights through the AQ\_USER\_ROLE or the AQ\_ADMINISTRATOR\_ROLE.



The DBMS\_ASSERT package provides an interface to validate properties of the input value.

**See Also:** *Oracle Database PL/SQL Language Reference* for more information about "Avoiding SQL Injection in PL/SQL"

This chapter contains the following topics:

- [Using DBMS\\_ASSERT](#)
  - Operational Notes
- [Summary of DBMS\\_ASSERT Subprograms](#)

---

## Using DBMS\_ASSERT

- [Operational Notes](#)

## Operational Notes

If the condition which determines the property asserted in a function is not met then a value error is raised. Otherwise the input value is returned through the return value. Most functions return the value unchanged, however, several functions modify the value.

## Summary of DBMS\_ASSERT Subprograms

**Table 26–1** *DBMS\_APPLICATION\_INFO Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">ENQUOTE_LITERAL Function</a> on page 26-5	Enquotes a string literal
<a href="#">ENQUOTE_NAME Function</a> on page 26-6	Encloses a name in double quotes
<a href="#">NOOP Functions</a> on page 26-7	Returns the value without any checking
<a href="#">QUALIFIED_SQL_NAME Function</a> on page 26-8	Verifies that the input string is a qualified SQL name
<a href="#">SCHEMA_NAME Function</a> on page 26-9	Verifies that the input string is an existing schema name
<a href="#">SIMPLE_SQL_NAME Function</a> on page 26-10	Verifies that the input string is a simple SQL name
<a href="#">SQL_OBJECT_NAME Function</a> on page 26-11	Verifies that the input parameter string is a qualified SQL identifier of an existing SQL object



## ENQUOTE\_LITERAL Function

This function enquotes a string literal.

### Syntax

```
DBMS_ASSERT.ENQUOTE_LITERAL (  
    str          VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 26–2 ENQUOTE\_LITERAL Function Parameters**

Parameter	Description
str	String to enquote

### Usage Notes

- Add leading and trailing single quotes to a string literal.
- Verify that all single quotes except leading and trailing characters are paired with adjacent single quotes.

## ENQUOTE\_NAME Function

This function encloses a name in double quotes.

### Syntax

```
DBMS_ASSERT.ENQUOTE_NAME (  
    str          VARCHAR2,  
    capitalize   BOOLEAN DEFAULT TRUE)  
RETURN VARCHAR2;
```

### Parameters

**Table 26-3 ENQUOTE\_NAME Function Parameters**

Parameter	Description
<code>str</code>	String to enquote
<code>capitalize</code>	If TRUE or defaulted, alphabetic characters of <code>str</code> which was not in quotes are translated to upper case

## NOOP Functions

This function returns the value without any checking.

### Syntax

```
DBMS_ASSERT.NOOP (  
    str      VARCHAR2 CHARACTER SET ANY_CS)  
RETURN      VARCHAR2 CHARACTER SET str%CHARSET;
```

```
DBMS_ASSERT.NOOP (  
    str      CLOB CHARACTER SET ANY_CS)  
RETURN      CLOB CHARACTER SET str%CHARSET;
```

### Parameters

**Table 26–4** NOOP Function Parameters

Parameter	Description
str	Input value

## QUALIFIED\_SQL\_NAME Function

This function verifies that the input string is a qualified SQL name.

### Syntax

```
DBMS_ASSERT.QUALIFIED_SQL_NAME (  
    str      VARCHAR2 CHARACTER SET ANY_CS)  
RETURN      VARCHAR2 CHARACTER SET str%CHARSET;
```

### Parameters

**Table 26–5** *QUALIFIED\_SQL\_NAME Function Parameters*

Parameter	Description
str	Input value

### Exceptions

ORA44004: string is not a qualified SQL name

### Usage Notes

A qualified SQL name <qualified name> can be expressed by the following grammar:

```
<local qualified name> ::= <simple name> { '.' <simple name> }  
<database link name> ::= <local qualified name> [ '@' <connection string> ]  
<connection string> ::= <simple name>  
<qualified name> ::= <local qualified name> [ '@' <database link name> ]
```

## SCHEMA\_NAME Function

This function verifies that the input string is an existing schema name.

### Syntax

```
DBMS_ASSERT.SCHEMA_NAME (  
    str          VARCHAR2 CHARACTER SET ANY_CS)  
RETURN          VARCHAR2 CHARACTER SET str%CHARSET;
```

### Parameters

**Table 26–6** SCHEMA\_NAME Function Parameters

Parameter	Description
str	Input value

### Exceptions

ORA44001: Invalid schema name

### Usage Notes

By definition, a schema name need not be just a simple SQL name. For example, "FIRST LAST" is a valid schema name. As a consequence, care must be taken to quote the output of schema name before concatenating it with SQL text.

## SIMPLE\_SQL\_NAME Function

This function verifies that the input string is a simple SQL name.

### Syntax

```
DBMS_ASSERT.SIMPLE_SQL_NAME (  
    str      VARCHAR2 CHARACTER SET ANY_CS)  
RETURN      VARCHAR2 CHARACTER SET str%CHARSET;
```

### Parameters

**Table 26–7** SIMPLE\_SQL\_NAME Function Parameters

Parameter	Description
str	Input value

### Exceptions

ORA44003: string is not a simple SQL name

### Usage Notes

- The input value must meet the following conditions:
  - The name must begin with an alphabetic character. It may contain alphanumeric characters as well as the characters `_`, `$`, and `#` in the second and subsequent character positions.
  - Quoted SQL names are also allowed.
  - Quoted names must be enclosed in double quotes.
  - Quoted names allow any characters between the quotes.
  - Quotes inside the name are represented by two quote characters in a row, for example, "a name with "" inside" is a valid quoted name.
  - The input parameter may have any number of leading and/or trailing white space characters.
- The length of the name is not checked.

## SQL\_OBJECT\_NAME Function

This function verifies that the input parameter string is a qualified SQL identifier of an existing SQL object.

### Syntax

```
DBMS_ASSERT.SQL_OBJECT_NAME (  
    str          VARCHAR2 CHARACTER SET ANY_CS)  
RETURN          VARCHAR2 CHARACTER SET str%CHARSET;
```

### Parameters

**Table 26–8** *SQL\_OBJECT\_NAME Function Parameters*

Parameter	Description
str	Input value

### Exceptions

ORA44002: Invalid object name

### Usage Notes

The use of synonyms requires that the base object exists.





---

---

## DBMS\_AUDIT\_MGMT

The DBMS\_AUDIT\_MGMT package provides subprograms to manage audit trail records. These subprograms enable audit administrators to manage the various audit trail types like database audit trails, operating system (OS) audit trails, and XML audit trails.

**See Also:** "Verifying Security Access with Auditing" in the *Oracle Database Security Guide* for more information on using the DBMS\_AUDIT\_MGMT package to manage audit trails

This chapter contains the following topics:

- [Using DBMS\\_AUDIT\\_MGMT](#)
  - Overview
  - Security Model
  - Constants
  - Views
- [Subprogram Groups](#)
  - Audit Trail Management Subprograms
  - Audit Trail Cleanup Subprograms
- [Summary of DBMS\\_AUDIT\\_MGMT Subprograms](#)

## Using DBMS\_AUDIT\_MGMT

This section contains topics which relate to using the DBMS\_AUDIT\_MGMT package. The following topics are included:

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)

## Overview

Database auditing helps meet your database security and compliance requirements. Audit records are written to database tables, operating system (OS) files, or XML files depending on the `AUDIT_TRAIL` initialization parameter setting.

When `AUDIT_TRAIL` is set to `DB`, database records are written to the `AUD$` table.

When `AUDIT_TRAIL` is set to `OS`, audit records are written to operating system files.

When `AUDIT_TRAIL` is set to `XML`, audit records are written to operating system files in XML format.

**See Also:** "Verifying Security Access with Auditing" in the *Oracle Database Security Guide* for more background information on database auditing

It is important to manage your audit records properly in order to ensure efficient performance and disk space management. The `DBMS_AUDIT_MGMT` subprograms enable you to efficiently manage your audit trail records.

The `DBMS_AUDIT_MGMT` package provides a subprogram that allows you to move the database audit trail tables out of the `SYSTEM` tablespace. This improves overall database performance by reducing the load on the `SYSTEM` tablespace. It also enables you to dedicate an optimized tablespace for audit records.

The `DBMS_AUDIT_MGMT` subprograms also enable you to manage your operating system and XML audit files. You can define properties like the maximum size and age of an audit file. This enables you to keep the file sizes of OS and XML audit files in check.

The `DBMS_AUDIT_MGMT` subprograms enable you to perform cleanup operations on all audit trail types. Audit trails can be cleaned based on the Last Archive Timestamp value. The Last Archive Timestamp represents the timestamp of the most recent audit record that was securely archived.

The `DBMS_AUDIT_MGMT` package provides a subprogram that enables audit administrators to set the last archive timestamp for archived audit records. This subprogram can also be used by external archival systems to set the last archive timestamp.

The `DBMS_AUDIT_MGMT` subprograms also enable you to configure jobs that periodically delete audit trail records. The frequency with which these jobs should run can be controlled by the audit administrator.

## Security Model

All `DBMS_AUDIT_MGMT` subprograms require the user to have `EXECUTE` privilege over the `DBMS_AUDIT_MGMT` package. The `SYSDBA` role has `EXECUTE` privileges on the package by default.

Oracle strongly recommends that only audit administrators should have `EXECUTE` privileges over the `DBMS_AUDIT_MGMT` package.

## Constants

The DBMS\_AUDIT\_MGMT package defines several enumerated constants that can be used for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS\_AUDIT\_MGMT.AUDIT\_TRAIL\_AUD\_STD.

The DBMS\_AUDIT\_MGMT package includes the constants shown in the following tables:

- [DBMS\\_AUDIT\\_MGMT Constants - Audit Trail Types](#)
- [DBMS\\_AUDIT\\_MGMT Constants - Audit Trail Properties](#)
- [DBMS\\_AUDIT\\_MGMT Constants - Purge Job Status](#)

Audit trails can be classified based on whether audit records are written to database tables, operating system files, or XML files. [Table 27-1](#) lists the audit trail type constants.

**Table 27-1 DBMS\_AUDIT\_MGMT Constants - Audit Trail Types**

Constant	Type	Description
AUDIT_TRAIL_ALL	PLS_INTEGER	All audit trail types. This includes the standard database audit trail (SYS.AUD\$ and SYS.FGA_LOG\$ tables), operating system (OS) audit trail, and XML audit trail.
AUDIT_TRAIL_AUD_STD	PLS_INTEGER	Standard database audit records in the SYS.AUD\$ table
AUDIT_TRAIL_DB_STD	PLS_INTEGER	Both standard audit (SYS.AUD\$) and FGA audit(SYS.FGA_LOG\$) records
AUDIT_TRAIL_FGA_STD	PLS_INTEGER	Standard database fine-grained auditing (FGA) records in the SYS.FGA_LOG\$ table
AUDIT_TRAIL_FILES	PLS_INTEGER	Both operating system (OS) and XML audit trails
AUDIT_TRAIL_OS	PLS_INTEGER	Operating system audit trail. This refers to the audit records stored in operating system files.
AUDIT_TRAIL_XML	PLS_INTEGER	XML audit trail. This refers to the audit records stored in XML files.

Audit trail properties determine the audit configuration settings. [Table 27-2](#) lists the constants related to audit trail properties.

**Table 27-2 DBMS\_AUDIT\_MGMT Constants - Audit Trail Properties**

Constant	Type	Description
CLEAN_UP_INTERVAL	PLS_INTEGER	Interval, in hours, after which the cleanup procedure is called to clear audit records in the specified audit trail
DB_DELETE_BATCH_SIZE	PLS_INTEGER	Specifies the batch size to be used for deleting audit records in database audit tables. The audit records are deleted in batches of size equal to DB_DELETE_BATCH_SIZE.
FILE_DELETE_BATCH_SIZE	PLS_INTEGER	Specifies the batch size to be used for deleting audit files in the audit directory. The audit files are deleted in batches of size equal to FILE_DELETE_BATCH_SIZE.

**Table 27–2 (Cont.) DBMS\_AUDIT\_MGMT Constants - Audit Trail Properties**

Constant	Type	Description
OS_FILE_MAX_AGE	PLS_INTEGER	Specifies the maximum number of days for which an operating system (OS) or XML audit file can be kept open before a new audit file gets created
OS_FILE_MAX_SIZE	PLS_INTEGER	Specifies the maximum size, in kilobytes (KB), to which an operating system (OS) or XML audit file can grow before a new file is opened

The audit trail purge job cleans the audit trail. [Table 27–3](#) lists the constants related to purge job status values.

**Table 27–3 DBMS\_AUDIT\_MGMT Constants - Purge Job Status**

Constant	Type	Description
PURGE_JOB_DISABLE	PLS_INTEGER	Disables a purge job
PURGE_JOB_ENABLE	PLS_INTEGER	Enables a purge job

## Views

The views listed in [Table 27-4](#) are used to display DBMS\_AUDIT\_MGMT configuration and cleanup events.

**Table 27-4 Views used by DBMS\_AUDIT\_MGMT**

<b>View</b>	<b>Description</b>
DBA_AUDIT_MGMT_CLEAN_EVENTS	Displays the cleanup event history
DBA_AUDIT_MGMT_CLEANUP_JOBS	Displays the currently configured audit trail purge jobs
DBA_AUDIT_MGMT_CONFIG_PARAMS	Displays the currently configured audit trail properties
DBA_AUDIT_MGMT_LAST_ARCH_TS	Displays the last archive timestamps set for the audit trails

**See Also:** *Oracle Database Reference* for more information on these views

---

## Subprogram Groups

The DBMS\_AUDIT\_MGMT package subprograms can be grouped into the following categories:

- [Audit Trail Management Subprograms](#)
- [Audit Trail Cleanup Subprograms](#)



## Audit Trail Management Subprograms

Audit trail management subprograms enable you to manage audit trail properties.

**Table 27-5** *Audit Trail Management Subprograms*

Subprogram	Description
<a href="#">CLEAR_AUDIT_TRAIL_PROPERTY Procedure</a> on page 27-14	Clears the value for the audit trail property that you specify
<a href="#">SET_AUDIT_TRAIL_LOCATION Procedure</a> on page 27-25	Moves the audit trail tables from their current tablespace to a user-specified tablespace
<a href="#">SET_AUDIT_TRAIL_PROPERTY Procedure</a> on page 27-27	Sets an audit trail property for the audit trail type that you specify

The [Summary of DBMS\\_AUDIT\\_MGMT Subprograms](#) contains a complete listing of all subprograms in the package.

## Audit Trail Cleanup Subprograms

Audit trail cleanup subprograms help you perform cleanup related operations on the audit trail records.

**Table 27–6 Audit Trail Cleanup Subprograms**

Subprogram	Description
<a href="#">CLEAN_AUDIT_TRAIL Procedure</a> on page 27-12	Deletes audit trail records/files that have been archived
<a href="#">CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure</a> on page 27-16	Clears the timestamp set by the <a href="#">SET_LAST_ARCHIVE_TIMESTAMP Procedure</a>
<a href="#">CREATE_PURGE_JOB Procedure</a> on page 27-17	Creates a purge job for periodically deleting the audit trail records/files
<a href="#">DEINIT_CLEANUP Procedure</a> on page 27-19	Undoes the setup and initialization performed by the <a href="#">INIT_CLEANUP Procedure</a>
<a href="#">DROP_PURGE_JOB Procedure</a> on page 27-20	Drops the purge job created using the <a href="#">CREATE_PURGE_JOB Procedure</a>
<a href="#">INIT_CLEANUP Procedure</a> on page 27-22	Sets up the audit management infrastructure and sets a default cleanup interval for audit trail records/files
<a href="#">IS_CLEANUP_INITIALIZED Function</a> on page 27-24	Checks to see if the <a href="#">INIT_CLEANUP Procedure</a> has been run for an audit trail type
<a href="#">SET_LAST_ARCHIVE_TIMESTAMP Procedure</a> on page 27-30	Sets a timestamp indicating when the audit records/files were last archived
<a href="#">SET_PURGE_JOB_INTERVAL Procedure</a> on page 27-32	Sets the interval at which the <a href="#">CLEAN_AUDIT_TRAIL Procedure</a> is called for the purge job that you specify
<a href="#">SET_PURGE_JOB_STATUS Procedure</a> on page 27-33	Enables or disables the purge job that you specify

The [Summary of DBMS\\_AUDIT\\_MGMT Subprograms](#) contains a complete listing of all subprograms in the package.

---

## Summary of DBMS\_AUDIT\_MGMT Subprograms

**Table 27–7 DBMS\_AUDIT\_MGMT Package Subprograms**

Subprogram	Description
<a href="#">CLEAN_AUDIT_TRAIL Procedure</a> on page 27-12	Deletes audit trail records that have been archived
<a href="#">CLEAR_AUDIT_TRAIL_PROPERTY Procedure</a> on page 27-14	Clears the value for the audit trail property that you specify
<a href="#">CLEAR_LAST_ARCHIVE_TIMESTAMP Procedure</a> on page 27-16	Clears the timestamp set by the <a href="#">SET_LAST_ARCHIVE_TIMESTAMP Procedure</a>
<a href="#">CREATE_PURGE_JOB Procedure</a> on page 27-17	Creates a purge job for periodically deleting the audit trail records
<a href="#">DEINIT_CLEANUP Procedure</a> on page 27-19	Undoes the setup and initialization performed by the <a href="#">INIT_CLEANUP Procedure</a>
<a href="#">DROP_PURGE_JOB Procedure</a> on page 27-20	Drops the purge job created using the <a href="#">CREATE_PURGE_JOB Procedure</a>
<a href="#">GET_AUDIT_COMMIT_DELAY Function</a> on page 27-21	Returns the Audit Commit Delay as the number of seconds. This is the maximum time that it takes to COMMIT an audit record to the database audit trail.
<a href="#">INIT_CLEANUP Procedure</a> on page 27-22	Sets up the audit management infrastructure and sets a default cleanup interval for audit trail records
<a href="#">IS_CLEANUP_INITIALIZED Function</a> on page 27-24	Checks to see if the <a href="#">INIT_CLEANUP Procedure</a> has been run for an audit trail type
<a href="#">SET_AUDIT_TRAIL_LOCATION Procedure</a> on page 27-25	Moves the audit trail tables from their current tablespace to a user-specified tablespace
<a href="#">SET_AUDIT_TRAIL_PROPERTY Procedure</a> on page 27-27	Sets the audit trail properties for the audit trail type that you specify
<a href="#">SET_LAST_ARCHIVE_TIMESTAMP Procedure</a> on page 27-30	Sets a timestamp indicating when the audit records were last archived
<a href="#">SET_PURGE_JOB_INTERVAL Procedure</a> on page 27-32	Sets the interval at which the <a href="#">CLEAN_AUDIT_TRAIL Procedure</a> is called for the purge job that you specify
<a href="#">SET_PURGE_JOB_STATUS Procedure</a> on page 27-33	Enables or disables the purge job that you specify

## CLEAN\_AUDIT\_TRAIL Procedure

This procedure deletes audit trail records. The `CLEAN_AUDIT_TRAIL` procedure is usually called after the [SET\\_LAST\\_ARCHIVE\\_TIMESTAMP Procedure](#) has been used to set the last archived timestamp for the audit records.

### Syntax

```
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL (
  audit_trail_type          IN PLS_INTEGER,
  use_last_arch_timestamp   IN BOOLEAN DEFAULT TRUE) ;
```

### Parameters

**Table 27–8** *CLEAN\_AUDIT\_TRAIL Procedure Parameters*

Parameter	Description
<code>audit_trail_type</code>	The audit trail type for which the cleanup operation needs to be performed. Audit trail types are listed in <a href="#">Table 27–1</a> , "DBMS_AUDIT_MGMT Constants - Audit Trail Types" on page 27-5.
<code>use_last_arch_timestamp</code>	Specifies whether the last archived timestamp should be used for deciding on the records that should be deleted.  A value of <code>TRUE</code> indicates that only audit records created before the last archive timestamp should be deleted.  A value of <code>FALSE</code> indicates that all audit records should be deleted.  The default value is <code>TRUE</code> . Oracle recommends using this value, as this helps guard against inadvertent deletion of records.

### Usage Notes

The following usage notes apply:

- When cleaning up operating system (OS) or XML audit files, only files in the current audit directory, specified by the `AUDIT_FILE_DEST` parameter, are cleaned up.
- For Windows platforms, no cleanup is performed when the `audit_trail_type` parameter is set to `DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS`. This is because operating system (OS) audit records on Windows are written to the Windows Event Viewer.
- For Unix platforms, no cleanup is performed for cases where the operating system (OS) audit records are written to the syslog. When the `audit_trail_type` parameter is set to `DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS`, it removes only the `*.aud` files under the directory specified by the `AUDIT_FILE_DEST` initialization parameter.

**See Also:** "AUDIT\_SYSLOG\_LEVEL" in the *Oracle Database Reference*

- When the `audit_trail_type` parameter is set to `DBMS_AUDIT_MGMT.AUDIT_TRAIL_XML`, this procedure only removes XML audit files (`*.xml`) from the current audit directory.

Oracle database maintains a book-keeping file (`adx_${ORACLE_SID}.txt`) for the XML audit files. This file is not removed by the cleanup procedure.

## Examples

The following example calls the `CLEAN_AUDIT_TRAIL` procedure to clean up the operating system (OS) audit trail records that were updated before the last archive timestamp.

```
BEGIN
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
  audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
  use_last_arch_timestamp => TRUE);
END;
/
```

## CLEAR\_AUDIT\_TRAIL\_PROPERTY Procedure

This procedure clears the value for the audit trail property that is specified. Audit trail properties are set using the [SET\\_AUDIT\\_TRAIL\\_PROPERTY Procedure](#).

The CLEAR\_AUDIT\_TRAIL\_PROPERTY procedure can optionally reset the property value to its default value through the `use_default_values` parameter.

### Syntax

```
DBMS_AUDIT_MGMT.CLEAR_AUDIT_TRAIL_PROPERTY (
  audit_trail_type      IN PLS_INTEGER,
  audit_trail_property  IN PLS_INTEGER,
  use_default_values    IN BOOLEAN DEFAULT FALSE) ;
```

### Parameters

**Table 27-9 CLEAR\_AUDIT\_TRAIL\_PROPERTY Procedure Parameters**

Parameter	Description
<code>audit_trail_type</code>	The audit trail type for which the property needs to be cleared. Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5
<code>audit_trail_property</code>	The audit trail property whose value needs to be cleared. You cannot clear the value for the <code>CLEANUP_INTERVAL</code> property.  Audit trail properties are listed in <a href="#">Table 27-2, "DBMS_AUDIT_MGMT Constants - Audit Trail Properties"</a>
<code>use_default_values</code>	Specifies whether the default value of the <code>audit_trail_property</code> should be used in place of the cleared value. A value of <code>TRUE</code> causes the default value of the parameter to be used. A value of <code>FALSE</code> causes the <code>audit_trail_property</code> to have no value.  The default value for this parameter is <code>FALSE</code> .

### Usage Notes

The following usage notes apply:

- You can use this procedure to clear the value for an audit trail property that you do not wish to use. For example, if you do not want a restriction on the operating system audit file size, then you can use this procedure to reset the `OS_FILE_MAX_SIZE` property.

You can also use this procedure to reset an audit trail property to its default value. You need to set `use_default_values` to `TRUE` when invoking the procedure.

- The `DB_DELETE_BATCH_SIZE` property needs to be individually cleared for the `AUDIT_TRAIL_AUD_STD` and `AUDIT_TRAIL_FGA_STD` audit trail types. You cannot clear this property collectively using the `AUDIT_TRAIL_DB_STD` and `AUDIT_TRAIL_ALL` audit trail types.
- If you clear the value of the `DB_DELETE_BATCH_SIZE` property with `use_default_value` set to `FALSE`, the default value of `DB_DELETE_BATCH_SIZE` is still assumed. This is because audit records are always deleted in batches.

- The `FILE_DELETE_BATCH_SIZE` property needs to be individually cleared for the `AUDIT_TRAIL_OS` and `AUDIT_TRAIL_XML` audit trail types. You cannot clear this property collectively using the `AUDIT_TRAIL_FILES` and `AUDIT_TRAIL_ALL` audit trail types.
- If you clear the value of the `FILE_DELETE_BATCH_SIZE` property with `use_default_value` set to `FALSE`, the default value of `FILE_DELETE_BATCH_SIZE` is still assumed. This is because audit files are always deleted in batches.
- You cannot clear the value for the `CLEANUP_INTERVAL` property.

## Examples

The following example calls the `CLEAR_AUDIT_TRAIL_PROPERTY` procedure to clear the value for the audit trail property, `OS_FILE_MAX_SIZE`. The procedure uses a value of `FALSE` for the `USE_DEFAULT_VALUES` parameter. This means that there will be no maximum size threshold for operating system (OS) audit files.

```
BEGIN
DBMS_AUDIT_MGMT.CLEAR_AUDIT_TRAIL_PROPERTY (
  AUDIT_TRAIL_TYPE      =>  DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
  AUDIT_TRAIL_PROPERTY  =>  DBMS_AUDIT_MGMT.OS_FILE_MAX_SIZE,
  USE_DEFAULT_VALUES    =>  FALSE );
END;
/
```

## CLEAR\_LAST\_ARCHIVE\_TIMESTAMP Procedure

This procedure clears the timestamp set by the [SET\\_LAST\\_ARCHIVE\\_TIMESTAMP Procedure](#).

### Syntax

```
DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP (
  audit_trail_type      IN PLS_INTEGER,
  rac_instance_number  IN PLS_INTEGER DEFAULT NULL) ;
```

### Parameters

**Table 27-10 CLEAR\_LAST\_ARCHIVE\_TIMESTAMP Procedure Parameters**

Parameter	Description
audit_trail_type	The audit trail type for which the timestamp needs to be cleared. Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5.
rac_instance_number	The instance number for the Oracle Real Application Clusters (Oracle RAC) instance. The default value is NULL. The rac_instance_number is not relevant for single instance databases.

### Usage Notes

The following usage notes apply:

- The timestamp for only one audit\_trail\_type can be cleared at a time.
- The following are invalid audit\_trail\_type values for this procedure and cannot be used:
  - AUDIT\_TRAIL\_ALL
  - AUDIT\_TRAIL\_DB\_STD
  - AUDIT\_TRAIL\_FILES

### Examples

The following example calls the CLEAR\_LAST\_ARCHIVE\_TIMESTAMP procedure to clear the timestamp value for the operating system (OS) audit trail type.

```
BEGIN
DBMS_AUDIT_MGMT.CLEAR_LAST_ARCHIVE_TIMESTAMP (
  audit_trail_type      => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
  rac_instance_number  => 1);
END;
/
```



## CREATE\_PURGE\_JOB Procedure

This procedure creates a purge job for periodically deleting the audit trail records.

This procedure carries out the cleanup operation at intervals specified by the user. It calls the [CLEAN\\_AUDIT\\_TRAIL Procedure](#) to perform the cleanup operation.

The [SET\\_PURGE\\_JOB\\_INTERVAL Procedure](#) is used to modify the frequency of the purge job.

The [SET\\_PURGE\\_JOB\\_STATUS Procedure](#) is used to enable or disable the purge job.

The [DROP\\_PURGE\\_JOB Procedure](#) is used to drop a purge job created with the CREATE\_PURGE\_JOB procedure.

### Syntax

```
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB (
  audit_trail_type          IN PLS_INTEGER,
  audit_trail_purge_interval IN PLS_INTEGER,
  audit_trail_purge_name    IN VARCHAR2,
  use_last_arch_timestamp  IN BOOLEAN DEFAULT TRUE) ;
```

### Parameters

**Table 27-11 CREATE\_PURGE\_JOB Procedure Parameters**

Parameter	Description
audit_trail_type	The audit trail type for which the purge job needs to be created. Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5.
audit_trail_purge_interval	The interval, in hours, at which the clean up procedure is called. A lower value means that the cleanup is performed more often.
audit_trail_purge_name	A name to identify the purge job.
use_last_arch_timestamp	Specifies whether the last archived timestamp should be used for deciding on the records that should be deleted.  A value of <code>TRUE</code> indicates that only audit records created before the last archive timestamp should be deleted.  A value of <code>FALSE</code> indicates that all audit records should be deleted.  The default value is <code>TRUE</code> .

### Usage Notes

Use this procedure to schedule the [CLEAN\\_AUDIT\\_TRAIL Procedure](#) for your audit trail records.

### Examples

The following example calls the CREATE\_PURGE\_JOB procedure to create a cleanup job called CLEANUP, for all audit trail types. It sets the audit\_trail\_purge\_interval parameter to 100. This means that the cleanup job is invoked every 100 hours. It also sets the use\_last\_arch\_timestamp parameter

value to TRUE. This means that all audit records older than the last archive timestamp are deleted.

```
BEGIN
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB (
  audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,
  audit_trail_purge_interval => 100 /* hours */,
  audit_trail_purge_name => 'CLEANUP',
  use_last_arch_timestamp => TRUE);
END;
/
```

## DEINIT\_CLEANUP Procedure

This procedure undoes the setup and initialization performed by the [INIT\\_CLEANUP Procedure](#). The DEINIT\_CLEANUP procedure clears the value of the `default_cleanup_interval` parameter. However, when used for audit tables, it does not move the audit trail tables back to their original tablespace.

### Syntax

```
DBMS_AUDIT_MGMT.DEINIT_CLEANUP(
    audit_trail_type IN PLS_INTEGER) ;
```

### Parameters

**Table 27–12 DEINIT\_CLEANUP Procedure Parameters**

Parameter	Description
<code>audit_trail_type</code>	The audit trail type for which the procedure needs to be called.  Audit trail types are listed in <a href="#">Table 27–1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5

### Examples

The following example clears the `default_cleanup_interval` parameter setting for the standard database audit trail:

```
BEGIN
DBMS_AUDIT_MGMT.DEINIT_CLEANUP(
    AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD);
END;
/
```

## DROP\_PURGE\_JOB Procedure

This procedure drops the purge job created using the [CREATE\\_PURGE\\_JOB Procedure](#). The name of the purge job is passed as an argument.

### Syntax

```
DBMS_AUDIT_MGMT.DROP_PURGE_JOB(  
    audit_trail_purge_name    IN VARCHAR2) ;
```

### Parameters

**Table 27–13** *DROP\_PURGE\_JOB Procedure Parameters*

Parameter	Description
audit_trail_purge_name	The name of the purge job which is being deleted. This is the purge job name that you specified with the <a href="#">CREATE_PURGE_JOB Procedure</a> .

### Examples

The following example calls the DROP\_PURGE\_JOB procedure to drop the purge job called CLEANUP.

```
BEGIN  
DBMS_AUDIT_MGMT.DROP_PURGE_JOB(  
    AUDIT_TRAIL_PURGE_NAME => 'CLEANUP');  
END;  
/
```

## GET\_AUDIT\_COMMIT\_DELAY Function

This function returns the Audit Commit Delay as the number of seconds. Audit Commit Delay is the maximum time that it takes to COMMIT an audit record to the database audit trail. If it takes more time to COMMIT an audit record than defined by the Audit Commit Delay, then a copy of the audit record is written to the operating system (OS) audit trail.

The Audit Commit Delay value is useful when determining the last archive timestamp for database audit records.

### Syntax

```
DBMS_AUDIT_MGMT.GET_AUDIT_COMMIT_DELAY  
RETURN NUMBER;
```

## INIT\_CLEANUP Procedure

This procedure sets up the audit management infrastructure and a default cleanup interval for the audit trail records. If the audit trail tables are in the `SYSTEM` tablespace, then the procedure moves them to the `SYSAUX` tablespace.

Moving the audit trail tables out of the `SYSTEM` tablespace enhances overall database performance. The `INIT_CLEANUP` procedure moves the audit trail tables to the `SYSAUX` tablespace. If the [SET\\_AUDIT\\_TRAIL\\_LOCATION Procedure](#) has already moved the audit tables elsewhere, then no tables are moved.

The [SET\\_AUDIT\\_TRAIL\\_LOCATION Procedure](#) enables you to specify an alternate target tablespace for the database audit tables.

The `INIT_CLEANUP` procedure is currently not relevant for the `AUDIT_TRAIL_OS`, `AUDIT_TRAIL_XML`, and `AUDIT_TRAIL_FILES` audit trail types. No preliminary set up is required for these audit trail types.

**See Also:** [Table 27-1, "DBMS\\_AUDIT\\_MGMT Constants - Audit Trail Types"](#) on page 27-5 for a list of all audit trail types

This procedure also sets a default cleanup interval for the audit trail records.

### Syntax

```
DBMS_AUDIT_MGMT.INIT_CLEANUP (
  audit_trail_type          IN PLS_INTEGER,
  default_cleanup_interval IN PLS_INTEGER);
```

### Parameters

**Table 27-14** *INIT\_CLEANUP Procedure Parameters*

Parameter	Description
<code>audit_trail_type</code>	The audit trail type for which the clean up operation needs to be initialized.  Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5
<code>default_cleanup_interval</code>	The default time interval, in hours, after which the cleanup procedure should be called. The minimum value is 1 and the maximum is 999.

### Usage Notes

The following usage notes apply:

- This procedure may involve data movement across tablespaces. This can be a resource intensive operation especially if your database audit trail tables are already populated. Oracle recommends that you invoke the procedure during non-peak hours.
- You should ensure that the `SYSAUX` tablespace, into which the audit trail tables are being moved, has sufficient space to accommodate the audit trail tables. You should also optimize the `SYSAUX` tablespace for frequent write operations.
- You can change the `default_cleanup_interval` later using the [SET\\_AUDIT\\_TRAIL\\_PROPERTY Procedure](#).

- If you do not wish to move the audit trail tables to the SYSAUX tablespace, then you should use the `DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION` procedure to move the audit trail tables to another tablespace before calling the `INIT_CLEANUP` procedure.

**See Also:** ["SET\\_AUDIT\\_TRAIL\\_LOCATION Procedure"](#) on page 27-25

## Examples

The following example calls the `INIT_CLEANUP` procedure to set a `default_cleanup_interval` of 12 hours for all audit trail types:

```
BEGIN
DBMS_AUDIT_MGMT.INIT_CLEANUP(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,
    default_cleanup_interval => 12 /* hours */);
END;
/
```

**See Also:** [Table 27-1, "DBMS\\_AUDIT\\_MGMT Constants - Audit Trail Types"](#) on page 27-5 for a list of all audit trail types

## IS\_CLEANUP\_INITIALIZED Function

This function checks to see if the [INIT\\_CLEANUP Procedure](#) has been run for an audit trail type. The `IS_CLEANUP_INITIALIZED` function returns `TRUE` if the procedure has already been run for the audit trail type. It returns `FALSE` if the procedure has not been run for the audit trail type.

This function is currently not relevant for the `AUDIT_TRAIL_OS`, `AUDIT_TRAIL_XML`, and `AUDIT_TRAIL_FILES` audit trail types. The function always returns `TRUE` for these audit trail types. No preliminary set up is required for these audit trail types.

**See Also:** [Table 27-1, "DBMS\\_AUDIT\\_MGMT Constants - Audit Trail Types"](#) on page 27-5 for a list of all audit trail types

### Syntax

```
DBMS_AUDIT_MGMT.DEINIT_CLEANUP(
    audit_trail_type IN PLS_INTEGER)
RETURN BOOLEAN;
```

### Parameters

**Table 27-15 IS\_CLEANUP\_INITIALIZED Function Parameters**

Parameter	Description
<code>audit_trail_type</code>	The audit trail type for which the function needs to be called.  Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5

### Examples

The following example checks to see if the standard database audit trail type has been initialized for cleanup operation. If the audit trail type has not been initialized, then it calls the [INIT\\_CLEANUP Procedure](#) to initialize the audit trail type.

```
BEGIN
  IF
    NOT DBMS_AUDIT_MGMT.IS_CLEANUP_INITIALIZED(DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD)
  THEN
    DBMS_AUDIT_MGMT.INIT_CLEANUP(
      audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD,
      default_cleanup_interval => 12 /* hours */);
  END IF;
END;
```



## SET\_AUDIT\_TRAIL\_LOCATION Procedure

This procedure moves the audit trail tables from their current tablespace to a user-specified tablespace.

The SET\_AUDIT\_TRAIL\_LOCATION procedure is currently not relevant for the AUDIT\_TRAIL\_OS, AUDIT\_TRAIL\_XML, and AUDIT\_TRAIL\_FILES audit trail types. The AUDIT\_FILE\_DEST initialization parameter is the only way you can specify the destination directory for these audit trail types.

### See Also:

- [Table 27-1, "DBMS\\_AUDIT\\_MGMT Constants - Audit Trail Types"](#) on page 27-5 for a list of all audit trail types
- "AUDIT\_FILE\_DEST" in the *Oracle Database Reference*

### Syntax

```
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION(
  audit_trail_type          IN PLS_INTEGER,
  audit_trail_location_value IN VARCHAR2) ;
```

### Parameters

**Table 27-16 SET\_AUDIT\_TRAIL\_LOCATION Procedure Parameters**

Parameter	Description
audit_trail_type	The audit trail type for which the audit trail location needs to be set.  Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5
audit_trail_location_value	The target location/tablespace for the audit trail records

### Usage Notes

The following usage notes apply:

- This procedure involves data movement across tablespaces. This can be a resource intensive operation especially if your database audit trail tables are already populated. Oracle recommends that you invoke the procedure during non-peak hours.
- You should ensure that the target tablespace, into which the audit trail tables are being moved, has sufficient space to accommodate the audit trail tables. You should also optimize the target tablespace for frequent write operations.
- This procedure is valid for the following audit\_trail\_type values only:
  - AUDIT\_TRAIL\_AUD\_STD
  - AUDIT\_TRAIL\_FGA\_STD
  - AUDIT\_TRAIL\_DB\_STD

### Examples

The following example moves the database audit trail tables, AUD\$ and FGA\_LOG\$, from the current tablespace to a user-created tablespace called RECORDS:

```
BEGIN
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
    audit_trail_location_value => 'RECORDS');
END;
/
```

## SET\_AUDIT\_TRAIL\_PROPERTY Procedure

This procedure sets an audit trail property for the audit trail type that is specified.

The procedure sets properties like `OS_FILE_MAX_SIZE`, `OS_FILE_MAX_AGE`, and `FILE_DELETE_BATCH_SIZE` for operating system (OS) and XML audit trail types. The `OS_FILE_MAX_SIZE` and `OS_FILE_MAX_AGE` properties determine the maximum size and age of an audit trail file before a new audit trail file gets created. The `FILE_DELETE_BATCH_SIZE` property specifies the number of audit trail files that are deleted in one batch.

The procedure sets properties like `DB_DELETE_BATCH_SIZE` and `CLEANUP_INTERVAL` for the database audit trail type. `DB_DELETE_BATCH_SIZE` specifies the batch size in which records get deleted from audit trail tables. This ensures that if a cleanup operation gets interrupted midway, the process does not need to start afresh the next time it is invoked. This is because all batches before the last processed batch are already committed.

The `CLEANUP_INTERVAL` specifies the frequency, in hours, with which the cleanup procedure is called.

### Syntax

```
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY (
  audit_trail_type          IN PLS_INTEGER,
  audit_trail_property      IN PLS_INTEGER,
  audit_trail_property_value IN PLS_INTEGER) ;
```

### Parameters

**Table 27-17 SET\_AUDIT\_TRAIL\_PROPERTY Procedure Parameters**

Parameter	Description
<code>audit_trail_type</code>	The audit trail type for which the property needs to be set. Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5
<code>audit_trail_property</code>	The audit trail property that is being set. Audit trail properties are listed in <a href="#">Table 27-2, "DBMS_AUDIT_MGMT Constants - Audit Trail Properties"</a>

**Table 27–17 (Cont.) SET\_AUDIT\_TRAIL\_PROPERTY Procedure Parameters**

Parameter	Description
audit_trail_property_value	<p>The value of the property specified using <code>audit_trail_property</code>. The following are valid values for audit trail properties:</p> <ul style="list-style-type: none"> <li>■ <code>OS_FILE_MAX_SIZE</code> can have a minimum value of 1 and maximum value of 2000000. The default value is 10000. <code>OS_FILE_MAX_SIZE</code> is measured in kilobytes (KB).</li> <li>■ <code>OS_FILE_MAX_AGE</code> can have a minimum value of 1 and a maximum value of 497. The default value is 5. <code>OS_FILE_MAX_AGE</code> is measured in days.</li> <li>■ <code>DB_DELETE_BATCH_SIZE</code> can have a minimum value of 100 and a maximum value of 1000000. The default value is 10000. <code>DB_DELETE_BATCH_SIZE</code> is measured as the number of audit records that are deleted in one batch.</li> <li>■ <code>FILE_DELETE_BATCH_SIZE</code> can have a minimum value of 100 and a maximum value of 1000000. The default value is 1000. <code>FILE_DELETE_BATCH_SIZE</code> is measured as the number of audit files that are deleted in one batch.</li> <li>■ <code>CLEANUP_INTERVAL</code> can have a minimum value of 1 and a maximum value of 999. The default value is set using the <a href="#">INIT_CLEANUP Procedure</a>. <code>CLEANUP_INTERVAL</code> is measured in hours.</li> </ul>

## Usage Notes

The following usage notes apply:

- The audit trail properties for which you do not explicitly set values use their default values.
- If you have set both the `OS_FILE_MAX_SIZE` and `OS_FILE_MAX_AGE` properties for an operating system (OS) or XML audit trail type, then a new audit trail file gets created depending on which of these two limits is reached first.
 

For example, let us take a scenario where `OS_FILE_MAX_SIZE` is 10000 and `OS_FILE_MAX_AGE` is 5. If the operating system audit file is already more than 5 days old and has a size of 9000 KB, then a new audit file is opened. This is because one of the limits has been reached.
- The `DB_DELETE_BATCH_SIZE` property needs to be individually set for the `AUDIT_TRAIL_AUD_STD` and `AUDIT_TRAIL_FGA_STD` audit trail types. You cannot set this property collectively using the `AUDIT_TRAIL_DB_STD` and `AUDIT_TRAIL_ALL` audit trail types.
- The `DB_DELETE_BATCH_SIZE` property enables you to control the number of audit records that are deleted in one batch. Setting a large value for this parameter requires increased allocation for the undo log space.
- The `FILE_DELETE_BATCH_SIZE` property needs to be individually set for the `AUDIT_TRAIL_OS` and `AUDIT_TRAIL_XML` audit trail types. You cannot set this property collectively using the `AUDIT_TRAIL_FILES` and `AUDIT_TRAIL_ALL` audit trail types.

- The `FILE_DELETE_BATCH_SIZE` property enables you to control the number of audit files that are deleted in one batch. Setting a very large value may engage the `GEN0` background process for a long time.

## Examples

The following example calls the `SET_AUDIT_TRAIL_PROPERTY` procedure to set the `OS_FILE_MAX_SIZE` property for the operating system (OS) audit trail. It sets this property value to 102400. This means that a new audit file gets created every time the current audit file size reaches 100 MB.

```
BEGIN
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
    audit_trail_property => DBMS_AUDIT_MGMT.OS_FILE_MAX_SIZE,
    audit_trail_property_value => 102400 /* 100MB*/ );
END;
/
```

The following example calls the `SET_AUDIT_TRAIL_PROPERTY` procedure to set the `OS_FILE_MAX_AGE` property for the operating system (OS) audit trail. It sets this property value to 5. This means that a new audit file gets created every sixth day.

```
BEGIN
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
    audit_trail_property => DBMS_AUDIT_MGMT.OS_FILE_MAX_AGE,
    audit_trail_property_value => 5 /* days */);
END;
/
```

The following example calls the `SET_AUDIT_TRAIL_PROPERTY` procedure to set the `DB_DELETE_BATCH_SIZE` property for the `AUDIT_TRAIL_AUD_STD` audit trail. It sets this property value to 100000. This means that during a cleanup operation, audit records are deleted from the `SYS.AUD$` table in batches of size 100000.

```
BEGIN
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
    audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD,
    audit_trail_property => DBMS_AUDIT_MGMT.DB_DELETE_BATCH_SIZE,
    audit_trail_property_value => 100000 /* delete batch size */);
END;
/
```

## SET\_LAST\_ARCHIVE\_TIMESTAMP Procedure

This procedure sets a timestamp indicating when the audit records were last archived. The audit administrator provides the timestamp to be attached to the audit records. The [CLEAN\\_AUDIT\\_TRAIL Procedure](#) uses this timestamp to decide on the audit records to be deleted.

### Syntax

```
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP (
  audit_trail_type      IN PLS_INTEGER,
  last_archive_time     IN TIMESTAMP,
  rac_instance_number   IN PLS_INTEGER DEFAULT NULL) ;
```

### Parameters

**Table 27-18 SET\_LAST\_ARCHIVE\_TIMESTAMP Procedure Parameters**

Parameter	Description
audit_trail_type	The audit trail type for which the timestamp needs to be set. Audit trail types are listed in <a href="#">Table 27-1, "DBMS_AUDIT_MGMT Constants - Audit Trail Types"</a> on page 27-5.
last_archive_time	The <code>TIMESTAMP</code> value based on which the audit records or files should be deleted. This indicates the last time when the audit records/files were archived.
rac_instance_number	The instance number for the Oracle Real Application Clusters (Oracle RAC) instance. The default value is <code>NULL</code> .  The <code>rac_instance_number</code> is not relevant for the database audit trail type, as the database audit trail tables are shared by all Oracle RAC instances. The <code>rac_instance_number</code> is also not relevant for a single-instance database.

### Usage Notes

The following usage notes apply:

- The `last_archive_time` must be specified in Coordinated Universal Time (UTC) when the audit trail types are `AUDIT_TRAIL_AUD_STD` or `AUDIT_TRAIL_FGA_STD`. This is because the database audit trails store the timestamps in UTC. UTC is also known as Greenwich Mean Time (GMT).
- The `last_archive_time` must be specified as the local time zone time when the audit trail types are `AUDIT_TRAIL_OS` or `AUDIT_TRAIL_XML`. The time zone must be the time zone of the machine where the OS or XML audit files were created. This is because the operating system audit files are cleaned based on the audit file's Last Modification Timestamp property. The Last Modification Timestamp property value is stored in the local time zone of the machine.
- The following `audit_trail_type` values are valid for this procedure:
  - `AUDIT_TRAIL_AUD_STD`
  - `AUDIT_TRAIL_FGA_STD`
  - `AUDIT_TRAIL_OS`
  - `AUDIT_TRAIL_XML`

- When using an Oracle Real Application Clusters (Oracle RAC) database, Oracle recommends that you use the Network Time Protocol (NTP) to synchronize individual Oracle RAC nodes.

## Examples

The following example calls the `SET_LAST_ARCHIVE_TIMESTAMP` procedure to set the last archive timestamp for the operating system (OS) audit trail type on Oracle RAC instance 1. It uses the `TO_TIMESTAMP` function to convert a character string into a timestamp value.

A subsequent call to the [CLEAN\\_AUDIT\\_TRAIL Procedure](#), with `use_last_arch_timestamp` set to `TRUE`, will delete all those OS audit files from the current `AUDIT_FILE_DEST` directory that were modified before 10-Sep-2007 14:10:10.0.

```
BEGIN
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP (
  audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
  last_archive_time =>
TO_TIMESTAMP('10-SEP-0714:10:10.0', 'DD-MON-RRHH24:MI:SS.FF'),
  rac_instance_number => 1);
END;
/
```

## SET\_PURGE\_JOB\_INTERVAL Procedure

This procedure sets the interval at which the [CLEAN\\_AUDIT\\_TRAIL Procedure](#) is called for the purge job specified. The purge job must have already been created using the [CREATE\\_PURGE\\_JOB Procedure](#).

### Syntax

```
DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL(
  audit_trail_purge_name      IN VARCHAR2,
  audit_trail_interval_value  IN PLS_INTEGER) ;
```

### Parameters

**Table 27–19 SET\_PURGE\_JOB\_INTERVAL Procedure Parameters**

Parameter	Description
audit_trail_purge_name	The name of the purge job for which the interval is being set. This is the purge job name that you specified with the <a href="#">CREATE_PURGE_JOB Procedure</a> .
audit_trail_interval_value	The interval, in hours, at which the clean up procedure should be called. This value modifies the audit_trail_purge_interval parameter set using the <a href="#">CREATE_PURGE_JOB Procedure</a>

### Usage Notes

Use this procedure to modify the audit\_trail\_purge\_interval parameter set using the [CREATE\\_PURGE\\_JOB Procedure](#).

### Examples

The following example calls the SET\_PURGE\_JOB\_INTERVAL procedure to change the frequency at which the purge job called CLEANUP is invoked. The new interval is set to 24 hours.

```
BEGIN
DBMS_AUDIT_MGMT.SET_PURGE_JOB_INTERVAL (
  AUDIT_TRAIL_PURGE_NAME      => 'CLEANUP',
  AUDIT_TRAIL_INTERVAL_VALUE  => 24 );
END;
/
```



## SET\_PURGE\_JOB\_STATUS Procedure

This procedure enables or disables the specified purge job. The purge job must have already been created using the [CREATE\\_PURGE\\_JOB Procedure](#).

### Syntax

```
DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS (
  audit_trail_purge_name    IN VARCHAR2,
  audit_trail_status_value  IN PLS_INTEGER) ;
```

### Parameters

**Table 27–20 SET\_PURGE\_JOB\_STATUS Procedure Parameters**

Parameter	Description
audit_trail_purge_name	The name of the purge job for which the status is being set. This is the purge job name that you specified with the <a href="#">CREATE_PURGE_JOB Procedure</a> .
audit_trail_status_value	One of the values specified in <a href="#">DBMS_AUDIT_MGMT Constants - Purge Job Status</a> . The value PURGE_JOB_ENABLE enables the specified purge job. The value PURGE_JOB_DISABLE disables the specified purge job.

### Examples

The following example calls the SET\_PURGE\_JOB\_STATUS procedure to enable the CLEANUP purge job.

```
BEGIN
DBMS_AUDIT_MGMT.SET_PURGE_JOB_STATUS (
  audit_trail_purge_name    => 'CLEANUP',
  audit_trail_status_value  => DBMS_AUDIT_MGMT.PURGE_JOB_ENABLE);
END;
/
```



---

---

## DBMS\_AUTO\_SQLTUNE

The DBMS\_AUTO\_SQLTUNE package is the interface for managing the Automatic SQL Tuning task. Unlike DBMS\_SQLTUNE, the DBMS\_AUTO\_SQLTUNE package requires the DBA role.

---

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

---

The chapter contains the following topics:

- [Using DBMS\\_AUTO\\_SQLTUNE](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_AUTO\\_SQLTUNE Subprograms](#)

---

## Using DBMS\_AUTO\_SQLTUNE

- [Overview](#)
- [Security Model](#)

## Overview

The DBMS\_AUTO\_SQLTUNE package is the interface to SQL Tuning Advisor (DBMS\_SQLTUNE) when run within the Autotask framework. The database creates the automated system task SYS\_AUTO\_SQL\_TUNING\_TASK as part of the catalog scripts. This task automatically chooses a set of high-load SQL from AWR and runs the SQL Tuning Advisor on this SQL. The automated task performs the same comprehensive analysis as any other SQL Tuning task.

The automated task tests any SQL profiles it finds by executing both the old and new query plans. Automatic SQL Tuning differs from manual SQL tuning in one important way. If automatic implementation of SQL profiles is enabled (the default is disabled), then the database implements any SQL profiles that promise a great performance benefit. The implementation occurs at tuning time so that the database can immediately benefit from the new plan. You can enable or disable automatic implementation by using the SET\_AUTO\_TUNING\_TASK\_PARAMETER API to set the ACCEPT\_SQL\_PROFILES parameter.

In each maintenance window, the automated tuning task stores its results as a new execution. Each execution result has the same task name but a different execution name. Query the DBA\_ADVISOR\_EXECUTIONS view for information about task executions. Use the [REPORT\\_AUTO\\_TUNING\\_TASK Function](#) to view reports that span multiple executions.

## Security Model

This package is available to users with the DBA role. For other users, you must grant the EXECUTE privilege on the package explicitly. Note that the EXECUTE\_AUTO\_TUNING\_TASK procedure is an exception: only SYS can invoke it.

Users can call APIs in this package to control how the automatic tuning task behaves when it runs, such as enabling automatic SQL profile creation and configuring the total and per-SQL time limits under which the task runs. Because these settings affect the overall performance of the database, it may not be appropriate for all users with the ADVISOR privilege to have access to this package.

---

## Summary of DBMS\_AUTO\_SQLTUNE Subprograms

**Table 28–1 DBMS\_AUTO\_SQLTUNE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">EXECUTE_AUTO_TUNING_TASK Function &amp; Procedure</a> on page 28-6	Executes the Automatic SQL Tuning task immediately (SYS only)
<a href="#">REPORT_AUTO_TUNING_TASK Function</a> on page 28-7	Displays a text report of the automatic tuning task's history
<a href="#">SET_AUTO_TUNING_TASK_PARAMETER Procedures</a> on page 28-9	Changes a task parameter value for the daily automatic runs

## EXECUTE\_AUTO\_TUNING\_TASK Function & Procedure

This function and procedure executes the Automatic SQL Tuning task (`SYS_AUTO_SQL_TUNING_TASK`). Both the function and the procedure run in the context of a new task execution. The difference is that the function returns that new execution name. Note that only `SYS` can invoke this subprogram.

### Syntax

```
DBMS_AUTO_SQLTUNE.EXECUTE_AUTO_TUNING_TASK(
  execution_name  IN VARCHAR2           := NULL,
  execution_params IN dbms_advisor.argList := NULL,
  execution_desc  IN VARCHAR2           := NULL)
RETURN VARCHAR2;
```

```
DBMS_AUTO_SQLTUNE.EXECUTE_AUTO_TUNING_TASK(
  execution_name  IN VARCHAR2           := NULL,
  execution_params IN dbms_advisor.argList := NULL,
  execution_desc  IN VARCHAR2           := NULL);
```

### Parameters

**Table 28-2 EXECUTE\_TUNING\_TASK Function & Procedure Parameters**

Parameter	Description
<code>execution_name</code>	A name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by function.
<code>execution_params</code>	List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They override the values for the parameters stored in the task (set through the <a href="#">SET_AUTO_TUNING_TASK_PARAMETER Procedures</a> ).
<code>execution_desc</code>	A 256-length string describing the execution

### Usage Notes

A tuning task can be executed multiple times without having to reset it.

### Examples

```
EXEC DBMS_AUTO_SQLTUNE.EXECUTE_AUTO_TUNING_TASK(:exec_name);
```



## REPORT\_AUTO\_TUNING\_TASK Function

This procedure displays the results of an Automatic SQL Tuning task.

### Syntax

```
DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK (
  begin_exec      IN  VARCHAR2  := NULL,
  end_exec        IN  VARCHAR2  := NULL,
  type            IN  VARCHAR2  := 'TEXT',
  level           IN  VARCHAR2  := 'TYPICAL',
  section         IN  VARCHAR2  := ALL,
  object_id       IN  NUMBER     := NULL,
  result_limit    IN  NUMBER     := NULL)
RETURN CLOB;
```

### Parameters

**Table 28-3** REPORT\_TUNING\_TASK Function Parameters

Parameter	Description
begin_exec	Name of the beginning task execution to use. If NULL, the report is generated for the most recent task execution.
end_exec	Name of the ending task execution to use. If NULL, the report is generated for the most recent task execution.
type	Type of the report to produce. Possible values are TEXT which produces a text report.
level	Level of detail in the report: <ul style="list-style-type: none"> <li>▪ BASIC: simple version of the report. Just show info about the actions taken by the advisor.</li> <li>▪ TYPICAL: show information about every statement analyzed, including requests not implemented.</li> <li>▪ ALL: highly detailed report level, also provides annotations about statements skipped over.</li> </ul>
section	Section of the report to include: <ul style="list-style-type: none"> <li>▪ SUMMARY: summary information</li> <li>▪ FINDINGS: tuning findings</li> <li>▪ PLAN: explain plans</li> <li>▪ INFORMATION: general information</li> <li>▪ ERROR: statements with errors</li> <li>▪ ALL: all sections</li> </ul>
object_id	Advisor framework object id that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.
result_limit	Maximum number of SQL statements to show in the report

### Return Values

A CLOB containing the desired report.

### Examples

```
-- Get the whole report for the most recent execution
```

```
SELECT DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK
FROM   DUAL;

-- Show the summary for a range of executions
SELECT DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK(:begin_exec, :end_exec, 'TEXT',
        'TYPICAL', 'SUMMARY')
FROM   DUAL;

-- Show the findings for the statement of interest
SELECT DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK(:exec, :exec, 'TEXT',
        'TYPICAL', 'FINDINGS', 5)
FROM   DUAL;
```

## SET\_AUTO\_TUNING\_TASK\_PARAMETER Procedures

This procedure updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER as used for the reserved auto tuning task, SYS\_AUTO\_SQL\_TUNING\_TASK.

### Syntax

```
DBMS_AUTO_SQLTUNE.SET_AUTO_TUNING_TASK_PARAMETER (  
  parameter      IN  VARCHAR2,  
  value          IN  VARCHAR2);
```

```
DBMS_AUTO_SQLTUNE.SET_AUTO_TUNING_TASK_PARAMETER (  
  parameter      IN  VARCHAR2,  
  value          IN  NUMBER);
```

## Parameters

**Table 28–4 SET\_AUTO\_TUNING\_TASK\_PARAMETER Procedure Parameters**

Parameter	Description
parameter	<p>Name of the parameter to set. The possible tuning parameters that can be set by this procedure using the parameter in the form VARCHAR2:</p> <ul style="list-style-type: none"> <li>■ MODE: tuning scope (comprehensive, limited)</li> <li>■ USERNAME: username under which the statement is parsed</li> <li>■ DAYS_TO_EXPIRE: number of days until the task is deleted</li> <li>■ EXECUTION_DAYS_TO_EXPIRE: number of days until the tasks's executions is deleted (without deleting the task)</li> <li>■ DEFAULT_EXECUTION_TYPE: the task defaults to this type of execution when none is specified by the <a href="#">EXECUTE_AUTO_TUNING_TASK Function &amp; Procedure</a>.</li> <li>■ TIME_LIMIT: global time out (seconds)</li> <li>■ LOCAL_TIME_LIMIT: per-statement time out (seconds)</li> <li>■ TEST_EXECUTE: FULL/AUTO/OFF. <ul style="list-style-type: none"> <li>* FULL - test-execute for as much time as necessary, up to the local time limit for the SQL (or the global task time limit if no SQL time limit is set)</li> <li>* AUTO - test-execute for an automatically-chosen time proportional to the tuning time</li> <li>* OFF - do not test-execute</li> </ul> </li> <li>■ BASIC_FILTER: basic filter for SQL tuning set</li> <li>■ OBJECT_FILTER: object filter for SQL tuning set</li> <li>■ PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</li> <li>■ RANK_MEASURE1: first ranking measure for SQL tuning set</li> <li>■ RANK_MEASURE2: second possible ranking measure for SQL tuning set</li> <li>■ RANK_MEASURE3: third possible ranking measure for SQL tuning set</li> <li>■ RESUME_FILTER: a extra filter for SQL tuning sets besides BASIC_FILTER</li> <li>■ SQL_LIMIT: maximum number of SQL statements to tune</li> <li>■ SQL_PERCENTAGE: percentage filter of SQL tuning set statements</li> </ul> <p>The following parameters are supported for the automatic tuning task only:</p> <ul style="list-style-type: none"> <li>■ ACCEPT_SQL_PROFILES: TRUE/FALSE: whether the task should accept SQL profiles automatically</li> <li>■ MAX_AUTO_SQL_PROFILES: maximum number of automatic SQL profiles allowed on the system, in sum</li> <li>■ MAX_SQL_PROFILES_PER_EXEC: maximum number of SQL profiles that can be automatically implemented per execution of the task.</li> </ul>
value	New value of the specified parameter

---

---

## DBMS\_AUTO\_TASK\_ADMIN

The DBMS\_AUTO\_TASK\_ADMIN package provides an interface to AUTOTASK functionality. It is used by the DBA as well as Enterprise Manager to access the AUTOTASK controls. Enterprise Manager also uses the AUTOTASK Advisor.

**See Also:** *Oracle Database Administrator's Guide* for more information about "Configuring Automated Maintenance Task"

This chapter contains the following sections:

- [Using DBMS\\_AUTO\\_TASK\\_ADMIN](#)
  - Constants
- [Summary of DBMS\\_AUTO\\_TASK\\_ADMIN Subprograms](#)

---

## Using DBMS\_AUTO\_TASK\_ADMIN

- [Constants](#)

## Constants

The DBMS\_AUTO\_TASK\_ADMIN package uses the constants shown in [Table 29-1](#):

**Table 29-1 DBMS\_AUTO\_TASK\_ADMIN Constants**

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
PRIORITY_MEDIUM	VARCHAR2	'MEDIUM'	Task with this priority should be executed as time permits
PRIORITY_HIGH	VARCHAR2	'HIGH'	Task with this priority should be executed within the current Maintenance Window
PRIORITY_URGENT	VARCHAR2	'URGENT'	Task with this priority is to be executed at the earliest opportunity

## Summary of DBMS\_AUTO\_TASK\_ADMIN Subprograms

**Table 29–2 DBMS\_XMLSTORE Package Subprograms**

Method	Description
<a href="#">DISABLE Procedures</a> on page 29-5	Prevents AUTOTASK from executing any requests from a specified client or operation.
<a href="#">ENABLE Procedures</a> on page 29-6	Allows a previously disabled client, operation, target type, or individual target to be enabled under AUTOTASK control
<a href="#">GET_CLIENT_ATTRIBUTES Procedure</a> on page 29-7	Returns values of select client attributes
<a href="#">GET_P1_RESOURCES Procedure</a> on page 29-8	Returns percent of resources allocated to each AUTOTASK High Priority Consumer Groups
<a href="#">OVERRIDE_PRIORITY Procedures</a> on page 29-9	Manually overrides task priority.
<a href="#">SET_CLIENT_SERVICE Procedure</a> on page 29-10	Associates an AUTOTASK Client with a specified Service
<a href="#">SET_P1_RESOURCES Procedure</a> on page 29-11	Sets percentage-based resource allocation for each High Priority Consumer Group used by AUTOTASK Clients



## DISABLE Procedures

This procedure prevents AUTOTASK from executing any requests from a specified client or operation.

### Syntax

Disables all AUTOTASK functionality.

```
DBMS_AUTO_TASK_ADMIN.DISABLE;
```

Disables all tasks for the client or operation.

```
DBMS_AUTO_TASK_ADMIN.DISABLE (
  client_name      IN   VARCHAR2,
  operation        IN   VARCHAR2,
  window_name     IN   VARCHAR2);
```

### Parameters

**Table 29–3** *DISABLE Procedure Parameters*

Parameter	Description
client_name	Name of the client, as found in DBA_AUTOTASK_CLIENT View
operation	Name of the operation as specified in DBA_AUTOTASK_OPERATION View
window_name	Optional name of the window in which client is to be disabled

### Usage Notes

- If operation and window\_name are both NULL, the client is disabled.
- If operation is not NULL, window\_name is ignored and the operation is disabled
- If operation is NULL and window\_name is not NULL, the client is disabled in the specified window.

## ENABLE Procedures

This procedure allows a previously disabled client, operation, target type, or individual target to be enabled under AUTOTASK control. Specifying the DEFERRED option postpones the effect of the call until the start of the next maintenance window. If IMMEDIATE option is specified the effect of this call is immediate – as long as there is a currently open maintenance window.

### Syntax

Re-enabling AUTOTASK. This version enables the specified client. Note that any explicitly disabled tasks or operations must be re-enabled individually.

```
DBMS_AUTO_TASK_ADMIN.ENABLE;
```

Re-enabling a client or operation. Note that any explicitly disabled tasks or operations must be re-enabled individually.

```
DBMS_AUTO_TASK_ADMIN.ENABLE (
  client_name      IN   VARCHAR2,
  operation        IN   VARCHAR2,
  window_name     IN   VARCHAR2);
```

### Parameters

**Table 29–4** *ENABLE Procedure Parameters*

Parameter	Description
client_name	Name of the client, as found in DBA_AUTOTASK_CLIENT View
operation	Name of the operation as specified in DBA_AUTOTASK_OPERATION View
window_name	Optional name of the window in which client is to be enabled

### Usage Notes

- If operation and window\_name are both NULL, the client is enabled.
- If operation is not NULL, window\_name is ignored and the specified operation is enabled
- If operation is NULL and window\_name is not NULL, the client is enabled in the specified window.

## GET\_CLIENT\_ATTRIBUTES Procedure

This procedure returns values of select client attributes.

### Syntax

```
DBMS_AUTO_TASK_ADMIN.GET_CLIENT_ATTRIBUTES (  
  client_name      IN   VARCHAR2,  
  service_name     OUT  VARCHAR2,  
  window_group     OUT  VARCHAR2);
```

### Parameters

**Table 29-5** GET\_CLIENT\_ATTRIBUTES Procedure Parameters

Parameter	Description
client_name	Name of the client, as found in DBA_AUTOTASK_CLIENT View
service_name	Service name for client, may be NULL
window_group	Name of the window group in which the client is active

## GET\_P1\_RESOURCES Procedure

This procedure returns percent of resources allocated to each AUTOTASK High Priority Consumer Group.

### Syntax

```
DBMS_AUTO_TASK_ADMIN.GET_P1_RESOURCES (  
  stats_group_pct      OUT  NUMBER,  
  seg_group_pct        OUT  NUMBER,  
  tune_group_pct       OUT  NUMBER,  
  health_group_pct     OUT  NUMBER);
```

### Parameters

**Table 29–6** GET\_P1\_RESOURCES Procedure Parameters

Parameter	Description
stats_group_pct	%resources for Statistics Gathering
seg_group_pct	%resources for Space Management
tune_group_pct	%resources for SQL Tuning
health_group_pct	%resources for Health Checks

### Usage Notes

Values will add up to 100%.

## OVERRIDE\_PRIORITY Procedures

This procedure is used to manually override task priority. This can be done at the client, operation or individual task level. This priority assignment is honored during the next maintenance window in which the named client is active. Specifically, setting the priority to `URGENT` causes a high priority job to be generated at the start of the maintenance window. Setting `priority` to `CLEAR` removes the override.

### Syntax

Override Priority for a Client.

```
DBMS_AUTO_TASK_ADMIN.OVERRIDE_PRIORITY (
  client_name      IN   VARCHAR2,
  priority         IN   VARCHAR2);
```

Override Priority for an Operation.

```
DBMS_AUTO_TASK_ADMIN.OVERRIDE_PRIORITY (
  client_name      IN   VARCHAR2,
  operation        IN   VARCHAR2,
  priority         IN   VARCHAR2);
```

Override Priority for a Task.

```
DBMS_AUTO_TASK_ADMIN.OVERRIDE_PRIORITY (
  client_name      IN   VARCHAR2,
  operation        IN   VARCHAR2,
  task_target_type IN   VARCHAR2,
  task_target_name IN   VARCHAR2,
  priority         IN   VARCHAR2);
```

### Parameters

**Table 29–7** *OVERRIDE\_PRIORITY Procedure Parameters*

Parameter	Description
<code>client_name</code>	Name of the client, as found in <code>DBA_AUTOTASK_CLIENT</code> View
<code>priority</code>	<code>URGENT</code> , <code>HIGH</code> , <code>MEDIUM</code> or <code>LOW</code>
<code>operation</code>	Name of the operation as specified in <code>DBA_AUTOTASK_OPERATION</code> View
<code>task_target_type</code>	Type of target to be affected, as found in <code>V\$AUTOTASK_TARGET_TYPE</code> View
<code>task_target_name</code>	Name of the specific target to be affected

## SET\_CLIENT\_SERVICE Procedure

This procedure associates an AUTOTASK Client with a specified Service.

### Syntax

```
DBMS_AUTO_TASK_ADMIN.SET_CLIENT_SERVICE(  
    client_name      IN    VARCHAR2,  
    service_name    IN    VARCHAR2);
```

### Parameters

**Table 29–8 SET\_CLIENT\_SERVICE Procedure Parameters**

Parameter	Description
client_name	Name of the client, as found in DBA_AUTOTASK_CLIENT View
service_name	Service name for client, may be NULL

### Usage Notes

All work performed on behalf of the Client takes place only on instances where the service is enabled.

## SET\_P1\_RESOURCES Procedure

This procedure sets percentage-based resource allocation for each High Priority Consumer Group used by AUTOTASK Clients.

### Syntax

```
DBMS_AUTO_TASK_ADMIN.SET_P1_RESOURCES (
  stats_group_pct    OUT  NUMBER,
  seq_group_pct      OUT  NUMBER,
  tune_group_pct     OUT  NUMBER,
  health_group_pct   OUT  NUMBER);
```

### Parameters

**Table 29–9 SET\_P1\_RESOURCES Procedure Parameters**

Parameter	Description
stats_group_pct	%resources for Statistics Gathering
seq_group_pct	%resources for Space Management
tune_group_pct	%resources for SQL Tuning
health_group_pct	%resources for Health Checks

### Usage Notes

Values must be integers in the range 0 to 100, and must add up to 100 (percent), otherwise, an exception is raised.





---

---

## DBMS\_AUTO\_TASK\_IMMEDIATE

The DBMS\_AUTO\_TASK\_IMMEDIATE package consists of one subprogram whose function is to initiate gathering of optimizer statistics without delay.

This chapter contains the following topics:

- [Summary of DBMS\\_AUTO\\_TASK\\_IMMEDIATE Subprograms](#)

---

## Summary of DBMS\_AUTO\_TASK\_IMMEDIATE Subprograms

**Table 30–1** *DBMS\_AUTO\_TASK\_IMMEDIATE Subprograms*

Subprogram	Description
<a href="#">GATHER_OPTIMIZER_STATS Procedure</a> on page 30-3	Initiates gathering of optimizer statistics without waiting for the start of a maintenance window

## **GATHER\_OPTIMIZER\_STATS Procedure**

This procedure initiates gathering of optimizer statistics without waiting for the start of a maintenance window.

### **Syntax**

```
DBMS_AUTO_TASK_IMMEDIATE.GATHER_OPTIMIZER_STATS;
```



---

---

## DBMS\_AW\_STATS

DBMS\_AW\_STATS contains subprograms for managing optimizer statistics for cubes and dimensions. Generating the statistics does not have a significant performance cost.

**See Also:** *Oracle OLAP User's Guide* regarding use of the OLAP option to support business intelligence and analytical applications

This chapter contains the following topic:

- [Using DBMS\\_AW\\_STATS](#)
- [Summary of DBMS\\_AW\\_STATS Subprograms](#)

---

## Using DBMS\_AW\_STATS

Cubes and dimensions are first class data objects that support multidimensional analytics. They are stored in a container called an analytic workspace. Multidimensional objects and analytics are available with the OLAP option to Oracle Database.

Optimizer statistics are used to create execution plans for queries that join two cube views or join a cube view to a table or a view of a table. They are also used for query rewrite to cube materialized views. You need to generate the statistics only for these types of queries.

Queries against a single cube do not use optimizer statistics. These queries are automatically optimized within the analytic workspace.

## Summary of DBMS\_AW\_STATS Subprograms

**Table 31-1** *DBMS\_AW\_STATS Package Subprograms*

Subprogram	Description
<a href="#">ANALYZE Procedure</a> on page 31-4	Generates optimizer statistics on cubes and cube dimensions.
<a href="#">CLEAR Procedure</a> on page 31-7	Clears optimizer statistics from cubes and cube dimensions.

## ANALYZE Procedure

This procedure generates optimizer statistics on a cube or a cube dimension. These statistics are used to generate some execution plans, as described in ["Using DBMS\\_AW\\_STATS"](#) on page 31-2.

For a cube, the statistics are for all of the measures and calculated measures associated with the cube. These statistics include:

- The average length of data values
- The length of the largest data value
- The minimum value
- The number of distinct values
- The number of null values

For a dimension, the statistics are for the dimension and its attributes, levels, and hierarchies. These statistics include:

- The average length of a value
- The length of the largest value
- The minimum value
- The maximum value

### Syntax

```
DBMS_AW_STATS.ANALYZE
    (inname          IN VARCHAR2);
```

### Parameters

**Table 31–2 ANALYZE Procedure Parameters**

Parameter	Description
inname	The qualified name of a cube or a dimension. For a cube, the format of a qualified name is <i>owner.cube_name</i> . For a dimension, the format is <i>owner.dimension_name</i> .

### Usage Notes

Always analyze the dimensions first, then the cube.

After analyzing a dimension, analyze all cubes that use that dimension.

### Example

This sample script generates optimizer statistics on UNITS\_CUBE and its dimensions.

```
BEGIN
    DBMS_AW_STATS.ANALYZE('time');
    DBMS_AW_STATS.ANALYZE('customer');
    DBMS_AW_STATS.ANALYZE('product');
    DBMS_AW_STATS.ANALYZE('channel');
    DBMS_AW_STATS.ANALYZE('units_cube');
END;
/
```



The following statements create and display an execution plan for a SELECT statement that joins columns from UNITS\_CUBE\_VIEW, CUSTOMER\_PRIMARY\_VIEW, and the ACCOUNTS table:

```

EXPLAIN PLAN FOR SELECT
  cu.long_description customer,
  a.city city,
  a.zip_pc zip,
  cu.level_name "LEVEL",
  round(f.sales) sales
/* From dimension views and cube view */
FROM time_calendar_view t,
  product_primary_view p,
  customer_view cu,
  channel_view ch,
  units_cube_view f,
  account a
/* Create level filters instead of GROUP BY */
WHERE t.long_description = '2004'
  AND p.level_name = 'TOTAL'
  AND cu.customer_account_id like 'COMP%'
  AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
  AND t.dim_key = f.TIME
  AND p.dim_key = f.product
  AND cu.dim_key = f.customer
  AND ch.dim_key = f.channel
  AND a.account_id = cu.customer_account_id
ORDER BY zip;

```

```
SQL> SELECT plan_table_output FROM TABLE(dbms_xplan.display());
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 3890178023
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	89	6 (34)	00:00:01
1	SORT ORDER BY		1	89	6 (34)	00:00:01
* 2	HASH JOIN		1	89	5 (20)	00:00:01
3	JOINED CUBE SCAN PARTIAL OUTER					
4	CUBE ACCESS	UNITS_CUBE				
5	CUBE ACCESS	CHANNEL				
6	CUBE ACCESS	CUSTOMER				
7	CUBE ACCESS	PRODUCT				
* 8	CUBE ACCESS	TIME	1	55	2 (0)	00:00:01
* 9	TABLE ACCESS FULL	ACCOUNT	3	102	2 (0)	00:00:01

```
Predicate Information (identified by operation id):
```

```

-----
2 - access("A"."ACCOUNT_ID"=SYS_OP_ATG(VALUE(KOKBF$),39,40,2))
8 - filter(SYS_OP_ATG(VALUE(KOKBF$),16,17,2)='2004' AND
  SYS_OP_ATG(VALUE(KOKBF$),39,40,2) LIKE 'COMP%' AND
  SYS_OP_ATG(VALUE(KOKBF$),47,48,2)='TOTAL' AND
  SYS_OP_ATG(VALUE(KOKBF$),25,26,2)='TOTAL')
```

```
9 - filter("A"."ACCOUNT_ID" LIKE 'COMP%')
```

Note

-----

- dynamic sampling used for this statement

30 rows selected.

## CLEAR Procedure

Clears the statistics generated by the [ANALYZE Procedure](#).

### Syntax

```
DBMS_AW_STATS.CLEAR (
    inname          IN VARCHAR2;
```

### Parameters

**Table 31–3 CLEAR Procedure Parameters**

Parameter	Description
inname	The qualified name of a cube or a dimension. For a cube, the format of a qualified name is <i>owner.cube_name</i> . For a dimension, the format is <i>owner.dimension_name</i> .

### Examples

The following scripts clears the statistics from UNITS\_CUBE and its dimensions.

```
BEGIN
    DBMS_AW_STATS.clear('units_cube');
    DBMS_AW_STATS.clear('time');
    DBMS_AW_STATS.clear('customer');
    DBMS_AW_STATS.clear('product');
    DBMS_AW_STATS.clear('channel');
END;
/
```



---

---

## DBMS\_CAPTURE\_ADM

The DBMS\_CAPTURE\_ADM package, one of a set of Oracle Streams packages, provides subprograms for starting, stopping, and configuring a capture process. The source of the captured changes is the redo logs, and the repository for the captured changes is a queue.

**See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and capture processes

This chapter contains the following topics:

- [Using DBMS\\_CAPTURE\\_ADM](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_CAPTURE\\_ADM Subprograms](#)

---

## Using DBMS\_CAPTURE\_ADM

This section contains topics which relate to using the DBMS\_CAPTURE\_ADM package.

- [Overview](#)
- [Security Model](#)

## Overview

This package provides interfaces to start, stop, and configure a capture process or a synchronous capture. This package includes subprograms for preparing database objects for instantiation.

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

When the DBMS\_CAPTURE\_ADM package is used to manage an Oracle Streams configuration, it requires that the user is granted the privileges of an Oracle Streams administrator.

When the DBMS\_CAPTURE\_ADM package is used to manage an XStream configuration, it requires that the user is granted the privileges of an XStream administrator.

---

---

**Note:**

- The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as setting a capture user. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.
  - Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.
- 
- 

**See Also:**

- *Oracle Streams Concepts and Administration* for information about configuring an Oracle Streams administrator
- *Oracle Database XStream Guide* for information about configuring an XStream administrator



---

## Summary of DBMS\_CAPTURE\_ADM Subprograms

**Table 32–1 DBMS\_CAPTURE\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ABORT_GLOBAL_INSTANTIATION Procedure</a> on page 32-7	Reverses the effects of running the <code>PREPARE_GLOBAL_INSTANTIATION</code> , <code>PREPARE_SCHEMA_INSTANTIATION</code> , and <code>PREPARE_TABLE_INSTANTIATION</code> procedures
<a href="#">ABORT_SCHEMA_INSTANTIATION Procedure</a> on page 32-8	Reverses the effects of running the <code>PREPARE_SCHEMA_INSTANTIATION</code> and <code>PREPARE_TABLE_INSTANTIATION</code> procedures
<a href="#">ABORT_SYNC_INSTANTIATION Procedure</a> on page 32-9	Reverses the effects of running the <code>PREPARE_SYNC_INSTANTIATION</code> procedure
<a href="#">ABORT_TABLE_INSTANTIATION Procedure</a> on page 32-10	Reverses the effects of running the <code>PREPARE_TABLE_INSTANTIATION</code> procedure
<a href="#">ALTER_CAPTURE Procedure</a> on page 32-11	Alters a capture process
<a href="#">ALTER_SYNC_CAPTURE Procedure</a> on page 32-16	Alters a synchronous capture
<a href="#">BUILD Procedure</a> on page 32-18	Extracts the data dictionary of the current database to the redo logs and automatically specifies database supplemental logging for all primary key and unique key columns
<a href="#">CREATE_CAPTURE Procedure</a> on page 32-19	Creates a capture process
<a href="#">CREATE_SYNC_CAPTURE Procedure</a> on page 32-29	Creates a synchronous capture
<a href="#">DROP_CAPTURE Procedure</a> on page 32-31	Drops a capture process
<a href="#">INCLUDE_EXTRA_ATTRIBUTE Procedure</a> on page 32-33	Includes or excludes an extra attribute in logical change records (LCRs) captured by the specified capture process or synchronous capture
<a href="#">PREPARE_GLOBAL_INSTANTIATION Procedure</a> on page 32-35	Performs the synchronization necessary for instantiating all the tables in the database at another database and can enable supplemental logging for key columns or all columns in these tables
<a href="#">PREPARE_SCHEMA_INSTANTIATION Procedure</a> on page 32-36	Performs the synchronization necessary for instantiating all tables in the schema at another database and can enable supplemental logging for key columns or all columns in these tables
<a href="#">PREPARE_SYNC_INSTANTIATION Function</a> on page 32-37	Performs the synchronization necessary for instantiating one or more tables at another database and returns the prepare SCN
<a href="#">PREPARE_TABLE_INSTANTIATION Procedure</a> on page 32-38	Performs the synchronization necessary for instantiating the table at another database and can enable supplemental logging for key columns or all columns in the table
<a href="#">SET_PARAMETER Procedure</a> on page 32-39	Sets a capture process parameter to the specified value

**Table 32–1 (Cont.) DBMS\_CAPTURE\_ADM Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">START_CAPTURE Procedure</a> on page 32-49	Starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue
<a href="#">STOP_CAPTURE Procedure</a> on page 32-50	Stops the capture process from mining redo logs

---

---

**Note:** All subprograms commit unless specified otherwise.

---

---

## ABORT\_GLOBAL\_INSTANTIATION Procedure

This procedure reverses the effects of running the `PREPARE_GLOBAL_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, and `PREPARE_TABLE_INSTANTIATION` procedures.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to the database, schema, and table instantiations
- Removes any supplemental logging enabled by the `PREPARE_GLOBAL_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, and `PREPARE_TABLE_INSTANTIATION` procedures

### Syntax

```
DBMS_CAPTURE_ADM.ABORT_GLOBAL_INSTANTIATION;
```

## ABORT\_SCHEMA\_INSTANTIATION Procedure

This procedure reverses the effects of running the `PREPARE_SCHEMA_INSTANTIATION` procedure. It also reverses the effects of running the `PREPARE_TABLE_INSTANTIATION` procedure on tables in the specified schema.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to schema instantiations and table instantiations of tables in the schema
- Removes any supplemental logging enabled by the `PREPARE_SCHEMA_INSTANTIATION` procedure
- Removes any supplemental logging enabled by the `PREPARE_TABLE_INSTANTIATION` procedure for tables in the specified schema

### Syntax

```
DBMS_CAPTURE_ADM.ABORT_SCHEMA_INSTANTIATION(  
    schema_name IN VARCHAR2);
```

### Parameter

**Table 32–2** *ABORT\_SCHEMA\_INSTANTIATION Procedure Parameter*

Parameter	Description
<code>schema_name</code>	The name of the schema for which to abort the effects of preparing instantiation

## ABORT\_SYNC\_INSTANTIATION Procedure

This procedure reverses the effects of running the `PREPARE_SYNC_INSTANTIATION` procedure. Specifically, this procedure removes data dictionary information related to the table instantiation.

This procedure is overloaded. The `table_names` parameter is `VARCHAR2` data type in one version and `DBMS_UTILITY.UNCL_ARRAY` data type in the other version.

### Syntax

```
DBMS_CAPTURE_ADM.ABORT_SYNC_INSTANTIATION(
    table_names IN VARCHAR2);
```

```
DBMS_CAPTURE_ADM.ABORT_SYNC_INSTANTIATION(
    table_names IN DBMS_UTILITY.UNCL_ARRAY);
```

### Parameters

**Table 32–3** *ABORT\_SYNC\_INSTANTIATION Procedure Parameter*

Parameter	Description
<code>table_names</code>	<p>When the <code>table_names</code> parameter is <code>VARCHAR2</code> data type, a comma-delimited list of the tables for which to abort the effects of preparing instantiation. There must be no spaces between entries.</p> <p>When the <code>table_names</code> parameter is <code>DBMS_UTILITY.UNCL_ARRAY</code> data type, specify a PL/SQL associative array of this type that contains the names of the tables for which to abort the effects of preparing instantiation. The first table name is at position 1, the second at position 2, and so on. The table does not need to be <code>NULL</code> terminated.</p> <p>In either version of the procedure, specify the name of each table in the form <code>[schema_name.] table_name</code>. For example, <code>hr.employees</code>. If the schema is not specified, then the current user is the default.</p>

## ABORT\_TABLE\_INSTANTIATION Procedure

This procedure reverses the effects of running the `PREPARE_TABLE_INSTANTIATION` procedure.

Specifically, this procedure performs the following actions:

- Removes data dictionary information related to the table instantiation
- Removes any supplemental logging enabled by the `PREPARE_TABLE_INSTANTIATION` procedure

### Syntax

```
DBMS_CAPTURE_ADM.ABORT_TABLE_INSTANTIATION(  
    table_name IN VARCHAR2);
```

### Parameter

**Table 32–4** *ABORT\_TABLE\_INSTANTIATION Procedure Parameter*

Parameter	Description
<code>table_name</code>	The name of the table for which to abort the effects of preparing instantiation, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default.

## ALTER\_CAPTURE Procedure

This procedure alters a capture process.

---

**Note:** Starting with Oracle Database 11g Release 2 (11.2.0.2), the `start_time` parameter is included in this procedure.

---

**See Also:** *Oracle Streams Concepts and Administration* for more information about altering a capture process

### Syntax

```
DBMS_CAPTURE_ADM.ALTER_CAPTURE(
  capture_name          IN  VARCHAR2,
  rule_set_name        IN  VARCHAR2  DEFAULT NULL,
  remove_rule_set      IN  BOOLEAN   DEFAULT FALSE,
  start_scn            IN  NUMBER     DEFAULT NULL,
  use_database_link    IN  BOOLEAN   DEFAULT NULL,
  first_scn            IN  NUMBER     DEFAULT NULL,
  negative_rule_set_name IN  VARCHAR2  DEFAULT NULL,
  remove_negative_rule_set IN  BOOLEAN  DEFAULT FALSE,
  capture_user         IN  VARCHAR2  DEFAULT NULL,
  checkpoint_retention_time IN  NUMBER  DEFAULT NULL,
  start_time           IN  TIMESTAMP  DEFAULT NULL);
```

### Parameters

**Table 32–5 ALTER\_CAPTURE Procedure Parameters**

Parameter	Description
<code>capture_name</code>	The name of the capture process being altered. You must specify an existing capture process name. Do not specify an owner.
<code>rule_set_name</code>	<p>The name of the positive rule set for the capture process. The positive rule set contains the rules that instruct the capture process to capture changes.</p> <p>To change the positive rule set for the capture process, specify an existing rule set in the form [<i>schema_name.</i>] <i>rule_set_name</i>. For example, to specify a positive rule set in the <code>hr</code> schema named <code>job_capture_rules</code>, enter <code>hr.job_capture_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify <code>NULL</code> and the <code>remove_rule_set</code> parameter is set to <code>FALSE</code>, then the procedure retains any existing positive rule set. If you specify <code>NULL</code> and the <code>remove_rule_set</code> parameter is set to <code>TRUE</code>, then the procedure removes any existing positive rule set.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about the changes that can be captured by a capture process</p>

**Table 32–5 (Cont.) ALTER\_CAPTURE Procedure Parameters**

Parameter	Description
<code>remove_rule_set</code>	<p>If <code>TRUE</code>, then the procedure removes the positive rule set for the specified capture process. If you remove a positive rule set for a capture process, and the capture process does not have a negative rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the <code>SYS</code> and <code>SYSTEM</code> schemas.</p> <p>If you remove a positive rule set for a capture process, and the capture process has a negative rule set, then the capture process captures all supported changes that are not discarded by the negative rule set.</p> <p>If <code>FALSE</code>, then the procedure retains the positive rule set for the specified capture process.</p> <p>If the <code>rule_set_name</code> parameter is non-<code>NULL</code>, then ensure that this parameter is set to <code>FALSE</code>.</p>
<code>start_scn</code>	<p>A valid SCN for the database from which the capture process starts capturing changes. The SCN value must be greater than or equal to the first SCN for the capture process. Also, the capture process must be stopped before resetting its start SCN.</p> <p>An error is returned if an invalid SCN is specified or if the capture process is enabled.</p>
<code>use_database_link</code>	<p>If <code>TRUE</code>, then the capture process at a downstream database uses a database link to the source database for administrative purposes relating to the capture process. If you want a capture process that is not using a database link currently to begin using a database link, then specify <code>TRUE</code>. In this case, a database link with the same name as the global name of the source database must exist at the downstream database.</p> <p>If <code>FALSE</code>, then either the capture process is running on the source database, or the capture process at a downstream database does not use a database link to the source database. If you want a capture process that is using a database link currently to stop using a database link, then specify <code>FALSE</code>. In this case, you must prepare source database objects for instantiation manually when you add or change capture process rules that pertain to these objects.</p> <p>If <code>NULL</code>, then the current value of this parameter for the capture process is not changed.</p>



**Table 32-5 (Cont.) ALTER\_CAPTURE Procedure Parameters**

Parameter	Description
first_scn	<p>The lowest SCN in the redo log from which a capture process can capture changes. If you specify a new first SCN for the capture process, then the specified first SCN must meet the following requirements:</p> <ul style="list-style-type: none"> <li>■ It must be greater than the current first SCN for the capture process.</li> <li>■ It must be less than or equal to the current applied SCN for the capture process. However, this requirement does not apply if the current applied SCN for the capture process is zero.</li> <li>■ It must be less than or equal to the required checkpoint SCN for the capture process.</li> </ul> <p>An error is returned if the specified SCN does not meet the first three requirements. See "Usage Notes" on page 32-15 for information about determining an SCN value that meets all of these conditions.</p> <p>When the first SCN is modified, the capture process purges information from its LogMiner data dictionary that is required to restart it at an earlier SCN. See <a href="#">BUILD Procedure</a> on page 32-18 for more information about a LogMiner data dictionary.</p> <p>If the specified first SCN is higher than the current start SCN for the capture process, then the start SCN is set automatically to the new value of the first SCN.</p>
negative_rule_set_name	<p>The name of the negative rule set for the capture process. The negative rule set contains the rules that instruct the capture process to discard changes.</p> <p>To change the negative rule set for the capture process, specify an existing rule set in the form <code>[ schema_name. ] rule_set_name</code>. For example, to specify a negative rule set in the hr schema named neg_capture_rules, enter <code>hr.neg_capture_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</p> <p>If you specify NULL and the <code>remove_negative_rule_set</code> parameter is set to FALSE, then the procedure retains any existing negative rule set. If you specify NULL and the <code>remove_negative_rule_set</code> parameter is set to TRUE, then the procedure removes any existing negative rule set.</p> <p>If you specify both a positive and a negative rule set for a capture process, then the negative rule set is always evaluated first.</p>

**Table 32–5 (Cont.) ALTER\_CAPTURE Procedure Parameters**

Parameter	Description
<code>remove_negative_rule_set</code>	<p>If <code>TRUE</code>, then the procedure removes the negative rule set for the specified capture process. If you remove a negative rule set for a capture process, and the capture process does not have a positive rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the <code>SYS</code> and <code>SYSTEM</code> schemas.</p> <p>If you remove a negative rule set for a capture process, and a positive rule set exists for the capture process, then the capture process captures all changes that are not discarded by the positive rule set.</p> <p>If <code>FALSE</code>, then the procedure retains the negative rule set for the specified capture process.</p> <p>If the <code>negative_rule_set_name</code> parameter is non-NULL, then ensure that this parameter is set to <code>FALSE</code>.</p>
<code>capture_user</code>	<p>The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If <code>NULL</code>, then the capture user is not changed.</p> <p>To change the capture user, the user who invokes the <code>ALTER_CAPTURE</code> procedure must be granted the <code>DBA</code> role. Only the <code>SYS</code> user can set the <code>capture_user</code> to <code>SYS</code>.</p> <p>If you change the capture user, then this procedure grants the new capture user <code>enqueue</code> privilege on the queue used by the capture process and configures the user as a secure queue user of the queue. In addition, ensure that the capture user has the following privileges:</p> <ul style="list-style-type: none"> <li>▪ <code>EXECUTE</code> privilege on the rule sets used by the capture process</li> <li>▪ <code>EXECUTE</code> privilege on all rule-based transformation functions used in the rule set</li> </ul> <p>These privileges can be granted directly to the capture user, or they can be granted through roles.</p> <p>In addition, the capture user must be granted <code>EXECUTE</code> privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles.</p> <p>The capture process is stopped and restarted automatically when you change the value of this parameter.</p> <p><b>Note:</b> If the capture user for a capture process is dropped using <code>DROP USER . . . CASCADE</code>, then the capture process is also dropped automatically.</p>

**Table 32–5 (Cont.) ALTER\_CAPTURE Procedure Parameters**

Parameter	Description
checkpoint_retention_time	<p>Either the number of days that a capture process retains checkpoints before purging them automatically, or DBMS_CAPTURE_ADM.INFINITE if checkpoints should not be purged automatically. If NULL, then the checkpoint retention time is not changed.</p> <p>If a number is specified, then a capture process purges a checkpoint the specified number of days after the checkpoint was taken. Partial days can be specified using decimal values. For example, .25 specifies 6 hours.</p> <p>When a checkpoint is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the first_scn of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about checkpoint retention time</p>
start_time	<p>A valid time from which the capture process starts capturing changes. The capture process must be stopped before resetting its start time.</p> <p>An error is returned if an invalid time is specified or if the capture process is enabled.</p> <p>The start_scn and start_time parameters are mutually exclusive.</p>

## Usage Notes

If you want to alter the first SCN for a capture process, then the value specified must meet the conditions in the description for the first\_scn parameter.

## Examples

The following query determines the current first SCN, applied SCN, and required checkpoint SCN for each capture process in a database:

```
SELECT CAPTURE_NAME, FIRST_SCN, APPLIED_SCN, REQUIRED_CHECKPOINT_SCN
       FROM DBA_CAPTURE;
```

## ALTER\_SYNC\_CAPTURE Procedure

This procedure alters a synchronous capture.

**See Also:** *Oracle Streams Concepts and Administration* for more information about altering a capture process

### Syntax

```
DBMS_CAPTURE_ADM.ALTER_SYNC_CAPTURE (
    capture_name IN VARCHAR2,
    rule_set_name IN VARCHAR2 DEFAULT NULL,
    capture_user IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 32–6 ALTER\_SYNC\_CAPTURE Procedure Parameters**

Parameter	Description
capture_name	The name of the synchronous capture being altered. You must specify an existing synchronous capture name. Do not specify an owner.
rule_set_name	<p>The name of the positive rule set for the synchronous capture. The positive rule set contains the rules that instruct the synchronous capture to capture changes.</p> <p>To change the rule set for the synchronous capture, specify an existing rule set in the form <code>[schema_name.] rule_set_name</code>. For example, to specify a positive rule set in the <code>strmadmin</code> schema named <code>sync_cap_rules</code>, enter <code>strmadmin.sync_cap_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You must specify a rule set that was created using the <code>DBMS_STREAMS_ADM</code> package.</p> <p>If <code>NULL</code>, then the rule set is not changed.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about the changes that can be captured by a synchronous capture</p>
capture_user	<p>The user in whose security domain a synchronous capture captures changes that satisfy its rule set and runs custom rule-based transformations configured for synchronous capture rules. If <code>NULL</code>, then the capture user is not changed.</p> <p>To change the capture user, the user who invokes the <code>ALTER_SYNC_CAPTURE</code> procedure must be granted the <code>DBA</code> role. Only the <code>SYS</code> user can set the <code>capture_user</code> to <code>SYS</code>.</p> <p>If you change the capture user, then this procedure grants the new capture user <code>enqueue</code> privilege on the queue used by the synchronous capture and configures the user as a secure queue user of the queue. In addition, ensure that capture user has the following privileges:</p> <ul style="list-style-type: none"> <li>▪ <code>EXECUTE</code> privilege on the rule sets used by the synchronous capture</li> <li>▪ <code>EXECUTE</code> privilege on all rule-based transformation functions used in the rule set</li> </ul> <p>These privileges can be granted directly to the capture user, or they can be granted through roles.</p> <p>In addition, the capture user must be granted <code>EXECUTE</code> privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the synchronous capture. These privileges must be granted directly to the capture user. They cannot be granted through roles.</p>

## Usage Notes

If the capture user for a synchronous capture is dropped using `DROP USER . . . CASCADE`, then the synchronous capture is also dropped automatically.

## BUILD Procedure

This procedure extracts the data dictionary of the current database to the redo log and automatically specifies database supplemental logging by running the following SQL statement:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

This procedure is overloaded. One version of this procedure contains the OUT parameter `first_scn`, and the other does not.

### Syntax

```
DBMS_CAPTURE_ADM.BUILD(  
    first_scn OUT NUMBER);
```

```
DBMS_CAPTURE_ADM.BUILD;
```

### Parameters

**Table 32–7** BUILD Procedure Parameter

Parameter	Description
<code>first_scn</code>	Contains the lowest SCN value corresponding to the data dictionary extracted to the redo log that can be specified as a first SCN for a capture process

### Usage Notes

The following usage notes apply to this procedure:

- You can run this procedure multiple times at a source database.
- If you plan to capture changes originating at a source database with a capture process, then this procedure must be executed at the source database at least once. When the capture process is started, either at a local source database or at a downstream database, the capture process uses the extracted information in the redo log to create a LogMiner data dictionary.
- A LogMiner data dictionary is a separate data dictionary used by a capture process to determine the details of a change that it is capturing. The LogMiner data dictionary is necessary because the primary data dictionary of the source database might not be synchronized with the redo data being scanned by a capture process.
- After executing this procedure, you can query the `FIRST_CHANGE#` column of the `V$ARCHIVED_LOG` dynamic performance view where the `DICTIONARY_BEGIN` column is `YES` to determine the lowest SCN value for the database that can be specified as a first SCN for a capture process. The first SCN for a capture process is the lowest SCN in the redo log from which the capture process can capture changes. You can specify the first SCN for a capture process when you run the `CREATE_CAPTURE` or `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

## CREATE\_CAPTURE Procedure

This procedure creates a capture process.

---

**Note:** Starting with Oracle Database 11g Release 2 (11.2.0.2), the `start_time` parameter is included in this procedure.

---

### See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about creating a capture process
- [Chapter 127, "DBMS\\_RULE\\_ADM"](#) for more information about rules and rule sets

## Syntax

```
DBMS_CAPTURE_ADM.CREATE_CAPTURE (
  queue_name           IN  VARCHAR2,
  capture_name        IN  VARCHAR2,
  rule_set_name       IN  VARCHAR2  DEFAULT NULL,
  start_scn           IN  NUMBER    DEFAULT NULL,
  source_database     IN  VARCHAR2  DEFAULT NULL,
  use_database_link   IN  BOOLEAN   DEFAULT FALSE,
  first_scn           IN  NUMBER    DEFAULT NULL,
  logfile_assignment  IN  VARCHAR2  DEFAULT 'implicit',
  negative_rule_set_name IN VARCHAR2  DEFAULT NULL,
  capture_user        IN  VARCHAR2  DEFAULT NULL,
  checkpoint_retention_time IN NUMBER  DEFAULT 60,
  start_time          IN  TIMESTAMP  DEFAULT NULL);
```

## Parameters

**Table 32–8 CREATE\_CAPTURE Procedure Parameters**

Parameter	Description
<code>queue_name</code>	The name of the queue into which the capture process enqueues changes. You must specify an existing queue in the form [ <i>schema_name</i> .] <i>queue_name</i> . For example, to specify a queue in the <code>hr</code> schema named <code>streams_queue</code> , enter <code>hr.streams_queue</code> . If the schema is not specified, then the current user is the default.  <b>Note:</b> The <code>queue_name</code> setting cannot be altered after the capture process is created.
<code>capture_name</code>	The name of the capture process being created. A <code>NULL</code> specification is not allowed. Do not specify an owner.  <b>Note:</b> The <code>capture_name</code> setting cannot be altered after the capture process is created.

**Table 32–8 (Cont.) CREATE\_CAPTURE Procedure Parameters**

Parameter	Description
rule_set_name	<p>The name of the positive rule set for the capture process. The positive rule set contains the rules that instruct the capture process to capture changes.</p> <p>If you want to use a positive rule set for the capture process, then you must specify an existing rule set in the form [<i>schema_name.</i>] <i>rule_set_name</i>. For example, to specify a positive rule set in the hr schema named job_capture_rules, enter hr.job_capture_rules. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package.</p> <p>If you specify NULL, and no negative rule set is specified, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS and SYSTEM schemas.</p> <p>If you specify NULL, and a negative rule set exists for the capture process, then the capture process captures all changes that are not discarded by the negative rule set.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about the changes that can be captured by a capture process</p>
start_scn	<p>A valid SCN for the database from which the capture process starts capturing changes.</p> <p>An error is returned if an invalid SCN is specified.</p> <p>The start_scn and start_time parameters are mutually exclusive.</p> <p><b>See Also:</b> "Usage Notes" on page 32-23 for more information setting the start_scn parameter</p>
source_database	<p>The global name of the source database. The source database is where the changes to be captured originated.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically.</p> <p>If NULL, or if the specified name is the same as the global name of the current database, then local capture is assumed and only the default values for use_database_link and first_scn can be specified.</p>



**Table 32–8 (Cont.) CREATE\_CAPTURE Procedure Parameters**

Parameter	Description
use_database_link	<p>If TRUE, then the capture process at a downstream database uses a database link to the source database for administrative purposes relating to the capture process. A database link with the same name as the global name of the source database must exist at the downstream database.</p> <p>The capture process uses the database link to prepare database objects for instantiation at the source database and run the DBMS_CAPTURE_ADM.BUILD procedure at the source database, if necessary.</p> <p>During the creation of a downstream capture process, if the first_scn parameter is set to NULL, then the use_database_link parameter must be set to TRUE. Otherwise, an error is returned.</p> <p>If FALSE, then either the capture process is running on the source database, or the capture process at a downstream database does not use a database link to the source database. In this case, you must perform the following administrative tasks manually:</p> <ul style="list-style-type: none"> <li>■ Run the DBMS_CAPTURE_ADM.BUILD procedure at the source database to extract the data dictionary at the source database to the redo log when a capture process is created.</li> <li>■ Obtain the first SCN for the downstream capture process if the first SCN is not specified during capture process creation. The first SCN is needed to create and maintain a capture process.</li> <li>■ Prepare source database objects for instantiation.</li> </ul>
first_scn	<p>The lowest SCN in the redo log from which a capture process can capture changes. A non-NULL value for this parameter is valid only if the DBMS_CAPTURE_ADM.BUILD procedure has been run at least once at the source database.</p> <p>You can query the FIRST_CHANGE# column of the V\$ARCHIVED_LOG dynamic performance view where the DICTIONARY_BEGIN column is YES to determine whether the DBMS_CAPTURE_ADM.BUILD procedure has been run on a source database. Any of the values returned by such a query can be used as a first_scn value if the redo log containing that SCN value is still available.</p> <p><b>See Also:</b> "Usage Notes" on page 32-23 for more information setting the first_scn parameter</p>

**Table 32–8 (Cont.) CREATE\_CAPTURE Procedure Parameters**

Parameter	Description
logfile_assignment	<p>If <code>implicit</code>, which is the default, then the capture process at a downstream database scans all redo log files added by redo transport services or manually from the source database to the downstream database.</p> <p>If <code>explicit</code>, then a redo log file is scanned by a capture process at a downstream database only if the capture process name is specified in the <code>FOR logminer_session_name</code> clause. If <code>explicit</code>, then the redo log file must be added manually to the downstream database, and redo transport services cannot be used to add redo log files to the capture process being created.</p> <p>If you specify <code>explicit</code> for this parameter for a local capture process, then the local capture process cannot use the online redo log to find changes. In this case, the capture process must use the archived redo log.</p> <p><b>See Also:</b> "Usage Notes" on page 32-23 for information about adding redo log files manually</p>
negative_rule_set_name	<p>The name of the negative rule set for the capture process. The negative rule set contains the rules that instruct the capture process to discard changes.</p> <p>If you want to use a negative rule set for the capture process, then you must specify an existing rule set in the form <code>[ schema_name. ] rule_set_name</code>. For example, to specify a negative rule set in the <code>hr</code> schema named <code>neg_capture_rules</code>, enter <code>hr.neg_capture_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>If you specify <code>NULL</code>, and no positive rule set is specified, then the capture process captures all supported changes to all objects in the database, excluding database objects in the <code>SYS</code> and <code>SYSTEM</code> schemas.</p> <p>If you specify <code>NULL</code>, and a positive rule set exists for the capture process, then the capture process captures all changes that are not discarded by the positive rule set.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify both a positive and a negative rule set for a capture process, then the negative rule set is always evaluated first.</p>
capture_user	<p>The user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules. If <code>NULL</code>, then the user who runs the <code>CREATE_CAPTURE</code> procedure is used.</p> <p><b>Note:</b> If the capture user for a capture process is dropped using <code>DROP USER . . . CASCADE</code>, then the capture process is also dropped automatically.</p> <p><b>See Also:</b> "Usage Notes" on page 32-23 for more information about this parameter.</p>

**Table 32–8 (Cont.) CREATE\_CAPTURE Procedure Parameters**

Parameter	Description
checkpoint_retention_time	<p>Either specify the number of days that a capture process retains checkpoints before purging them automatically, or specify DBMS_CAPTURE_ADM.INFINITE if checkpoints should not be purged automatically.</p> <p>If a number is specified, then a capture process purges a checkpoint the specified number of days after the checkpoint was taken. Partial days can be specified using decimal values. For example, .25 specifies 6 hours.</p> <p>When a checkpoint is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the first_scn of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about checkpoint retention time</p>
start_time	<p>A valid time from which the capture process starts capturing changes.</p> <p>An error is returned if an invalid time is specified.</p> <p>The start_scn and start_time parameters are mutually exclusive.</p> <p><b>See Also:</b> "<a href="#">Usage Notes</a>" on page 32-23 for more information setting the start_time parameter</p>

## Usage Notes

Consider the following usage notes when you run this procedure:

- [DBA Role Requirement](#)
- [Capture User Requirements](#)
- [First SCN and Start SCN Settings](#)
- [Explicit Log File Assignment](#)

### DBA Role Requirement

If the user who invokes this procedure is different from the user specified in the capture\_user parameter, then the invoking user must be granted the DBA role. If the user who invokes this procedure is the same as the user specified in the capture\_user parameter, then the DBA role is not required for the invoking user. Only the SYS user can set the capture\_user to SYS.

### Capture User Requirements

The capture\_user parameter specifies the user who captures changes that satisfy the capture process rule sets. This user must have the necessary privileges to capture changes. This procedure grants the capture user enqueue privilege on the queue used by the capture process and configures the user as a secure queue user of the queue.

In addition, ensure that the capture user has the following privileges:

- EXECUTE privilege on the rule sets used by the capture process

- EXECUTE privilege on all rule-based transformation functions used in the positive rule set

These privileges can be granted directly to the capture user, or they can be granted through roles.

In addition, the capture user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles.

---

---

**Note:**

- A capture user does not require privileges on a database object to capture changes to the database object. The capture process can pass these changes to a rule-based transformation function. Therefore, ensure that you consider security implications when you configure a capture process.
  - Creation of the first capture process in a database might take some time because the data dictionary is duplicated during this creation.
- 
- 

**First SCN and Start SCN Settings**

When you create a capture process using this procedure, you can specify the first SCN and start SCN for the capture process. A capture process scans the redo data from the first SCN or an existing capture process checkpoint forward, even if the start SCN is higher than the first SCN or the checkpoint SCN. In this case, the capture process does not capture any changes in the redo data before the start SCN. Oracle recommends that, at capture process creation time, the difference between the first SCN and start SCN be as small as possible to keep the amount of redo scanned by the capture process to a minimum.

---

---

**Note:** When you specify the `start_time` parameter instead of the `start_scn` parameter, the `start_time` corresponds with a specific SCN. In this case, the information in this section also applies to the SCN that corresponds with the specified `start_time`.

---

---

In some cases, the behavior of the capture process is different depending on the settings of these SCN values and on whether the capture process is local or downstream.

The following table describes capture process behavior for SCN value settings:

<b>first_scn Setting</b>	<b>start_scn Setting</b>	<b>Capture Process Type</b>	<b>Description</b>
Non-NULL	NULL	Local or Downstream	<p>The new capture process is created at the local database with a new LogMiner session starting from the value specified for the <code>first_scn</code> parameter. The start SCN is set to the specified first SCN value automatically, and the new capture process does not capture changes that were made before this SCN.</p> <p>The <code>BUILD</code> procedure in the <code>DBMS_CAPTURE_ADM</code> package is not run automatically. This procedure must have been run at least once before on the source database, and the specified first SCN must correspond to the SCN value of a previous build that is still available in the redo log. When the new capture process is started for the first time, it creates a new LogMiner data dictionary using the data dictionary information in the redo log. If the <code>BUILD</code> procedure has not been run at least once on the source database, then an error is raised when the capture process is started.</p> <p>Capture process behavior is the same for a local capture process and a downstream capture process created with these SCN settings, except that a local capture process is created at the source database and a downstream capture process is created at the downstream database.</p>
Non-NULL	Non-NULL	Local or Downstream	<p>If the specified value for the <code>start_scn</code> parameter is greater than or equal to the specified value for the <code>first_scn</code> parameter, then the new capture process is created at the local database with a new LogMiner session starting from the specified first SCN. In this case, the new capture process does not capture changes that were made before the specified start SCN. If the specified value for the <code>start_scn</code> parameter is less than the specified value for the <code>first_scn</code> parameter, then an error is raised.</p> <p>The <code>BUILD</code> procedure in the <code>DBMS_CAPTURE_ADM</code> package is not run automatically. This procedure must have been called at least once before on the source database, and the specified <code>first_scn</code> must correspond to the SCN value of a previous build that is still available in the redo log. When the new capture process is started for the first time, it creates a new LogMiner data dictionary using the data dictionary information in the redo log. If the <code>BUILD</code> procedure has not been run at least once on the source database, then an error is raised.</p> <p>Capture process behavior is the same for a local capture process and a downstream capture process created with these SCN settings, except that a local capture process is created at the source database and a downstream capture process is created at the downstream database.</p>

<b>first_scn Setting</b>	<b>start_scn Setting</b>	<b>Capture Process Type</b>	<b>Description</b>
NULL	Non-NULL	Local	<p>The new capture process creates a new LogMiner data dictionary if either one of the following conditions is true:</p> <ul style="list-style-type: none"> <li>■ There is no existing capture process for the local source database, and the specified value for the <code>start_scn</code> parameter is greater than or equal to the current SCN for the database.</li> <li>■ There are existing capture processes, but none of the capture processes have taken a checkpoint yet, and the specified value for the <code>start_scn</code> parameter is greater than or equal to the current SCN for the database.</li> </ul> <p>In either of these cases, the <code>BUILD</code> procedure in the <code>DBMS_CAPTURE_ADM</code> package is run during capture process creation. The new capture process uses the resulting build of the source data dictionary in the redo log to create a LogMiner data dictionary the first time it is started, and the first SCN corresponds to the SCN of the data dictionary build. If there are any in-flight transactions, then the <code>BUILD</code> procedure waits until these transactions commit before completing. An in-flight transaction is one that is active during capture process creation or a data dictionary build.</p> <p>However, if there is at least one existing local capture process for the local source database that has taken a checkpoint, then the new capture process shares an existing LogMiner data dictionary with one or more of the existing capture processes. In this case, a capture process with a first SCN that is lower than or equal to the specified start SCN must have been started successfully at least once. Also, if there are any in-flight transactions, then the capture process is created after these transactions commit.</p> <p>If there is no existing capture process for the local source database (or if no existing capture processes have taken a checkpoint yet), and the specified start SCN is less than the current SCN for the database, then an error is raised.</p>

first_scn Setting	start_scn Setting	Capture Process Type	Description
NULL	Non-NULL	Downstream	<p>When the CREATE_CAPTURE procedure creates a downstream capture process, the use_database_link parameter must be set to TRUE when the first_scn parameter is set to NULL. Otherwise, an error is raised. The database link is used to obtain the current SCN of the source database.</p> <p>The new capture process creates a new LogMiner data dictionary if either one of the following conditions is true:</p> <ul style="list-style-type: none"> <li>There is no existing capture process that captures changes to the source database at the downstream database, and the specified value for the start_scn parameter is greater than or equal to the current SCN for the source database.</li> <li>There are existing capture processes that capture changes to the source database at the downstream database, but none of the capture processes have taken a checkpoint yet, and the specified value for the start_scn parameter is greater than or equal to the current SCN for the source database.</li> </ul> <p>In either of these cases, the BUILD procedure in the DBMS_CAPTURE_ADM package is run during capture process creation. The first time you start the new capture process, it uses the resulting build of the source data dictionary in the redo log files copied to the downstream database to create a LogMiner data dictionary. Here, the first SCN for the new capture process corresponds to the SCN of the data dictionary build. If there are any in-flight transactions, then the BUILD procedure waits until these transactions commit before completing.</p> <p>However, if at least one existing capture process has taken a checkpoint and captures changes to the source database at the downstream database, then the new capture process shares an existing LogMiner data dictionary with one or more of these existing capture processes. In this case, one of these existing capture processes with a first SCN that is lower than or equal to the specified start SCN must have been started successfully at least once. Also, if there are any in-flight transactions, then the capture process is created after these transactions commit.</p> <p>If there is no existing capture process that captures changes to the source database at the downstream database (or no existing capture process has taken a checkpoint), and the specified start_scn parameter value is less than the current SCN for the source database, then an error is raised.</p>
NULL	NULL	Local or Downstream	<p>The behavior is the same as setting the first_scn parameter to NULL and setting the start_scn parameter to the current SCN of the source database.</p>

---

**Note:** When you create a capture process using the DBMS\_STREAMS\_ADM package, both the first SCN and the start SCN are set to NULL during capture process creation.

---

**See Also:** [BUILD Procedure](#) on page 32-18 for more information about the BUILD procedure and the LogMiner data dictionary

### Explicit Log File Assignment

If you specify explicit for the logfile\_assignment parameter, then you add a redo log file manually to a downstream database using the following statement:

```
ALTER DATABASE REGISTER LOGICAL LOGFILE  
    file_name FOR capture_process;
```

Here, *file\_name* is the name of the redo log file being added and *capture\_process* is the name of the capture process that will use the redo log file at the downstream database. The *capture\_process* is equivalent to the *logminer\_session\_name* and must be specified. The redo log file must be present at the site running the downstream database. You must transfer this file manually to the site running the downstream database using the DBMS\_FILE\_TRANSFER package, FTP, or some other transfer method.

**See Also:** *Oracle Database SQL Language Reference* for more information about the ALTER DATABASE statement and *Oracle Data Guard Concepts and Administration* for more information registering redo log files



## CREATE\_SYNC\_CAPTURE Procedure

This procedure creates a synchronous capture.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about creating a synchronous capture

### Syntax

```
DBMS_CAPTURE_ADM.CREATE_SYNC_CAPTURE (
  queue_name      IN  VARCHAR2,
  capture_name    IN  VARCHAR2,
  rule_set_name   IN  VARCHAR2,
  capture_user    IN  VARCHAR2  DEFAULT NUL);
```

### Parameters

**Table 32–9** CREATE\_SYNC\_CAPTURE Procedure Parameters

Parameter	Description
queue_name	<p>The name of the queue into which the synchronous capture enqueues changes. You must specify an existing queue in the form [<i>schema_name</i>.] <i>queue_name</i>. For example, to specify a queue in the <i>strmadmin</i> schema named <i>streams_queue</i>, enter <i>strmadmin.streams_queue</i>. If the schema is not specified, then the current user is the default.</p> <p><b>Note:</b> The <i>queue_name</i> setting cannot be altered after the synchronous capture is created.</p>
capture_name	<p>The name of the synchronous capture being created. A NULL specification is not allowed. Do not specify an owner.</p> <p><b>Note:</b> The <i>capture_name</i> setting cannot be altered after the synchronous capture is created.</p>
rule_set_name	<p>The name of the positive rule set for the synchronous capture. The positive rule set contains the rules that instruct the synchronous capture to capture changes.</p> <p>Specify an existing rule set in the form [<i>schema_name</i>.] <i>rule_set_name</i>. For example, to specify a positive rule set in the <i>strmadmin</i> schema named <i>sync_cap_rules</i>, enter <i>strmadmin.sync_cap_rules</i>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You must specify a rule set that was created using the <i>DBMS_STREAMS_ADM</i> package.</p> <p>If NULL, then an error is returned.</p> <p><b>Note:</b> Synchronous capture rules must be added to the synchronous capture rule set using the <i>ADD_TABLE_RULES</i> or <i>ADD_SUBSET_RULES</i> procedure in the <i>DBMS_STREAMS_ADM</i> package. A synchronous capture ignores rules added to the rule set with other procedures.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about the changes that can be captured by a synchronous capture</p>

**Table 32–9 (Cont.) CREATE\_SYNC\_CAPTURE Procedure Parameters**

Parameter	Description
capture_user	<p>The user in whose security domain the synchronous capture captures changes that satisfy its rule set and runs custom rule-based transformations configured for synchronous capture rules. If NULL, then the user who runs the CREATE_SYNC_CAPTURE procedure is used.</p> <p>Only a user who is granted the DBA role can set a capture user. Only the SYS user can set the capture_user to SYS.</p> <p><b>Note:</b> If the capture user for a synchronous capture is dropped using DROP USER . . . CASCADE, then the synchronous capture is also dropped automatically.</p> <p><b>See Also:</b> "Usage Notes" on page 32-30 for more information about this parameter.</p>

## Usage Notes

When the CREATE\_SYNC\_CAPTURE procedure creates a synchronous capture, the procedure must obtain an exclusive lock on each table for which it will capture changes. The rules in the specified rule set for the synchronous capture determine these tables. If there are outstanding transactions on a table for which the synchronous capture will capture changes, then the procedure waits until it can obtain a lock.

The capture\_user parameter specifies the user who captures changes that satisfy the synchronous capture rule set. This user must have the necessary privileges to capture changes.

In addition, ensure that the capture user has the following privileges:

- ENQUEUE privilege on the queue specified in the queue\_name parameter
- EXECUTE privilege on the rule set used by the synchronous capture
- EXECUTE privilege on all rule-based transformation functions used in the rule set

These privileges can be granted directly to the capture user, or they can be granted through roles.

In addition, the capture user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the synchronous capture. These privileges must be granted directly to the capture user. These privileges cannot be granted through roles.

---



---

**Note:** A capture user does not require privileges on a database object to capture changes to the database object. The synchronous capture can pass these changes to a rule-based transformation function. Therefore, ensure that you consider security implications when you configure a synchronous capture.

---



---

## DROP\_CAPTURE Procedure

This procedure drops a capture process.

### Syntax

```
DBMS_CAPTURE_ADM.DROP_CAPTURE(
  capture_name          IN VARCHAR2,
  drop_unused_rule_sets IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 32–10 DROP\_CAPTURE Procedure Parameters**

Parameter	Description
capture_name	The name of the capture process being dropped. Specify an existing capture process name. Do not specify an owner.
drop_unused_rule_sets	If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified capture process if these rule sets are not used by any other Oracle Streams client. Oracle Streams clients include capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set.  If FALSE, then the procedure does not drop the rule sets used by the specified capture process, and the rule sets retain their rules.

### Usage Notes

The following usage notes apply to this procedure:

- [The Capture Process Must Be Stopped Before It Is Dropped](#)
- [The DROP\\_CAPTURE Procedure and Rules-related Information](#)

#### **The Capture Process Must Be Stopped Before It Is Dropped**

A capture process must be stopped before it can be dropped.

**See Also:** [STOP\\_CAPTURE Procedure](#) on page 32-50

#### **The DROP\_CAPTURE Procedure and Rules-related Information**

When you use this procedure to drop a capture process, rules-related information for the capture process created by the DBMS\_STREAMS\_ADM package is removed from the data dictionary views for Oracle Streams rules. Information about such a rule is removed even if the rule is not in either rule set for the capture process.

The following are the data dictionary views for Oracle Streams rules:

- ALL\_STREAMS\_GLOBAL\_RULES
- DBA\_STREAMS\_GLOBAL\_RULES
- ALL\_STREAMS\_MESSAGE\_RULES
- DBA\_STREAMS\_MESSAGE\_RULES
- ALL\_STREAMS\_SCHEMA\_RULES
- DBA\_STREAMS\_SCHEMA\_RULES

- ALL\_STREAMS\_TABLE\_RULES
- DBA\_STREAMS\_TABLE\_RULES
- ALL\_STREAMS\_RULES
- DBA\_STREAMS\_RULES

**See Also:** *Oracle Streams Concepts and Administration* for more information about Oracle Streams data dictionary views

## INCLUDE\_EXTRA\_ATTRIBUTE Procedure

This procedure includes or excludes an extra attribute in logical change records (LCRs) captured by the specified capture process or synchronous capture.

### Syntax

```
DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE (
  capture_name   IN  VARCHAR2,
  attribute_name IN  VARCHAR2,
  include        IN  BOOLEAN   DEFAULT TRUE);
```

### Parameters

**Table 32–11 INCLUDE\_EXTRA\_ATTRIBUTE Procedure Parameters**

Parameter	Description
capture_name	The name of the capture process or synchronous capture. Specify an existing capture process name or synchronous capture name. Do not specify an owner.
attribute_name	The name of the attribute to be included in or excluded from LCRs captured by the capture process or synchronous capture. The following names are valid settings: <ul style="list-style-type: none"> <li>▪ row_id The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, or in row LCRs for index-organized tables. The type is VARCHAR2.</li> <li>▪ serial# The serial number of the session that performed the change captured in the LCR. The type is NUMBER.</li> <li>▪ session# The identifier of the session that performed the change captured in the LCR. The type is NUMBER.</li> <li>▪ thread# The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is NUMBER.</li> <li>▪ tx_name The name of the transaction that includes the LCR. The type is VARCHAR2.</li> <li>▪ username The name of the user who performed the change captured in the LCR. The type is VARCHAR2.</li> </ul>
include	If TRUE, then the specified attribute is included in LCRs captured by the capture process or synchronous capture.  If FALSE, then the specified attribute is excluded from LCRs captured by the capture process or synchronous capture.

### Usage Notes

Some information is not captured by a capture process or synchronous capture unless you use this procedure to specify that the information should be captured. If you want

to exclude an extra attribute that is being captured by a capture process or synchronous capture, then specify the attribute and specify `FALSE` for the `include` parameter.

## PREPARE\_GLOBAL\_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating all the tables in the database at another database and can enable supplemental logging for key columns or all columns in these tables. This procedure prepares the tables in the database for instantiation when a capture process will be used to capture changes to the tables in the database.

This procedure records the lowest SCN of each object in the database for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the database for instantiation.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation and supplemental logging

### Syntax

```
DBMS_CAPTURE_ADM.PREPARE_GLOBAL_INSTANTIATION
    supplemental_logging IN VARCHAR2 DEFAULT 'KEYS');
```

### Parameter

**Table 32–12** *PREPARE\_GLOBAL\_INSTANTIATION Procedure Parameter*

Parameter	Description
supplemental_logging	<p>Either NONE, KEYS, or ALL.</p> <p>If NONE is specified, then this procedure does not enable supplemental logging for any columns in the tables in the database. This procedure does not remove existing supplemental logging specifications for these tables.</p> <p>If KEYS is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the database and for any table added to the database in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally. Specifying KEYS does not enable supplemental logging of bitmap join index columns.</p> <p>If ALL is specified, then this procedure enables supplemental logging for all columns in the tables in the database and for any table added to the database in the future. The columns are logged unconditionally. Supplemental logging is not enabled for columns of the following types: LOB, LONG, LONG RAW, user-defined types, and Oracle-supplied types.</p>

### Usage Notes

Run this procedure at the source database.

If you use a capture process to capture all of the changes to a database, then use this procedure to prepare the tables in the database for instantiation after the capture process has been configured.

## PREPARE\_SCHEMA\_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating all tables in the schema at another database and can enable supplemental logging for key columns or all columns in these tables. This procedure prepares the tables in the schema for instantiation when a capture process will be used to capture changes to the tables in the schema.

This procedure records the lowest SCN of each object in the schema for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object. Running this procedure prepares all current and future objects in the schema for instantiation.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation and supplemental logging

### Syntax

```
DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
  schema_name          IN VARCHAR2,
  supplemental_logging IN VARCHAR2 DEFAULT 'KEYS');
```

### Parameters

**Table 32–13** *PREPARE\_SCHEMA\_INSTANTIATION Procedure Parameters*

Parameter	Description
schema_name	The name of the schema. For example, hr.
supplemental_logging	<p>Either NONE, KEYS, or ALL.</p> <p>If NONE is specified, then this procedure does not enable supplemental logging for any columns in the tables in the schema. This procedure does not remove existing supplemental logging specifications for these tables.</p> <p>If KEYS is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the schema and for any table added to this schema in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally. Specifying KEYS does not enable supplemental logging of bitmap join index columns.</p> <p>If ALL is specified, then this procedure enables supplemental logging for all columns in the tables in the schema and for any table added to this schema in the future. The columns are logged unconditionally. Supplemental logging is not enabled for columns of the following types: LOB, LONG, LONG RAW, user-defined types, and Oracle-supplied types.</p>

### Usage Notes

Run this procedure at the source database. If you use a capture process to capture all of the changes to a schema, then use this procedure to prepare the tables in the schema for instantiation after the capture process has been configured.



## PREPARE\_SYNC\_INSTANTIATION Function

This function performs the synchronization necessary for instantiating one or more tables at another database. This function returns the prepare system change number (SCN) for the table or tables being prepared for instantiation.

This function prepares one or more tables for instantiation when a synchronous capture will be used to capture changes to the tables.

This function records the lowest SCN of each table for instantiation (prepare SCN). SCNs after the lowest SCN for an object can be used for instantiating the object.

This function is overloaded. The `table_names` parameter is `VARCHAR2` data type in one version and `DBMS_UTILITY.UNCL_ARRAY` data type in the other version.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation

### Syntax

```
DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION(
    table_names IN VARCHAR2)
RETURN NUMBER;
```

```
DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION(
    table_names IN DBMS_UTILITY.UNCL_ARRAY)
RETURN NUMBER;
```

### Parameters

**Table 32–14** *PREPARE\_SYNC\_INSTANTIATION Function Parameter*

Parameter	Description
<code>table_names</code>	<p>When the <code>table_names</code> parameter is <code>VARCHAR2</code> data type, a comma-delimited list of the tables to prepare for instantiation. There must be no spaces between entries.</p> <p>When the <code>table_names</code> parameter is <code>DBMS_UTILITY.UNCL_ARRAY</code> data type, specify a PL/SQL associative array of this type that contains the names of the tables to prepare for instantiation. The first table name is at position 1, the second at position 2, and so on. The table does not need to be <code>NULL</code> terminated.</p> <p>In either version of the function, specify the name of each table in the form <code>[schema_name.] table_name</code>. For example, <code>hr.employees</code>. If the schema is not specified, then the current user is the default.</p>

## PREPARE\_TABLE\_INSTANTIATION Procedure

This procedure performs the synchronization necessary for instantiating the table at another database and can enable supplemental logging for key columns or all columns in the table. This procedure prepares the table for instantiation when a capture process will be used to capture changes to the table.

This procedure records the lowest SCN of the table for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about instantiation and supplemental logging

### Syntax

```
DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name          IN  VARCHAR2,
    supplemental_logging IN  VARCHAR2 DEFAULT 'KEYS');
```

### Parameters

**Table 32–15** *PREPARE\_TABLE\_INSTANTIATION Procedure Parameters*

Parameter	Description
table_name	The name of the table specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, <i>hr.employees</i> . If the schema is not specified, then the current user is the default.
supplemental_logging	Either NONE, KEYS, or ALL.  If NONE is specified, then this procedure does not enable supplemental logging for any columns in the table. This procedure does not remove existing supplemental logging specifications for the table.  If KEYS is specified, then this procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the table. The procedure places the key columns for the table in three separate log groups: the primary key columns in an unconditional log group, the unique key columns and bitmap index columns in a conditional log group, and the foreign key columns in a conditional log group. Specifying KEYS does not enable supplemental logging of bitmap join index columns.  If ALL is specified, then this procedure enables supplemental logging for all columns in the table. The procedure places all of the columns for the table in an unconditional log group. Supplemental logging is not enabled for columns of the following types: LOB, LONG, LONG RAW, user-defined types, and Oracle-supplied types.

### Usage Notes

Run this procedure at the source database. If you use a capture process to capture all of the changes to a table, then use this procedure to prepare the table for instantiation after the capture process has been configured.

## SET\_PARAMETER Procedure

This procedure sets a capture process parameter to the specified value.

### Syntax

```
DBMS_CAPTURE_ADM.SET_PARAMETER (
  capture_name IN VARCHAR2,
  parameter    IN VARCHAR2,
  value        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 32–16 SET\_PARAMETER Procedure Parameters**

Parameter	Description
capture_name	The name of the capture process. Do not specify an owner.
parameter	The name of the parameter you are setting. See " <a href="#">Capture Process Parameters</a> " on page 32-39 for a list of these parameters.
value	The value to which the parameter is set. If NULL, then the parameter is set to its default value.

### Capture Process Parameters

The following table lists the parameters for the capture process.

---



---

**Note:** Starting with Oracle Database 11g Release 2 (11.2.0.2), this subprogram includes the following new capture process parameters: `capture_idkey_objects`, `capture_sequence_nextval`, `excludeuserid`, `excludeuser`, `excludetrans`, `getapplops`, `getreplicates`, `ignore_transaction`, `ignore_unsupported_table`, and `max_sga_size`.

---



---

**Table 32–17 Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
capture_idkey_objects	Y or N	N	<p>If Y, then the capture process captures ID key logical change records (LCRs).</p> <p>If N, then the capture process does not capture ID key LCRs.</p> <p>Capture processes do not fully support capturing changes to some data types from the redo log. ID key LCRs enable an XStream configuration to capture these changes and process them with an XStream client application.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter and <i>Oracle Database XStream Guide</i> for more information about ID key LCRs</p>
capture_sequence_nextval	Y or N	N	<p>If Y, then the capture process captures sequence LCRs for all of the sequences in the database, except for sequences in Oracle-supplied, administrative schemas such as <code>SYS</code> and <code>SYSTEM</code>. The capture process's rule sets can filter sequence LCRs in the same way that they filter row LCRs and DDL LCRs.</p> <p>If N, then the capture process does not capture sequence LCRs.</p> <p>An apply process or XStream inbound server can use sequence LCRs to ensure that the sequence values at a destination database use the appropriate values. For increasing sequences, the sequence values at the destination are equal to or greater than the sequence values at the source database. For decreasing sequences, the sequence values at the destination are less than or equal to the sequence values at the source database.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter and "SET_PARAMETER Procedure" on page 21-56 for information about the <code>apply_sequence_nextval</code> apply process parameter</p>
disable_on_limit	Y or N	N	<p>If Y, then the capture process is disabled because it reached a value specified by the <code>time_limit</code> parameter or <code>message_limit</code> parameter.</p> <p>If N, then the capture process is restarted immediately after stopping because it reached a limit.</p> <p>When a capture process is restarted, it starts to capture changes at the point where it last stopped. A restarted capture process gets a new session identifier, and the processes associated with the capture process also get new session identifiers. However, the capture process number (<code>CPnn</code>) remains the same.</p>

**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
downstream_real_time_mine	Y or N	Y for local capture processes N for downstream capture processes	<p>If Y, then the capture process is a real-time downstream capture process. After setting this parameter to Y, switch the redo log file at the source database using the SQL statement ALTER SYSTEM ARCHIVE LOG CURRENT to begin real-time downstream capture. If this parameter is set to Y, then redo data from the source database must be sent to the standby redo log at the downstream database. See <i>Oracle Streams Replication Administrator's Guide</i> for information about creating a real-time downstream capture process.</p> <p>If N, then the capture process is an archived-log downstream capture process.</p> <p>An error is raised if an attempt is made to set this parameter for a local capture process.</p>
excludeuserid	Comma-delimited list of user ID values	NULL	<p>Controls whether the capture process captures data manipulation language (DML) changes made by the specified users.</p> <p>Whether the capture process captures these changes depends on the settings for the <code>getapplops</code> and <code>getreplicates</code> parameters.</p> <p>To view the user ID for a user, query the <code>USER_ID</code> column in the <code>ALL_USERS</code> data dictionary view.</p> <p>If NULL, then the capture process ignores this parameter.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter</p>
excludeuser	Comma-delimited list of user names	NULL	<p>Controls whether the capture process captures DML changes made by the specified users.</p> <p>Whether the capture process captures these changes depends on the settings for the <code>getapplops</code> and <code>getreplicates</code> parameters.</p> <p>Specify an exact pattern match for each user name. The pattern match is case sensitive. For example, specify <code>HR</code> for the <code>hr</code> user.</p> <p>If NULL, then the capture process ignores this parameter.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter</p>

**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
excludetrans	Comma-delimited list of transaction names	NULL	<p>Controls whether the capture process captures DML changes in the specified transactions.</p> <p>Whether the capture process captures these changes depends on the settings for the <code>getapplops</code> and <code>getreplicates</code> parameters.</p> <p>If NULL, then the capture process ignores this parameter.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter</p>
getapplops	Y or N	Y	<p>If Y, then the capture process captures DML changes if the original user is not specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is not specified in the <code>excludetrans</code> parameter.</p> <p>If N, then the capture process ignores DML changes if the original user is not specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is not specified in the <code>excludetrans</code> parameter.</p> <p>In either case, the capture process captures a DML change only if it satisfies the capture process's rule sets.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter</p>
getreplicates	Y or N	N	<p>If Y, then the capture process captures DML changes if the original user is specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is specified in the <code>excludetrans</code> parameter.</p> <p>If N, then the capture process ignores DML changes if the original user is specified in the <code>excludeuserid</code> or <code>excludeuser</code> parameters and the transaction name is specified in the <code>excludetrans</code> parameter.</p> <p>In either case, the capture process captures a DML change only if it satisfies the capture process's rule sets.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter</p>

**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
ignore_transaction	A valid transaction ID or NULL	NULL	<p>Instructs the capture process to ignore the specified transaction from the source database, effective immediately.</p> <p>The capture process eliminates all subsequent LCRs for the transaction. If the specified transaction is committed successfully at the source database, the destination database will receive a ROLLBACK statement instead, and any LCRs from the transaction that were enqueued before the ignore transaction request are rolled back at the destination database.</p> <p>If NULL, then the capture process ignores this parameter.</p> <p>Use caution when setting this parameter because ignoring a transaction might lead to data divergence between the source database and destination database.</p> <p>To ignore multiple transactions, specify each transaction in a separate call to the SET_PARAMETER procedure. The DBA_CAPTURE_PARAMETERS view displays a comma-delimited list of all transactions to be ignored. To clear the list of ignored transactions, run the SET_PARAMETER procedure and specify NULL for the ignore_transaction parameter.</p>
ignore_unsupported_table	A fully qualified table name, *, or -	*	<p>Controls the behavior of the capture process when it tries to capture changes to a specified table or to an unsupported table.</p> <p>A capture process tries to capture changes to an unsupported table when its rule sets instruct it to do so. If you do not want the capture process to try to capture changes to unsupported tables, then ensure that the capture process's rule sets exclude unsupported tables.</p> <p>When a table name is specified, the capture process does not capture changes to the specified table. The table name must be entered in the form <i>table_owner. table_name</i>. For example, <i>hr.employees</i>. To specify multiple tables, specify each table in a separate call to the SET_PARAMETER procedure.</p> <p>When * is specified and the capture process tries to capture a change to an unsupported table, the capture process ignores the change and continues to run. The change to the unsupported table is not captured, and the capture process records the unsupported table in the alert log.</p> <p>When - is specified and the capture process tries to capture a change to an unsupported table, the capture process aborts.</p>

**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
max_sga_size	A positive integer	INFINITE	<p>Controls the amount of system global area (SGA) memory allocated specifically to the capture process, in megabytes. The capture process attempts to allocate memory up to this limit. A capture process uses Oracle LogMiner to scan for changes in the redo log.</p> <p>The memory is allocated for the duration of the capture process session and is released when the capture process becomes disabled.</p> <p><b>Note:</b> The sum of system global area (SGA) memory allocated for all components on a database must be less than the value set for the <code>STREAMS_POOL_SIZE</code> initialization parameter.</p> <p>If <code>NULL</code>, then the capture component uses the original default value. A <code>NULL</code> value has the same effect as resetting the parameter to its default value.</p> <p><b>Note:</b> This parameter is intended for XStream. Do not use or attempt to set this parameter in an Oracle Streams replication environment unless XStream capabilities are enabled by the <code>DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS</code> procedure.</p> <p><b>See Also:</b> "Usage Notes" on page 32-47 for more information about this parameter</p>
maximum_scn	A valid SCN or INFINITE	INFINITE	<p>The capture process is disabled before capturing a change record with an SCN greater than or equal to the value specified.</p> <p>If <code>INFINITE</code>, then the capture process runs regardless of the SCN value.</p>
merge_threshold	A negative integer, 0, a positive integer, or INFINITE	60	<p>The amount of time, in seconds, between the message creation time of the original capture process and the message creation time of the cloned capture process.</p> <p>Specifically, if the difference, in seconds, between the <code>CAPTURE_MESSAGE_CREATE_TIME</code> of the cloned capture process and the original capture process is less than or equal to the value specified for this parameter, then automatic merge begins by running the <code>MERGE_STREAMS</code> procedure. If the difference is greater than the value specified by this parameter, then automatic merge does not begin, and the value is recorded in the <code>LAG</code> column of the <code>DBA_STREAMS_SPLIT_MERGE</code> view. The <code>CAPTURE_MESSAGE_CREATE_TIME</code> is recorded in the <code>V\$STREAMS_CAPTURE</code> view.</p> <p>This parameter is relevant only when changes captured by the capture process are applied by two or more apply processes and the <code>split_threshold</code> parameter is set to a value other than <code>INFINITE</code>.</p> <p>If a negative value is specified, then automatic merge is disabled.</p> <p>If 0 (zero) is specified, then there must be no lag between the original capture process and the cloned capture process to begin the merge.</p> <p>If <code>INFINITE</code> is specified, then automatic merging starts immediately.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i></p>



**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
message_limit	A positive integer or INFINITE	INFINITE	<p>The capture process stops after capturing the specified number of messages.</p> <p>If INFINITE, then the capture process continues to run regardless of the number of messages captured.</p>
message_tracking_frequency	0 or a positive integer	2000000	<p>The frequency at which messages captured by the capture process are tracked automatically.</p> <p>For example, if this parameter is set to the default value of 2000000, then every two-millionth message is tracked automatically.</p> <p>The tracking label used for automatic message tracking is <code>capture_process_name:AUTOTRACK</code>, where <code>capture_process_name</code> is the name of the capture process. Only the first 20 bytes of the capture process name are used; the rest is truncated if it exceeds 20 bytes.</p> <p>If 0 (zero), then no messages are tracked automatically.</p> <p>See <i>Oracle Streams Replication Administrator's Guide</i> for more information about message tracking.</p>
parallelism	A positive integer	1	<p>The number of preparer servers that can concurrently mine the redo log for the capture process.</p> <p>A capture process consists of one reader server, one or more preparer servers, and one builder server. The preparer servers concurrently format changes found in the redo log into logical change records (LCRs). Each reader server, preparer server, and builder server is a process, and the number of preparer servers equals the number specified for the <code>parallelism</code> capture process parameter. So, if <code>parallelism</code> is set to 5, then a capture process uses a total of seven processes: one reader server, five preparer servers, and one builder server.</p> <p>Setting the <code>parallelism</code> parameter to a number higher than the number of available parallel execution servers might disable the capture process. Ensure that the <code>PROCESSES</code> initialization parameter is set appropriately when you set the <code>parallelism</code> capture process parameter.</p> <p><b>Note:</b> When you change the value of this parameter, the capture process is stopped and restarted automatically.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about capture process components</p>
skip_autofiltered_table_ddl	Y or N	Y	<p>If Y, then the capture process does not capture data definition language (DDL) changes to tables that are automatically filtered by the capture process.</p> <p>If N, then the capture process can capture DDL changes to tables that are automatically filtered by the capture process if the DDL changes satisfy the capture process rule sets.</p> <p>The <code>AUTO_FILTERED</code> column in the <code>DBA_STREAMS_UNSUPPORTED</code> data dictionary view shows which tables are automatically filtered by capture processes.</p>

**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
split_threshold	0, a positive integer, or INFINITE	1800	<p>The amount of time, in seconds, that a stream is broken before the stream is automatically split from other streams that flow from the capture process. When a stream is split, the capture process, queue, and propagation are cloned.</p> <p>In this case, a stream is a flow of logical change records (LCRs) that flows from a capture process to an apply. A stream is broken when LCRs captured by the capture process cannot reach the apply process. For example, a stream is broken when the relevant propagation or apply process is disabled.</p> <p>This parameter is relevant only when changes captured by the capture process are applied by two or more apply processes.</p> <p>If 0 (zero), then automatic splitting starts immediately.</p> <p>If INFINITE, then automatic splitting is disabled. The stream is not split regardless of the amount of time that it is broken.</p> <p>This parameter is designed to be used with the merge_threshold parameter. You can monitor an automatic split and merge operation by querying the DBA_STREAMS_SPLIT_MERGE view.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i></p>
startup_seconds	0, a positive integer, or INFINITE	0	<p>The maximum number of seconds to wait for another instantiation of the same capture process to finish. If the other instantiation of the same capture process does not finish within this time, then the capture process does not start. This parameter is useful only if you are starting the capture process manually.</p> <p>If INFINITE, then the capture process does not start until another instantiation of the same capture process finishes.</p>
time_limit	A positive integer or INFINITE	INFINITE	<p>The capture process stops as soon as possible after the specified number of seconds since it started.</p> <p>If INFINITE, then the capture process continues to run until it is stopped explicitly.</p>

**Table 32–17 (Cont.) Capture Process Parameters**

Parameter Name	Possible Values	Default	Description
trace_level	0 or a positive integer	0	Set this parameter only under the guidance of Oracle Support Services.
write_alert_log	Y or N	Y	If Y, then the capture process writes a message to the alert log on exit. If N, then the capture process does not write a message to the alert log on exit. The message specifies the reason the capture process stopped.
xout_client_exists	Y or N	Y if the capture process sends LCRs to XStream outbound servers N if the capture process sends LCRs to Oracle Streams apply processes	Y indicates that the capture process sends LCRs to one or more XStream outbound servers. N indicates that the capture process sends LCRs to one or more Oracle Streams apply processes. A single capture process cannot send LCRs to both outbound servers and apply processes. In an XStream configuration where an outbound server runs on a different database than its capture process, set this parameter to Y to enable the capture process to send LCRs to the outbound server. <b>Note:</b> Using XStream requires purchasing a license for the Oracle GoldenGate product. See <i>Oracle Database XStream Guide</i> .

## Usage Notes

The following usage notes apply to the SET\_PARAMETER procedure:

- [Delays Are Possible Before New Parameter Settings Take Effect](#)
- [Parameters Interpreted as Positive Integers](#)
- [Parameters with a System Change Number \(SCN\) Setting](#)
- [Parameters that Require XStream Capabilities](#)

### Delays Are Possible Before New Parameter Settings Take Effect

When you alter a parameter value, a short amount of time might pass before the new value for the parameter takes effect.

### Parameters Interpreted as Positive Integers

For all parameters that are interpreted as positive integers, the maximum possible value is 4,294,967,295. Where applicable, specify INFINITE for larger values.

### Parameters with a System Change Number (SCN) Setting

For parameters that require an SCN setting, any valid SCN value can be specified.

### Parameters that Require XStream Capabilities

A capture process uses the following parameters only when the capture process is sending logical change records (LCRs) to an XStream outbound server or when XStream capabilities are enabled for Oracle Streams components:

- capture\_idkey\_lcrs
- capture\_sequence\_nextval
- excludeuserid
- excludeuser

- `excludetrans`
- `getapplops`
- `getreplicates`
- `max_sga_size`

The `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure enables XStream capabilities for Oracle Streams. When XStream capabilities are not enabled by the `DBMS_XSTREAM_ADM.ENABLE_GG_XSTREAM_FOR_STREAMS` procedure, a capture process raises an error if one of these parameters is set to any value other than its default value.

When XStream capabilities are enabled for Oracle Streams and the `capture_idkey_lcrs` parameter is set to `Y`, a capture process can capture ID key LCRs. ID key LCRs do not contain all of the columns for a row change. Instead, they contain the rowid of the changed row, a group of key columns to identify the row in the table, and the data for the scalar columns of the table that are supported by capture processes. An apply process can apply these changes using the information available the ID key LCRs.

To determine the database objects for which a capture process will capture ID key LCRs, run the following query on the source database:

```
SELECT OWNER, OBJECT_NAME
       FROM DBA_XSTREAM_OUT_SUPPORT_MODE
       WHERE SUPPORT_MODE='ID KEY';
```

---

---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

---

## START\_CAPTURE Procedure

This procedure starts the capture process, which mines redo logs and enqueues the mined redo information into the associated queue.

The start status is persistently recorded. Hence, if the status is `ENABLED`, then the capture process is started upon database instance startup.

The capture process is a background Oracle process and is prefixed by `c`.

The enqueue and dequeue state of `DBMS_AQADM.START_QUEUE` and `DBMS_AQADM.STOP_QUEUE` have no effect on the start status of a capture process.

**See Also:** [Chapter 144, "DBMS\\_STREAMS\\_ADM"](#)

### Syntax

```
DBMS_CAPTURE_ADM.START_CAPTURE (
  capture_name IN VARCHAR2);
```

### Parameters

**Table 32–18** *START\_CAPTURE Procedure Parameter*

Parameter	Description
<code>capture_name</code>	The name of the capture process. Do not specify an owner. The capture process uses LogMiner to capture changes in the redo information. A <code>NULL</code> setting is not allowed.

### Usage Notes

The capture process status is persistently recorded. Hence, if the status is `ENABLED`, then the capture process is started upon database instance startup. A capture process (`cnnn`) is an Oracle background process.

## STOP\_CAPTURE Procedure

This procedure stops the capture process from mining redo logs.

### Syntax

```
DBMS_CAPTURE_ADM.STOP_CAPTURE(
  capture_name IN VARCHAR2,
  force        IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 32–19 STOP\_CAPTURE Procedure Parameters**

Parameter	Description
capture_name	The name of the capture process. A NULL setting is not allowed. Do not specify an owner.
force	If TRUE, then the procedure stops the capture process as soon as possible. If the capture process cannot stop normally, then it aborts.  If FALSE, then the procedure stops the capture process as soon as possible. If the capture process cannot stop normally, then an ORA-26672 error is returned, and the capture process might continue to run.

### Usage Notes

The following usage notes apply to this procedure:

- The capture process status is persistently recorded. Hence, if the status is DISABLED or ABORTED, then the capture process is not started upon database instance startup.
- A capture process is an Oracle background process with a name in the form CPnn, where nn can include letters and numbers.
- The enqueue and dequeue state of DBMS\_AQADM.START\_QUEUE and DBMS\_AQADM.STOP\_QUEUE have no effect on the stop status of a capture process.

---

---

## DBMS\_CDC\_PUBLISH

---

---

**Note:** Oracle Change Data Capture will be de-supported in a future release of Oracle Database and will be replaced with Oracle GoldenGate. Therefore, Oracle strongly recommends that you use Oracle GoldenGate for new applications.

For Oracle Database 11g Release 2 (11.2), Change Data Capture continues to function as in earlier releases. If you are currently using Change Data Capture, then you will be able to continue to do so for the foreseeable future. However, Change Data Capture will not be further enhanced, and will only be supported based on the current, documented functionality.

---

---

The `DBMS_CDC_PUBLISH` package, one of a set of Change Data Capture packages, is used by a publisher to set up an Oracle Change Data Capture system to capture and publish change data from one or more Oracle relational source tables.

Change Data Capture captures and publishes only committed data. Oracle Change Data Capture identifies new data that has been added to, updated in, or removed from relational tables, and publishes the change data in a form that is usable by subscribers.

Typically, a Change Data Capture system has one **publisher** who captures and publishes changes for any number of Oracle relational source tables. The publisher then provides subscribers (applications or individuals) with access to the published data. Subscribers access the published data using the `DBMS_CDC_SUBSCRIBE` package.

**See Also:** *Oracle Database Data Warehousing Guide* for information regarding Oracle Change Data Capture

This chapter contains the following topics:

- [Using DBMS\\_CDC\\_PUBLISH](#)
  - Overview
  - Deprecated Subprograms
  - Security Model
  - Views
- [Summary of DBMS\\_CDC\\_PUBLISH Subprograms](#)

---

## Using DBMS\_CDC\_PUBLISH

This section contains the following topics, which relate to using the DBMS\_CDC\_PUBLISH package:

- [Overview](#)
- [Deprecated Subprograms](#)
- [Security Model](#)
- [Views](#)



## Overview

Through the DBMS\_CDC\_PUBLISH package, the publisher creates and maintains change sources, change sets, and change tables, and eventually drops them when they are no longer useful.

The publisher, typically a database administrator, is concerned primarily with the source of the data and with creating the schema objects that describe the structure of the capture system: change sources, change sets, and change tables.

Most Change Data Capture systems have one publisher and many subscribers. The publisher accomplishes the following main objectives:

1. Determines which source table changes need to be published.
2. Decides whether to capture changes asynchronously or synchronously.
3. Uses the subprograms in the DBMS\_CDC\_PUBLISH package to capture change data from the source tables and make it available by creating and administering the change source, change set, and change table objects.
4. Allows controlled access to subscribers by using the SQL GRANT and REVOKE statements to grant and revoke the SELECT privilege on change tables for users and roles. (This is necessary to allow the subscribers to subscribe to the change data using the DBMS\_CDC\_SUBSCRIBE package.)

**See Also:** [Chapter 34, "DBMS\\_CDC\\_SUBSCRIBE"](#) for information on the package used to subscribe to published change data

## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

---

The following subprograms are deprecated with Oracle Database 11g:

- `DBMS_CDC_PUBLISH.DROP_SUBSCRIPTION` with a subscription handle  
When dropping a subscription, the publisher should now specify the name of the subscription to be dropped, not the subscription handle.
- `DBMS_CDC_PUBLISH.DROP_SUBSCRIBER_VIEW`  
Dropping a subscriber view is now performed automatically by Change Data Capture.

## Security Model

You must have the EXECUTE\_CATALOG\_ROLE role to use the DBMS\_CDC\_PUBLISH package. Additional privileges and roles are required depending on the publishing mode and whether the publisher is on the source or staging database. See the section on Granting Privileges and Roles to the Publisher in *Oracle Database Data Warehousing Guide* for details.

## Views

The `DBMS_CDC_PUBLISH` package uses the views listed in the section on Getting Information About the Change Data Capture Environment in *Oracle Database Data Warehousing Guide*.

## Summary of DBMS\_CDC\_PUBLISH Subprograms

Table 33–1 describes the subprograms in the DBMS\_CDC\_PUBLISH supplied package and the mode or modes with which each can be used. A value of All in the Mode column indicates that the subprogram can be used with synchronous and all modes of asynchronous Change Data Capture, a value of Asynchronous in the Mode column indicates that the subprogram can be used with all modes of asynchronous Change Data Capture (HotLog, Distributed HotLog, and AutoLog).

**Table 33–1 DBMS\_CDC\_PUBLISH Package Subprograms**

Subprogram	Mode	Description
<a href="#">ALTER_AUTOLOG_CHANGE_SOURCE Procedure</a> on page 33-8	Asynchronous AutoLog	Changes one or more properties of an existing AutoLog change source
<a href="#">ALTER_CHANGE_SET Procedure</a> on page 33-10	All	Changes one or more of the properties of an existing change set
<a href="#">ALTER_CHANGE_TABLE Procedure</a> on page 33-13	All	Adds or drops columns for an existing change table, or changes the properties of an existing change table
<a href="#">ALTER_HOTLOG_CHANGE_SOURCE Procedure</a> on page 33-16	Asynchronous Distributed HotLog	Changes one or more properties of an existing Distributed HotLog change source
<a href="#">CREATE_AUTOLOG_CHANGE_SOURCE Procedure</a> on page 33-18	Asynchronous AutoLog	Creates an AutoLog change source
<a href="#">CREATE_CHANGE_SET Procedure</a> on page 33-20	All	Creates a change set
<a href="#">CREATE_CHANGE_TABLE Procedure</a> on page 33-23	All	Creates a change table in a specified schema
<a href="#">CREATE_HOTLOG_CHANGE_SOURCE Procedure</a> on page 33-27	Asynchronous Distributed HotLog	Creates a Distributed HotLog change source
<a href="#">DROP_CHANGE_SET Procedure</a> on page 33-29	All	Drops an existing change set
<a href="#">DROP_CHANGE_SOURCE Procedure</a> on page 33-30	Asynchronous Autolog and Asynchronous Distributed Hotlog	Drops an existing AutoLog or Distributed HotLog change source
<a href="#">DROP_CHANGE_TABLE Procedure</a> on page 33-31	All	Drops an existing change table
<a href="#">DROP_SUBSCRIPTION Procedure</a> on page 33-32	All	Allows a publisher to drop a subscription that was created by a subscriber
<a href="#">GET_DDLOPER Function</a> on page 33-33	All	Converts a binary integer into a user friendly string that describes the DDL operation that actually took place
<a href="#">PURGE Procedure</a> on page 33-34	All	Removes unneeded rows from all change tables in the staging database
<a href="#">PURGE_CHANGE_SET Procedure</a> on page 33-35	All	Removes unneeded rows from all change tables in a specified change set
<a href="#">PURGE_CHANGE_TABLE Procedure</a> on page 33-36	All	Removes unneeded rows from a specified change table

## ALTER\_AUTOLOG\_CHANGE\_SOURCE Procedure

This procedure changes the properties of an existing AutoLog change source.

### Syntax

```
DBMS_CDC_PUBLISH.ALTER_AUTOLOG_CHANGE_SOURCE (
    change_source_name IN VARCHAR2,
    description         IN VARCHAR2 DEFAULT NULL,
    remove_description IN CHAR DEFAULT 'N',
    first_scn          IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 33–2 ALTER\_AUTOLOG\_CHANGE\_SOURCE Procedure Parameters**

Parameter	Description
change_source_name	Name of an existing AutoLog change source. Change source names follow Oracle schema object naming rules.
description	New description of the change source. The description must be specified using 255 or fewer characters.
remove_description	A value of 'Y' or 'N'. If the value is 'Y', then the current description is changed to NULL. If the value is 'N', then the current description is unchanged. Do not specify the description parameter with this parameter.
first_scn	New first SCN.

### Exceptions

**Table 33–3 ALTER\_AUTOLOG\_CHANGE\_SOURCE Procedure Exceptions**

Exception	Description
ORA-31401	Specified change source is not an existing change source
ORA-31445	Invalid lock handle while acquiring lock on string
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31497	Invalid value specified for first_scn
ORA-31498	The description and remove_description parameters cannot both be specified
ORA-31499	Null value specified for required parameter
ORA-31501	Specified change source is not an AutoLog change source
ORA-31504	Cannot alter or drop predefined change source
ORA-31507	Specified parameter value longer than maximum length

### Usage Notes

- Properties supplied to this procedure with a NULL value are unchanged.
- This procedure can be used to change more than one property at a time.
- This procedure can be used in making SCN adjustments after determining which redo logs are no longer needed for an asynchronous AutoLog change set.

**See Also:** The section on asynchronous Change Data Capture and redo log files in *Oracle Database Data Warehousing Guide* for information on how the publisher can use the `ALTER_AUTOLOG_CHANGE_SOURCE` procedure in making SCN adjustments after determining which redo logs are no longer needed for an asynchronous AutoLog change set.

## ALTER\_CHANGE\_SET Procedure

This procedure changes the properties of an existing change set that was created with the CREATE\_CHANGE\_SET procedure.

### Syntax

```
DBMS_CDC_PUBLISH.ALTER_CHANGE_SET (
  change_set_name      IN VARCHAR2,
  description           IN VARCHAR2 DEFAULT NULL,
  remove_description   IN CHAR DEFAULT 'N',
  enable_capture       IN CHAR DEFAULT NULL,
  recover_after_error  IN CHAR DEFAULT NULL,
  remove_ddl           IN CHAR DEFAULT NULL,
  stop_on_ddl          IN CHAR DEFAULT NULL);
```

### Parameters

**Table 33–4 ALTER\_CHANGE\_SET Procedure Parameters**

Parameter	Description
change_set_name	Name of an existing change set. Change set names follow the Oracle schema object naming rules.
description	New description of the change set. Specify using 255 or fewer characters.
remove_description	A value of 'Y' or 'N'. If the value is 'Y', then the current description is changed to NULL. If the value is 'N', then the current description is unchanged. Do not specify the description parameter with this parameter.
enable_capture	A value of 'Y' or 'N'. If the value is 'Y', then change data capture is enabled for this change set. If the value is 'N', then change data capture is disabled for this change set. Synchronous change sets are created with change data capture enabled. Asynchronous change sets are created with change data capture disabled.
recover_after_error	A value of 'Y' or 'N'. If the value is 'Y', then Change Data Capture will attempt to recover from earlier capture errors. If the value is 'N', then Change Data Capture will not attempt to recover from earlier capture errors.
remove_ddl	A value of 'Y' or 'N'. If the value is 'Y' and the value of the recover_after_error parameter is 'Y', then any DDL records that may have caused capture errors will be filtered out during recovery. If the value is 'N', then DDL records that may have caused capture errors will not be filtered out during recovery. This parameter has meaning only when the recover_after_error parameter is specified with a value of 'Y'.



**Table 33–4 (Cont.) ALTER\_CHANGE\_SET Procedure Parameters**

Parameter	Description
stop_on_ddl	<p>A value of 'Y' or 'N'.</p> <p>If the value is 'Y', then Change Data Capture stops when a DDL event is detected.</p> <p>If the value is 'N', then Change Data Capture continues when a DDL event is detected.</p> <p>See the Usage Notes for additional information about this parameter.</p>

## Exceptions

**Table 33–5 ALTER\_CHANGE\_SET Procedure Exceptions**

Exception	Description
ORA-31410	Specified change set is not an existing change set
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31445	Invalid lock handle while acquiring lock on string
ORA-31468	Cannot process DDL change record
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31485	Invalid database link
ORA-31498	The <code>description</code> and <code>remove_description</code> parameters cannot both be specified
ORA-31499	Null value specified for required parameter
ORA-31505	Cannot alter or drop predefined change set
ORA-31507	Specified parameter value longer than maximum length
ORA-31508	Invalid parameter value for synchronous change set
ORA-31514	Change set disabled due to capture error

## Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture. However, the predefined synchronous change set, `SYNC_SET`, cannot be altered, and the following parameters cannot be altered for publisher-defined synchronous change sets: `enable_capture`, `recover_after_error`, `remove_ddl`, and `stop_on_ddl`.
- Properties supplied to this procedure with a NULL value are unchanged.
- This procedure can alter more than one parameter at a time.
- Enabling or disabling an asynchronous HotLog or AutoLog change set starts or stops the Oracle Streams capture process and apply process underlying the change set. Enabling or disabling an asynchronous Distributed HotLog change set starts or stops the Oracle Streams apply process underlying the change set.
- The effect of the `stop_on_ddl` parameter is as follows:
  - When the `stop_on_ddl` parameter is set to 'Y', asynchronous Change Data Capture stops if DDL is encountered during change data capture. Some DDL statements can adversely affect capture, such as a statement that drops a source table column that is being captured. The publisher has an opportunity

to analyze and adjust to DDL changes that may adversely affect change tables while capture is stopped, thereby preventing possible errors during capture.

Because these statements do not affect the column data itself, Change Data Capture does not stop capturing change data when the `stop_on_ddl` parameter is set to 'Y' and any of the following statements is encountered:

- \* ANALYZE TABLE
- \* LOCK TABLE
- \* GRANT privileges to access a table
- \* REVOKE privileges to access a table
- \* COMMENT on a table
- \* COMMENT on a column

These statements can be issued on the source database without concern for their impact on Change Data Capture processing.

- When the `stop_on_ddl` parameter is set to 'N', Change Data Capture does not stop if DDL is encountered during change data capture. If a change set does not stop on DDL, but a DDL change occurs that affects change tables, that change can result in a capture error. There are also system conditions that can cause capture errors, such as being out of disk space.

**See Also:** *Oracle Database Data Warehousing Guide* for information on the effects of, and how to recover from, a capture error

Whenever a DDL statement causes processing to stop, a message is written to the alert log indicating for which change set processing has been stopped and the DDL statement that caused it to be stopped. Similarly, whenever DDL statements are ignored by Change Data Capture and processing continues, a message is written to the alert log indicating which DDL statement was ignored.

- The publisher can attempt to recover an asynchronous change set after a capture error by specifying 'Y' for the `recover_after_error` parameter. Capture errors can occur when any of the following is true:
  - The `stop_on_ddl` parameter is set to 'Y' and there is a DDL record in the change data. In this case, to recover from the error, the publisher must also specify 'Y' for the `remove_ddl` parameter.
  - The `stop_on_ddl` parameter is set to 'N' and there is a DDL record that affects capture. For example, if the publisher drops and re-creates a change table, it causes an error the next time that Change Data Capture attempts to add change data to the named change table.
  - A miscellaneous error occurs, such as running out of disk space, or a redo log file error (such as ORA-01688: unable to extend table *string.string* partition *string* by *string* in tablespace *string*).

**See Also:** *Oracle Database Data Warehousing Guide* for more information on how to recover from a capture error.

## ALTER\_CHANGE\_TABLE Procedure

This procedure adds columns to, or drops columns from, or changes the properties of, a change table that was created with the CREATE\_CHANGE\_TABLE procedure.

### Syntax

```
DBMS_CDC_PUBLISH.ALTER_CHANGE_TABLE (
    owner                IN VARCHAR2,
    change_table_name    IN VARCHAR2,
    operation            IN VARCHAR2,
    column_list         IN VARCHAR2,
    rs_id               IN CHAR,
    row_id              IN CHAR,
    user_id             IN CHAR,
    timestamp           IN CHAR,
    object_id           IN CHAR,
    source_colmap       IN CHAR,
    target_colmap       IN CHAR,
    ddl_markers         IN CHAR DEFAULT NULL);
```

### Parameters

**Table 33–6 ALTER\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
owner	The schema that owns the change table.
change_table_name	The change table that is being altered. Change table names follow the Oracle schema object naming rules.
operation	Either the value ADD or DROP to indicate whether to add or drop the user columns specified with the column_list parameter and any control columns specified by other parameters.
column_list	User column names and datatypes for each column of the source table that should be added to, or dropped from, the change table. The list is comma-delimited.

**Table 33–6 (Cont.) ALTER\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
rs_id	Each listed parameter specifies a particular control column, as follows:
row_id	
user_id	<ul style="list-style-type: none"> <li>▪ The <code>rs_id</code> parameter specifies the <code>RSID\$</code> control column.</li> </ul>
timestamp	<ul style="list-style-type: none"> <li>▪ The <code>row_id</code> parameter specifies the <code>ROW_ID\$</code> control column.</li> </ul>
object_id	<ul style="list-style-type: none"> <li>▪ The <code>user_id</code> parameter specifies the <code>USERNAME\$</code> control column.</li> </ul>
source_colmap	<ul style="list-style-type: none"> <li>▪ The <code>timestamp</code> parameter specifies the <code>TIMESTAMP\$</code> control column.</li> </ul>
target_colmap	<ul style="list-style-type: none"> <li>▪ The <code>object_id</code> parameter specifies the <code>SYS_NC_OID\$</code> control column.</li> </ul>
ddl_markers	<ul style="list-style-type: none"> <li>▪ The <code>source_colmap</code> parameter specifies the <code>SOURCE_COLMAP\$</code> control column.</li> <li>▪ The <code>target_colmap</code> parameter specifies the <code>TARGET_COLMAP\$</code> control column.</li> <li>▪ The <code>ddl_markers</code> parameter tracks all DDL operations on the source table and stores information about those operations in the change table. These are the three additional control columns you get when <code>ddl_markers</code> is enabled. There are three values: <code>DDLOPER\$</code> is a bit vector that indicates what kind of DDL operation happened. (Use the procedure <code>DBMS_CDC_PUBLISH.GET_DDLOPER(ddloper\$)</code> to get the name of the DDL operation.) <code>DDLDESC\$</code> is a CLOB containing the actual DDL statement executed. <code>DDLPOBJN\$</code> is not used in this release.</li> </ul> <p>Each parameter must have a value of either 'Y' or 'N', where:</p> <ul style="list-style-type: none"> <li>▪ 'Y': Adds the specified control column to, or drops it from the change table, as indicated by the <code>operation</code> parameter.</li> <li>▪ 'N': Neither adds the specified control column, nor drops it from the change table.</li> </ul>

**See Also:** *Oracle Database Data Warehousing Guide* for a complete description of control columns.

## Exceptions

**Table 33–7 ALTER\_CHANGE\_TABLE Procedure Exceptions**

Exception	Description
ORA-31403	Specified change table already contains the specified column
ORA-31409	One or more values for input parameters are incorrect
ORA-31415	Specified change set does not exist
ORA-31416	Invalid <code>SOURCE_COLMAP</code> value
ORA-31417	Column list contains control column <i>control-column-name</i>
ORA-31421	Change table does not exist
ORA-31422	Specified owner schema does not exist
ORA-31423	Specified change table does not contain the specified column
ORA-31454	Invalid value specified for operation parameter, expecting ADD or DROP

**Table 33–7 (Cont.) ALTER\_CHANGE\_TABLE Procedure Exceptions**

Exception	Description
ORA-31455	Nothing to alter
ORA-31456	Error executing a procedure in the DBMS_CDC_UTILITY package
ORA-31459	System triggers for DBMS_CDC_PUBLISH package are not installed
ORA-31471	Invalid OBJECT_ID value

## Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher cannot add and drop user columns in the same call to the ALTER\_CHANGE\_TABLE procedure; these schema changes require separate calls.
- The publisher must not specify the name of the control columns in the column\_list parameter.
- When altering an asynchronous change table, the publisher must accept the default value or specify 'N' for the source\_colmap and object\_id parameters. In addition, for the asynchronous Distributed HotLog mode, the publisher also must accept the default value or specify 'N' for the row\_id and username parameters when the change source is 9.2 or 10.1.

**See Also:** *Oracle Database Data Warehousing Guide* for information about the impact on subscriptions when a publisher adds a column to a change table.

## ALTER\_HOTLOG\_CHANGE\_SOURCE Procedure

This procedure changes the properties of an existing Distributed HotLog change source.

### Syntax

```
DBMS_CDC_PUBLISH.ALTER_HOTLOG_CHANGE_SOURCE (
    change_source_name IN VARCHAR2,
    description         IN VARCHAR2 DEFAULT NULL,
    remove_description IN CHAR DEFAULT 'N',
    enable_source      IN CHAR DEFAULT NULL);
```

### Parameters

**Table 33–8 ALTER\_HOTLOG\_CHANGE\_SOURCE Procedure Parameters**

Parameter	Description
change_source_name	Name of an existing Distributed HotLog change source. Change source names follow Oracle schema object naming rules.
description	New description of the change source. The description must be specified using 255 or fewer characters.
remove_description	A value of 'Y' or 'N'. If the value is 'Y', then the current description is changed to NULL. If the value is 'N', then the current description is unchanged. Do not specify the description parameter with this parameter.
enable_source	A value of 'Y' or 'N'. If the value is 'Y', then the change source is enabled. If the value is 'N', then the change source is disabled.

### Exceptions

**Table 33–9 ALTER\_HOTLOG\_CHANGE\_SOURCE Procedure Exceptions**

Exception	Description
ORA-31401	Change source is not an existing change source
ORA-31455	Nothing to ALTER
ORA-31480	Staging database and source database cannot be the same
ORA-31481	Change source is not a HotLog change source
ORA-31482	Invalid option for non-distributed HotLog change source
ORA-31484	Source database must be at least 9.2.0.6 or greater
ORA-31485	Invalid database link
ORA-31498	The description and remove_description parameters cannot both be specified
ORA-31499	Null value specified for required parameter
ORA-31504	Cannot alter or drop predefined change source
ORA-31507	Parameter value longer than maximum length
ORA-31532	Cannot enable change source

**Table 33–9 (Cont.) ALTER\_HOTLOG\_CHANGE\_SOURCE Procedure Exceptions**

<b>Exception</b>	<b>Description</b>
ORA-31534	Change Data Capture publisher is missing DBA role

### Usage Notes

- Properties supplied to this procedure with a NULL value are unchanged.
- This procedure can be used to change more than one property at a time.
- Enabling or disabling a Distributed HotLog change source starts or stops the Oracle Streams capture process that underlies the change source.
- This procedure cannot be used to alter the change source for the asynchronous HotLog mode of Change Database Capture. The change source for the asynchronous HotLog mode is the predefined change source, `HOTLOG_SOURCE`, which cannot be altered.

## CREATE\_AUTOLOG\_CHANGE\_SOURCE Procedure

This procedure creates an AutoLog change source. An AutoLog change source is based on a set of redo log files automatically copied by redo transport services to the system on which the staging database resides.

### Syntax

```
DBMS_CDC_PUBLISH.CREATE_AUTOLOG_CHANGE_SOURCE (
    change_source_name IN VARCHAR2,
    description         IN VARCHAR2 DEFAULT NULL,
    source_database    IN VARCHAR2,
    first_scn          IN NUMBER,
    online_log         IN CHAR DEFAULT 'N');
```

### Parameters

**Table 33–10 CREATE\_AUTOLOG\_CHANGE\_SOURCE Procedure Parameters**

Parameter	Description
change_source_name	Name of the change source. Change source names follow the Oracle schema object naming rules.
description	Description of the change source. Specify using 255 or fewer characters.
source_database	Global name of the change source's source database instance.
first_scn	The SCN of the start of a LogMiner dictionary that is in the change source's archived redo log files.
online_log	A value of 'Y' or 'N'. If the value is 'Y', then the change source uses the AutoLog online option to hot-mine the source database online redo log to gather change data. There can only be one change source with <code>online_log='Y'</code> on a given staging database.  If the value is 'N', then the change source uses the AutoLog archive option to get change data from archived redo log files. There can be one or more change sources with <code>online_log='N'</code> on a given staging database.

### Exceptions

**Table 33–11 CREATE\_AUTOLOG\_CHANGE\_SOURCE Procedure Exceptions**

Exception	Description
ORA-31436	Duplicate change source specified
ORA-31497	Invalid value specified for <code>first_scn</code>
ORA-31499	Null value specified for required parameter
ORA-31507	Specified parameter value is longer than the maximum length
ORA-31508	Invalid parameter value for synchronous change set
ORA-31535	Cannot support change source in this configuration

### Usage Notes

- The publisher can use this procedure for asynchronous Change Data Capture only.



- The publisher must take care when specifying a value for the `source_database` parameter. Change Data Capture does not validate this value when creating the change source. The publisher can query the `GLOBAL_NAME` column in the `GLOBAL_NAME` view at the source database for the `source_database` parameter value.
- The publisher must configure redo transport services to automatically copy the log files to the system on which the staging database resides.

**See Also:** The section on performing asynchronous AutoLog publishing in *Oracle Database Data Warehousing Guide* for information on configuring redo transport services to automatically copy the log files to the system on which the staging database resides.

- An AutoLog change source must begin with an archived redo log file that contains a LogMiner dictionary. The `CREATE_AUTOLOG_CHANGE_SOURCE first_scn` parameter indicates the SCN for this dictionary extraction and is the point at which the change source can begin capturing changes. The publisher can determine the value for the `first_scn` parameter using either of the following methods:

- Direct `DBMS_CAPTURE_ADM.BUILD` to return the value when the dictionary is built:

```
SET SERVEROUTPUT ON
VARIABLE FSCN NUMBER;
BEGIN
  :FSCN := 0;
  DBMS_CAPTURE_ADM.BUILD(:FSCN);
  DBMS_OUTPUT.PUT_LINE('The first_scn value is ' || :FSCN);
END;
/
The first_scn value is 207722
```

- Make the following query on the source database. If this query returns multiple distinct values for `first_change#`, then the data dictionary has been extracted more than once and the publisher should choose the `first_change#` value that is the most appropriate to the change source.

```
SELECT DISTINCT FIRST_CHANGE#, NAME
FROM V$ARCHIVED_LOG
WHERE DICTIONARY_BEGIN = 'YES';
```

**See Also:** The section on performing asynchronous AutoLog publishing in *Oracle Database Data Warehousing Guide* for information on archived redo log files and the LogMiner dictionary.

- For the asynchronous mode of Change Data Capture, the amount of change data captured is dependent on the level of supplemental logging enabled at the source database.

**See Also:** *Oracle Database Data Warehousing Guide* for information about supplemental logging.

## CREATE\_CHANGE\_SET Procedure

This procedure allows the publisher to create a change set. For asynchronous HotLog and AutoLog Change Data Capture, the publisher can optionally provide beginning and ending date values at which to begin and end change data capture.

### Syntax

```
DBMS_CDC_PUBLISH.CREATE_CHANGE_SET(
    change_set_name      IN VARCHAR2,
    description          IN VARCHAR2 DEFAULT NULL,
    change_source_name  IN VARCHAR2,
    stop_on_ddl         IN CHAR DEFAULT 'N',
    begin_date          IN DATE DEFAULT NULL,
    end_date            IN DATE DEFAULT NULL);
```

### Parameters

**Table 33–12 CREATE\_CHANGE\_SET Procedure Parameters**

Parameter	Description
change_set_name	Name of the change set. Change set names follow the Oracle schema object naming rules.
description	Description of the change set. Specify using 255 or fewer characters.
change_source_name	Name of the existing change source to contain this change set.
stop_on_ddl	A value of 'Y' or 'N'. If the value is 'Y', then Change Data Capture stops when a DDL event is detected. If the value is 'N', then Change Data Capture continues when a DDL event is detected. See the Usage Notes for additional information about this parameter.
begin_date	Date on which the publisher wants the change set to begin capturing changes. A value for this parameter is valid for the asynchronous HotLog and AutoLog modes of Change Data Capture only.
end_date	Date on which the publisher wants the change set to stop capturing changes. A value for this parameter is valid for the asynchronous HotLog and AutoLog modes of Change Data Capture only.

### Exceptions

**Table 33–13 CREATE\_CHANGE\_SET Procedure Exceptions**

Exception	Description
ORA-31401	Specified change source is not an existing change source
ORA-31407	The end_date must be greater than the begin_date
ORA-31408	Invalid value specified for begin_scn or end_scn
ORA-31437	Duplicate change set specified
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31483	Cannot have spaces in the parameter
ORA-31485	Invalid database link

**Table 33–13 (Cont.) CREATE\_CHANGE\_SET Procedure Exceptions**

Exception	Description
ORA-31487	Cannot support begin dates or end dates in this configuration
ORA-31488	Cannot support change set in this configuration
ORA-31499	Null value specified for required parameter
ORA-31503	Invalid date supplied for <code>begin_date</code> or <code>end_date</code>
ORA-31507	Specified parameter value longer than maximum length
ORA-31508	Invalid parameter value for synchronous change set

## Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture. However, the default values for the following parameters are the only supported values for synchronous change sets: `begin_date`, `end_date`, and `stop_on_ddl`. The default values for the following parameters are the only supported values for asynchronous Distributed HotLog change sets: `begin_date` and `end_date`.
- When the change source is Distributed HotLog on a release of Oracle Database earlier than 10.2, Change Data Capture inserts rows into the `CHANGE_PROPAGATION` and `CHANGE_PROPAGATION_SETS` views on the staging database.
- An AutoLog online change source (created with `online_log='Y'`) can only contain one change set.
- The `begin_date` and `end_date` parameters are optional. The publisher can specify neither of them, one of them, or both. The effect of these parameters is as follows:
  - When a `begin_date` is specified, changes from transactions that begin on or after that date are captured.
  - When a `begin_date` is not specified, capture starts with the earliest available change data.
  - When an `end_date` is specified, changes from transactions that are committed on or before that date are captured.
  - When an `end_date` is not specified, Change Data Capture continues indefinitely.
- The effect of the `stop_on_ddl` parameter is as follows:
  - When the `stop_on_ddl` parameter is set to 'Y', asynchronous Change Data Capture stops if DDL is encountered during change data capture. Some DDL statements can adversely affect capture, such as a statement that drops a source table column that is being captured. The publisher has an opportunity to analyze and adjust to DDL changes that may adversely affect change tables while capture is stopped, thereby preventing possible errors during capture.
 

Because these statements do not affect the column data itself, Change Data Capture does not stop capturing change data when the `stop_on_ddl` parameter is set to 'Y' and any of the following statements is encountered:

    - \* `ANALYZE TABLE`
    - \* `LOCK TABLE`
    - \* `GRANT` privileges to access a table

- \* REVOKE privileges to access a table
- \* COMMENT on a table
- \* COMMENT on a column

These statements can be issued on the source database without concern for their impact on Change Data Capture processing.

- When the `stop_on_ddl` parameter is set to 'N', Change Data Capture does not stop if DDL is encountered during change data capture. If a change set does not stop on DDL, but a DDL change occurs that affects capture, that change can result in a capture error.

**See Also:** *Oracle Database Data Warehousing Guide* for information on the effects of, and how to recover from, a capture error.

Whenever a DDL statement causes processing to stop, a message is written to the alert log indicating for which change set processing has been terminated and the DDL statement that caused it to be terminated. Similarly, whenever DDL statements are ignored by Change Data Capture and processing continues, a message is written to the alert log indicating which DDL statement was ignored.

## CREATE\_CHANGE\_TABLE Procedure

This procedure creates a change table in a specified schema.

---



---

**Note:** Oracle recommends that the publisher be certain that the source table that will be referenced in a `CREATE_CHANGE_TABLE` procedure has been created prior to calling this procedure, particularly if the change set that will be specified in the procedure has the `stop_on_ddl` parameter set to 'Y'.

---



---

### Syntax

```
DBMS_CDC_PUBLISH.CREATE_CHANGE_TABLE(
  owner                IN VARCHAR2,
  change_table_name    IN VARCHAR2,
  change_set_name      IN VARCHAR2,
  source_schema        IN VARCHAR2,
  source_table         IN VARCHAR2,
  column_type_list     IN VARCHAR2,
  capture_values       IN VARCHAR2,
  rs_id                IN CHAR,
  row_id               IN CHAR,
  user_id              IN CHAR,
  timestamp            IN CHAR,
  object_id            IN CHAR,
  source_colmap        IN CHAR,
  target_colmap        IN CHAR,
  options_string       IN VARCHAR2,
  ddl_markers          IN CHAR DEFAULT 'Y');
```

### Parameters

**Table 33–14** CREATE\_CHANGE\_TABLE Procedure Parameters

Parameter	Description
<code>owner</code>	Name of the schema that owns the change table.
<code>change_table_name</code>	Name of the change table that is being created. Change table names follow the Oracle schema object naming rules.
<code>change_set_name</code>	Name of the change set in which this change table resides.
<code>source_schema</code>	The schema where the source table is located.
<code>source_table</code>	The source table from which the change records are captured.
<code>column_type_list</code>	The user columns and datatypes that are being tracked. Specify using a comma-delimited list.
<code>capture_values</code>	One of the following capture values for update operations: <ul style="list-style-type: none"> <li>■ OLD: Captures the original values from the source table.</li> <li>■ NEW: Captures the changed values from the source table.</li> <li>■ BOTH: Captures the original and changed values from the source table.</li> </ul>

**Table 33–14 (Cont.) CREATE\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description	
rs_id	<p>Each listed parameter specifies a particular control column as follows:</p> <ul style="list-style-type: none"> <li>■ The rs_id parameter specifies the RSID\$ control column.</li> <li>■ The row_id parameter specifies the ROW_ID\$ control column.</li> <li>■ The user_id parameter specifies the USERNAME\$ control column.</li> <li>■ The timestamp parameter specifies the TIMESTAMP\$ control column.</li> <li>■ The object_id parameter specifies the SYS_NC_OID\$ control column.</li> <li>■ The source_colmap parameter specifies the SOURCE_COLMAP\$ control column.</li> <li>■ The target_colmap parameter specifies the TARGET_COLMAP\$ control column.</li> <li>■ The ddl_markers parameter tracks all DDL operations on the source table and stores information about those operations in the change table. There are three values: DDLOPER\$ is a bit vector that indicates what kind of DDL operation happened. (Use the procedure DBMS_CDC_PUBLISH.GET_DDLOPER(ddloper\$) to get the name of the DDL operation.) DDLDESC\$ is a CLOB containing the actual DDL statement executed. DDLPDOBJN\$ is not used in this release.</li> </ul> <p>Each parameter can have a value of 'Y' or 'N', where:</p> <ul style="list-style-type: none"> <li>■ 'Y': Adds the specified control column to the change table.</li> <li>■ 'N': Does not add the specified control column to the change table.</li> </ul>	
row_id		
user_id		
timestamp		
object_id		
source_colmap		
target_colmap		
ddl_markers		
options_string		The syntactically correct options to be passed to a CREATE TABLE DDL statement. The options string is appended to the generated CREATE TABLE DDL statement after the closing parenthesis that defines the columns of the table. See the Usage Notes for more information.

**See Also:** *Oracle Database Data Warehousing Guide* for a complete description of control columns

## Exceptions

**Table 33–15 CREATE\_CHANGE\_TABLE Procedure Exceptions**

Exception	Description
ORA-31402	Unrecognized parameter specified
ORA-31409	One or more values for input parameters are incorrect
ORA-31415	Specified change set does not exist
ORA-31416	Invalid SOURCE_COLMAP value
ORA-31417	Column list contains control column <i>control-column-name</i>
ORA-31418	Specified source schema does not exist
ORA-31419	Specified source table does not exist
ORA-31420	Unable to submit the purge job

**Table 33–15 (Cont.) CREATE\_CHANGE\_TABLE Procedure Exceptions**

Exception	Description
ORA-31421	Change table does not exist
ORA-31422	Owner schema does not exist
ORA-31438	Duplicate change table
ORA-31447	Cannot create change tables in the SYS schema
ORA-31450	Invalid value for <code>change_table_name</code>
ORA-31451	Invalid value for <code>capture_values</code> , expecting: OLD, NEW, or BOTH
ORA-31452	Invalid value for parameter, expecting: Y or N
ORA-31459	System triggers for DBMS_CDC_PUBLISH package are not installed
ORA-31467	No column found in the source table
ORA-31471	Invalid OBJECT_ID value

## Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- A change table is a database table that contains the change data resulting from DML statements (INSERT, UPDATE, and DELETE) made to a source table. A given change table can capture changes from only one source table.
- A change table is a database table that contains two types of columns:
  - User columns, which are copies of actual columns of source tables that reside in the change table.
  - Control columns, which maintain special metadata for each change row in the change table. Information such as the DML operation performed, the capture time (time stamp), and changed column vectors are examples of control columns. The publisher must not specify the name of the control columns in the user column list.
- If there are multiple publishers on the staging database for the Distributed HotLog mode of Change Data capture, and one publisher defines a change table in another publisher's Distributed HotLog change set, then Change Data Capture uses the database link established by the publisher who created the change set to access the source database. Therefore, the database link to the source database established by the publisher who created the change set must be intact for the change table to be successfully created. If the change set publisher's database link is not present when creating a change table, an error is returned indicating that the connection description for the remote database was not found.
- The publisher must not attempt to control a change table's partitioning properties. Change Data Capture automatically manages the change table partitioning as part of its change table management.
- When creating a change table for any mode of asynchronous Change Data Capture, the publisher must accept the default value or specify 'N' for the `source_colmap` and `object_id` parameters. In addition, for the asynchronous Distributed HotLog mode of Change Data Capture, the publisher also must accept the default value or specify 'N' for the `row_id` and `username` parameters when the change source is 9.2 or 10.1.

- When the publisher specifies the `rs_id` parameter, the `RSID$` column is added to the change table. The `RSID$` column value reflects an operation's capture order within a transaction, but not across transactions. The publisher cannot use the `RSID$` column value by itself to order committed operations across transactions; it must be used in conjunction with the `CSCN$` column value.
- The publisher can control a change table's physical properties, tablespace properties, and so on, by specifying the `options_string` parameter. With the `options_string` parameter, the publisher can set any option that is valid for the `CREATE TABLE DDL` statement (except for partitioning properties).

---

---

**Note:** How the publisher defines the `options_string` parameter can have an effect on the performance and operations in a Change Data Capture system. For example, if the publisher places several constraints in the options column, it can have a noticeable effect on performance. Also, if the publisher uses `NOT NULL` constraints and a particular column is not changed in an incoming change row, then the constraint can cause the `INSERT` operation to fail and the transaction that contains the `INSERT` operation to be rolled back.

---

---

- Oracle recommends that change tables not be created in system tablespaces. This can be accomplished if the publisher's default tablespace is not the system tablespace or if the publisher specifies a tablespace in the `options_string` parameter. If a tablespace is not specified by the publisher, and the publisher's default table space is the system tablespace, then Change Data Capture creates change tables in the system tablespace.

**See Also:** *Oracle Database Data Warehousing Guide* for more information on, and examples of, creating change tables in tablespaces managed by the publisher.



## CREATE\_HOTLOG\_CHANGE\_SOURCE Procedure

This procedure creates a Distributed HotLog change source on the source database when the publisher runs this procedure from the staging database. A Distributed HotLog change source is based on data in the online redo log files that is automatically transferred to the staging database by Oracle Streams propagation.

### Syntax

```
DBMS_CDC_PUBLISH.CREATE_HOTLOG_CHANGE_SOURCE (
    change_source_name    IN VARCHAR2,
    description           IN VARCHAR2 DEFAULT NULL,
    source_database       IN VARCHAR2);
```

### Parameters

**Table 33–16 CREATE\_HOTLOG\_CHANGE\_SOURCE Procedure Parameters**

Parameters	Description
change_source_name	Name of the Distributed HotLog change source to be created. Each change source name must be unique and must follow the Oracle schema object naming rules.
description	Description of the change source. Specify using 255 or fewer characters.
source_database	The name of the database link defined from the staging database to the source database, where the source database is Oracle9i Database, Database 10g Release 1, Oracle Database 10g Release 2, or Oracle Database 11g Release 1. See <i>Oracle Database Data Warehousing Guide</i> for information on creating database links for the Distributed HotLog mode of Change Data Capture.

### Exceptions

**Table 33–17 CREATE\_HOTLOG\_CHANGE\_SOURCE Procedure Exceptions**

Exception	Description
ORA-31436	Duplicate change source
ORA-31480	Staging database and source database cannot be the same
ORA-31483	Cannot have spaces in the parameter
ORA-31484	Source database must be at least 9.2.0.6 or greater
ORA-31485	Invalid database link
ORA-31499	Null value specified for required parameter
ORA-31507	Parameter value longer than the maximum length
ORA-31534	Change Data Capture publisher is missing DBA role

### Usage Notes

- The publisher can use this procedure for the asynchronous Distributed HotLog mode of Change Data Capture only.

This procedure cannot be used to create a change source for the asynchronous HotLog mode of Change Database Capture. The publisher must use the

predefined change source, `HOTLOG_SOURCE`, for the asynchronous HotLog mode of Change Data Capture.

- A Distributed HotLog change source can contain one or more change sets, but they must all be on the same staging database.
- A staging database publisher cannot create multiple Distributed HotLog change sources with the same name, even when those change sources are on different source databases.
- When the publisher creates a change source on a release of Oracle Database earlier than 10.2, Change Data Capture:
  - Generates names for the Streams capture process, capture queue, and propagation based on the change source name. If a generated name is already in use, an error indicating that the capture process, queue, or propagation cannot be created is returned.
  - Inserts a row into the `CHANGE_SOURCES` view on the staging database where the `SOURCE_TYPE` column of the inserted row indicates that the source Oracle Database release is earlier than 10.2.
- Note that the database link indicated by the `source_database` parameter must exist when creating, altering, or dropping a Distributed HotLog change source and the change sets and change tables it contains. However, this database link is not required for change capture to occur. Once the required Distributed HotLog change sources, change sets and change tables are in place and enabled, this database link can be dropped without interrupting change capture. This database link would need to be recreated to create, alter, or drop Distributed HotLog change sources, change sets and change tables.

## DROP\_CHANGE\_SET Procedure

This procedure drops an existing change set that was created with the CREATE\_CHANGE\_SET procedure.

### Syntax

```
DBMS_CDC_PUBLISH.DROP_CHANGE_SET (
    change_set_name    IN VARCHAR2);
```

### Parameters

**Table 33–18 DROP\_CHANGE\_SET Procedure Parameters**

Parameter	Description
change_set_name	Name of the change set to be dropped. Change set names follow the Oracle schema object naming rules.

### Exceptions

**Table 33–19 DROP\_CHANGE\_SET Procedure Exceptions**

Exception	Description
ORA-31410	Specified change set is not an existing change set
ORA-31411	Specified change set is referenced by a change table
ORA-31485	Invalid database link
ORA-31499	Null value specified for required parameter
ORA-31505	Cannot alter or drop predefined change set
ORA-31507	Specified parameter value is longer than maximum length

### Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The change set to be dropped cannot contain any change tables.
- The predefined synchronous change set, SYNC\_SET, cannot be dropped.

## DROP\_CHANGE\_SOURCE Procedure

This procedure drops an existing AutoLog change source that was created with the CREATE\_AUTOLOG\_CHANGE\_SOURCE procedure or an existing Distributed HotLog change source that was created with the CREATE\_HOTLOG\_CHANGE\_SOURCE procedure.

### Syntax

```
DBMS_CDC_PUBLISH.DROP_CHANGE_SOURCE (  
    change_source_name    IN VARCHAR2);
```

### Parameters

**Table 33–20 DROP\_CHANGE\_SOURCE Procedure Parameters**

Parameter	Description
change_source_name	Name of the change source to be dropped. Change source names follow the Oracle schema object naming rules.

### Exceptions

**Table 33–21 DROP\_CHANGE\_SOURCE Procedure Exceptions**

Exception	Description
ORA-31401	Specified change source is not an existing change source
ORA-31406	Specified change source is referenced by a change set
ORA-31499	Null value specified for required parameter
ORA-31504	Cannot alter or drop predefined change source
ORA-31507	Specified parameter value longer than maximum length

### Usage Notes

- The change source to be dropped cannot contain any change sets.
- The predefined change sources, HOTLOG\_SOURCE and SYNC\_SOURCE, cannot be dropped.

## DROP\_CHANGE\_TABLE Procedure

This procedure drops an existing change table that was created with the CREATE\_CHANGE\_TABLE procedure.

### Syntax

```
DBMS_CDC_PUBLISH.DROP_CHANGE_TABLE (
    owner           IN VARCHAR2,
    change_table_name IN VARCHAR2,
    force_flag      IN CHAR);
```

### Parameters

**Table 33–22 DROP\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
owner	Name of the schema that owns the change table.
change_table_name	Name of the change table to be dropped. Change table names follow the Oracle schema object naming rules.
force_flag	Drops the change table, depending on whether or not there are subscriptions to it, as follows: <ul style="list-style-type: none"> <li>■ 'Y': Drops the change table even if there are subscriptions to it.</li> <li>■ 'N': Drops the change table only if there are no subscriptions to it.</li> </ul>

### Exceptions

**Table 33–23 DROP\_CHANGE\_TABLE Procedure Exceptions**

Exception	Description
ORA-31421	Change table does not exist
ORA-31422	Specified owner schema does not exist
ORA-31424	Change table has active subscriptions
ORA-31441	Table is not a change table

### Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- If the publisher wants to drop a change table while there are active subscriptions to that table, he or she must call the DROP\_CHANGE\_TABLE procedure using the force\_flag => 'Y' parameter. This tells Change Data Capture to override its normal safeguards and allow the change table to be dropped despite active subscriptions. The subscriptions that include the dropped table will no longer be valid, and subscribers will lose access to the change data.

## DROP\_SUBSCRIPTION Procedure

This procedure allows a publisher to drop a subscription that was created by a subscriber with a prior call to the `DBMS_CDC_SUBSCRIBE.CREATE_SUBSCRIPTION` procedure.

### Syntax

```
DBMS_CDC_PUBLISH.DROP_SUBSCRIPTION(
    subscription_name IN VARCHAR2);
```

### Parameters

**Table 33–24 DROP\_SUBSCRIPTION Procedure Parameters**

Parameter	Description
subscription_name	Name of the subscription that was specified by a previous call to the <code>DBMS_CDC_SUBSCRIBE.CREATE_SUBSCRIPTION</code> procedure. Subscription names follow the Oracle schema object naming rules.

### Exceptions

**Table 33–25 DROP\_SUBSCRIPTION Procedure Exceptions**

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31432	Invalid source table

### Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- This procedure works the same way as the `DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION` procedure.
- This procedure provides the publisher with a way to drop subscriptions that have not been dropped by the subscriber. It is possible that a subscription that is no longer needed still exists and is holding change data in a change table indefinitely. The publisher can use this procedure to remove such a subscription so that a purge operation can clean up its change data. Oracle recommends that the publisher attempt to verify that the subscription is not needed prior to dropping it. If that is not possible, the publisher should inform the subscription owner that the subscription has been dropped. Ideally, subscribers drop subscriptions that are no longer needed using the `DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION` procedure and the publisher need not use the `DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION` procedure.

## GET\_DDLOPER Function

This function converts a binary integer into a user friendly string that describes the DDL operation that actually took place.

### Syntax

```
DBMS_CDC_PUBLISH.GET_DDLOPER (ddloper IN BINARY_INTEGER)
RETURN VARCHAR2;
```

### Parameters

**Table 33–26 Function Parameters**

Parameter	Description
ddloper	An integer value representing what DDL operation actually occurred. This value is obtained from the ddloper\$ column in the subscriber's view.

### Example

The following illustrates how to use change markers. First, you execute a DDL statement and then verify that it has been captured.

```
ALTER TABLE cdc_psales DROP PARTITION Dec_06;
```

```
SELECT ddloper$, DECODE(ddloper$, NULL, 'NULL',
    DBMS_CDC_PUBLISH.GET_DDLOPER(ddloper$))
    AS DDL_OPER
FROM cdc_psales_act
WHERE DDLOPER$ IS NOT NULL
ORDER BY cscn$;
```

```
ddloper$          DDL_OPER
-----
512              Drop Partition
1 row selected.
```

```
SELECT ddldesc$
FROM cdc_psales_act
WHERE ddloper$
IS NOT NULL
ORDER BY cscn$;
```

```
DDLDESC$
-----
alter table cdc_psales drop partition Dec_06
1 row selected.
```

### Usage Notes

- If an invalid value for ddloper is given, then 'Invalid value for DDLOPR\$' will be returned by this function.
- This function only works for asynchronous Change Data Capture.

## PURGE Procedure

This procedure monitors change table usage by all subscriptions, determines which rows are no longer needed by any subscriptions, and removes the unneeded rows to prevent change tables from growing indefinitely. When called, this procedure purges all change tables on the staging database.

### Syntax

```
DBMS_CDC_PUBLISH.PURGE;
```

### Exceptions

Only standard Oracle exceptions (for example, a privilege violation) are returned during a purge operation.

### Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher can run this procedure manually or automatically:
  - The publisher can run this procedure manually from the command line to purge data from change tables.
  - The publisher can run this procedure in a script to routinely perform a purge operation and control the growth of change tables.
- Note that the `DBMS_CDC_PUBLISH.PURGE` procedure (used by the publisher and the Change Data Capture default purge job) is distinct from the `DBMS_CDC_SUBSCRIBE.PURGE_WINDOW` procedure (used by subscribers). A call to the `DBMS_CDC_PUBLISH.PURGE` procedure physically removes unneeded rows from change tables. A call to the `DBMS_CDC_SUBSCRIBE.PURGE_WINDOW` procedure, logically removes change rows from a subscription window, but does not physically remove rows from the underlying change tables.



## PURGE\_CHANGE\_SET Procedure

This procedure removes unneeded rows from all change tables in the named change set. This procedure allows a finer granularity purge operation than the basic PURGE procedure.

### Syntax

```
DBMS_CDC_PUBLISH.PURGE_CHANGE_SET (
    change_set_name IN VARCHAR2,
    force           IN CHAR DEFAULT 'Y',
    purge_date     IN DATE DEFAULT NULL);
```

### Parameters

**Table 33–27** *PURGE\_CHANGE\_SET Procedure Parameters*

Parameter	Description
change_set_name	Name of an existing change set. Change set names follow the Oracle schema object naming rules.
force	If 'Y', try to use partition split/drop, but if the required lock cannot be acquired, use a delete statement to purge. If 'N', only use split/drop partition statements to purge. If a lock cannot be acquired, then no data will be purged.
purge_date	All records that have a commit_timestamp of less than or equal to this date will be purged.

### Exceptions

**Table 33–28** *PURGE\_CHANGE\_SET Procedure Exceptions*

Exception	Description
ORA-31410	Change set is not an existing change set

### Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher can run this procedure manually from the command line or in a script to purge unneeded rows from change tables in a specific change set.
- Note that the DBMS\_CDC\_PUBLISH.PURGE\_CHANGE\_SET procedure (used by the publisher) is distinct from the DBMS\_CDC\_SUBSCRIBE.PURGE\_WINDOW procedure (used by subscribers). A call to the DBMS\_CDC\_PUBLISH.PURGE\_CHANGE\_SET procedure physically removes unneeded rows from change tables in the specified change set. A call to the DBMS\_CDC\_SUBSCRIBE.PURGE\_WINDOW procedure, logically removes change rows from a subscription window, but does not physically remove rows from the underlying change tables.

## PURGE\_CHANGE\_TABLE Procedure

This procedure removes unneeded rows from the named change table. This procedure allows a finer granularity purge operation than the basic PURGE procedure or the PURGE\_CHANGE\_SET procedure.

### Syntax

```
DBMS_CDC_PUBLISH.PURGE_CHANGE_TABLE (
  owner          IN VARCHAR2,
  change_table_name IN VARCHAR2,
  force          IN CHAR DEFAULT 'Y',
  purge_date     IN DATE DEFAULT NULL);
```

### Parameters

**Table 33–29** PURGE\_CHANGE\_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the named change table.
change_table_name	Name of an existing change table. Change table names follow the Oracle schema object naming rules.
force	If 'Y', try to use partition split/drop, but if the required lock cannot be acquired, use a delete statement to purge. If 'N', only use split/drop partition statements to purge. If a lock cannot be acquired, then no data will be purged.
purge_date	All records that have a commit_timestamp of less than or equal to this date will be purged.

### Exceptions

**Table 33–30** PURGE\_CHANGE\_TABLE Procedure Exceptions

Exception	Description
ORA-31421	Change table does not exist

### Usage Notes

- The publisher can use this procedure for asynchronous and synchronous Change Data Capture.
- The publisher can run this procedure manually from the command line or in a script to purge unneeded rows from a specified change table.
- Note that the DBMS\_CDC\_PUBLISH.PURGE\_CHANGE\_TABLE procedure (used by the publisher) is distinct from the DBMS\_CDC\_SUBSCRIBE.PURGE\_WINDOW procedure (used by subscribers). A call to the DBMS\_CDC\_PUBLISH.PURGE\_CHANGE\_TABLE procedure physically removes unneeded rows from the specified change table. A call to the DBMS\_CDC\_SUBSCRIBE.PURGE\_WINDOW procedure, logically removes change rows from a subscription window, but does not physically remove rows from the underlying change tables.

---

---

## DBMS\_CDC\_SUBSCRIBE

---

---

**Note:** Oracle Change Data Capture will be de-supported in a future release of Oracle Database and will be replaced with Oracle GoldenGate. Therefore, Oracle strongly recommends that you use Oracle GoldenGate for new applications.

For Oracle Database 11g Release 2 (11.2), Change Data Capture continues to function as in earlier releases. If you are currently using Change Data Capture, then you will be able to continue to do so for the foreseeable future. However, Change Data Capture will not be further enhanced, and will only be supported based on the current, documented functionality.

---

---

The `DBMS_CDC_SUBSCRIBE` package, one of a set of Change Data Capture packages, lets subscribers view and query change data that was captured and published with the `DBMS_CDC_PUBLISH` package.

A Change Data Capture system usually has one publisher and many subscribers. The **subscribers** (applications or individuals), use the Oracle supplied package, `DBMS_CDC_SUBSCRIBE`, to access published data.

**See Also:** *Oracle Database Data Warehousing Guide* for information regarding Oracle Change Data Capture.

This chapter contains the following topics:

- [Using DBMS\\_CDC\\_SUBSCRIBE](#)
  - Overview
  - Deprecated Subprograms
  - Security Model
  - Views
- [Summary of DBMS\\_CDC\\_SUBSCRIBE Subprograms](#)

## Using DBMS\_CDC\_SUBSCRIBE

This section contains the following topics, which relate to using the DBMS\_CDC\_SUBSCRIBE package:

- [Overview](#)
- [Deprecated Subprograms](#)
- [Security Model](#)
- [Views](#)

## Overview

The primary role of the subscriber is to use the change data. Through the DBMS\_CDC\_SUBSCRIBE package, each subscriber registers interest in source tables by subscribing to them.

Once the publisher sets up the system to capture data into change tables (which are viewed as publications by subscribers) and grants subscribers access to the change tables, subscribers can access and query the published change data for any of the source tables of interest. Using the subprograms in the DBMS\_CDC\_SUBSCRIBE package, the subscriber accomplishes the following main objectives:

1. Indicates the change data of interest by creating a subscription and associated subscriber views on published source tables and source columns
2. Activates the subscription to indicate that the subscriber is ready to receive change data
3. Extends the subscription window to receive a new set of change data
4. Uses SQL `SELECT` statements to retrieve change data from the subscriber views
5. Purges the subscription window when finished processing a block of changes
6. Drops the subscription when finished with the subscription

Figure 34–1 provides a graphical flowchart of the order in which subscribers most typically use the subprograms in the DBMS\_CDC\_SUBSCRIBE package (which are listed in Table 34–1). A subscriber would typically create a subscription, subscribe to one or more source tables and columns, activate the subscription, extend the subscription window, query the subscriber views, purge the subscription window, and then either extend the subscription window again or drop the subscription.

---

---

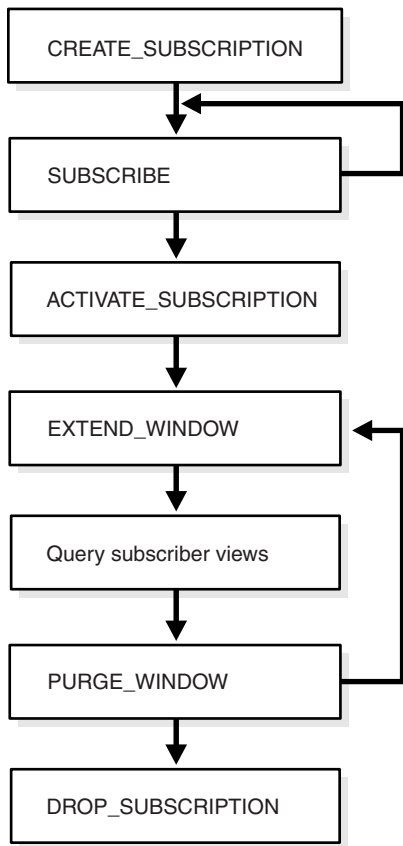
**Note:** If a subscriber uses the `PURGE_WINDOW` procedure immediately after using an `EXTEND_WINDOW` procedure, then change data may be lost without ever being processed.

---

---

**See Also:** [Chapter 34, "DBMS\\_CDC\\_SUBSCRIBE"](#) for information on the package for publishing change data.

**Figure 34-1 Subscription Flow**



This is a text description of arpls011.gif. This figure provides a graphical flowchart. The flowchart shows the following sequence:

1. CREATE\_SUBSCRIPTION
2. SUBSCRIBE
3. Either repeat step 2 or continue to step 4
4. ACTIVATE\_SUBSCRIPTION
5. EXTEND\_WINDOW
6. Query subscriber views
7. PURGE\_WINDOW
8. Either return to step 5 or continue to step 9
9. DROP\_SUBSCRIPTION

\*\*\*\*\*

## Deprecated Subprograms

The following subprograms are deprecated with Oracle Database 11g:

- `DROP_SUBSCRIBER_VIEW`

Subscribers no longer need to drop subscriber views. This work is now done automatically by Change Data Capture.

- `GET_SUBSCRIPTION_HANDLE`

Subscribers no longer explicitly specify subscription handles. Subscribers should use the `CREATE_SUBSCRIPTION` procedure instead to specify a subscription name.

- `PREPARE_SUBSCRIBER_VIEW`

Subscribers no longer need to prepare subscriber views. This work is now done automatically by Change Data Capture.

## Security Model

Change Data Capture grants EXECUTE privileges to PUBLIC on the DBMS\_CDC\_  
SUBSCRIBE package.



## Views

The DBMS\_CDC\_SUBSCRIBE package uses the views listed in the section on Getting Information About the Change Data Capture Environment in *Oracle Database Data Warehousing Guide*.

## Summary of DBMS\_CDC\_SUBSCRIBE Subprograms

**Table 34–1 DBMS\_CDC\_SUBSCRIBE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ACTIVATE_SUBSCRIPTION Procedure</a> on page 34-9	Indicates that a subscription is ready to start accessing change data
<a href="#">CREATE_SUBSCRIPTION Procedure</a> on page 34-10	Creates a subscription and associates it with one change set
<a href="#">DROP_SUBSCRIPTION Procedure</a> on page 34-12	Drops a subscription that was created with a prior call to the <code>CREATE_SUBSCRIPTION</code> procedure
<a href="#">EXTEND_WINDOW Procedure</a> on page 34-13	Sets a subscription window high boundary so that new change data can be seen
<a href="#">PURGE_WINDOW Procedure</a> on page 34-15	Sets the low boundary for a subscription window to notify Change Data Capture that the subscriber is finished processing a set of change data
<a href="#">SUBSCRIBE Procedure</a> on page 34-16	Specifies a source table and the source columns for which the subscriber wants to access change data and specifies the subscriber view through which the subscriber sees change data for the source table

## ACTIVATE\_SUBSCRIPTION Procedure

This procedure indicates that a subscription is ready to start accessing change data.

### Syntax

```
DBMS_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION (
    subscription_name IN VARCHAR2);
```

### Parameters

**Table 34–2 ACTIVATE\_SUBSCRIPTION Procedure Parameters**

Parameter	Description
subscription_name	The name of the subscription that was specified for a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

### Exceptions

**Table 34–3 ACTIVATE\_SUBSCRIPTION Procedure Exceptions**

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31426	Cannot modify active subscriptions
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31514	Change set disabled due to capture error

### Usage Notes

- The ACTIVATE\_SUBSCRIPTION procedure indicates that the subscriber is finished subscribing to tables, and the subscription is ready to start accessing change data.
- Once the subscriber activates the subscription:
  - No additional source tables can be added to the subscription.
  - Change Data Capture holds the available data for the source tables and sets the subscription window to empty.
  - The subscriber must use the EXTEND\_WINDOW procedure to see the initial set of change data.
  - The subscription cannot be activated again.
- A subscription cannot be activated if the underlying change set has reached its end\_date parameter value.

## CREATE\_SUBSCRIPTION Procedure

This procedure creates a subscription that is associated with one change set. This procedure replaces the deprecated `GET_SUBSCRIPTION_HANDLE` procedure.

### Syntax

```
DBMS_CDC_SUBSCRIBE.CREATE_SUBSCRIPTION (
    change_set_name      IN  VARCHAR2,
    description          IN  VARCHAR2,
    subscription_name    IN  VARCHAR2);
```

### Parameters

**Table 34-4 CREATE\_SUBSCRIPTION Procedure Parameters**

Parameter	Description
<code>change_set_name</code>	The name of an existing change set to which the subscriber subscribes
<code>description</code>	A description of the subscription (which might include, for example, the purpose for which it is used). The description must be specified using 255 or fewer characters.
<code>subscription_name</code>	A unique name for a subscription that must consist of 30 characters or fewer and cannot have a prefix of <code>CDC\$</code> . Subscription names follow the Oracle schema object naming rules.

### Exceptions

**Table 34-5 CREATE\_SUBSCRIPTION Procedure Exceptions**

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31415	Specified change set does not exist
ORA-31449	Invalid value for <code>change_set_name</code>
ORA-31457	Maximum length of description field exceeded
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31506	Duplicate subscription name specified
ORA-31510	Name uses reserved prefix <code>CDC\$</code>
ORA-31511	Name exceeds maximum length of 30 characters

### Usage Notes

- The `CREATE_SUBSCRIPTION` procedure allows a subscriber to register interest in a change set associated with source tables of interest.
- A subscriber can query the `ALL_PUBLISHED_COLUMNS` view to see all the published source tables for which the subscriber has privileges and the change sets in which the source table columns are published.
- Subscriptions are not shared among subscribers; rather, each subscription name is validated against a given subscriber's login ID.

- Subscriptions cannot be created if the underlying change set has reached its end\_date parameter value.

## DROP\_SUBSCRIPTION Procedure

This procedure drops a subscription.

### Syntax

```
DBMS_CDC_SUBSCRIBE.DROP_SUBSCRIPTION (  
    subscription_name IN VARCHAR2);
```

### Parameters

**Table 34–6** *DROP\_SUBSCRIPTION Procedure Parameters*

Parameter	Description
subscription_name	The name of the subscription that was specified for a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.

### Exceptions

**Table 34–7** *DROP\_SUBSCRIPTION Procedure Exceptions*

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist

### Usage Notes

Subscribers should be diligent about dropping subscriptions that are no longer needed so that change data will not be held in the change tables unnecessarily.

## EXTEND\_WINDOW Procedure

This procedure sets the subscription window high boundary so that new change data can be seen.

### Syntax

```
DBMS_CDC_SUBSCRIBE.EXTEND_WINDOW (
    subscription_name IN VARCHAR2,
    upper_bound       IN DATE DEFAULT NULL);
```

### Parameters

**Table 34–8** *EXTEND\_WINDOW Procedure Parameters*

Parameter	Description
subscription_name	The unique name of the subscription that was specified by a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.
upper_bound	A date/timestamp to move the upper bound of the subscription window to.

### Exceptions

**Table 34–9** *EXTEND\_WINDOW Procedure Exceptions*

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31429	Subscription has not been activated
ORA-31432	Invalid source table
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31508	Invalid parameter value for synchronous change set
ORA-31509	Publication does not exist
ORA-31514	Change set disabled due to capture error

### Usage Notes

- Until the subscriber calls the EXTEND\_WINDOW procedure to begin receiving change data, the subscription window remains empty.
  - The first time that the subscriber calls the EXTEND\_WINDOW procedure, it establishes the initial boundaries for the subscription window.
  - Subsequent calls to the EXTEND\_WINDOW procedure extend the high boundary of the subscription window so that new change data can be seen.
- Oracle recommends that subscribers not view change tables directly. Instead, subscribers should use the DBMS\_CDC\_SUBSCRIBE package and access data through subscriber views only. Control column values are guaranteed to be consistent only when viewed through subscriber views that have been updated with a call to the EXTEND\_WINDOW procedure.

- When the underlying change set for a subscription has reached its `end_date` parameter value, subsequent calls to the `EXTEND_WINDOW` procedure will not raise the high boundary.
- Subscriptions employing synchronous Change Data Capture are not allowed to extend the window to a specified date. Hence, the `upper_bound` parameter should not be specified.



## PURGE\_WINDOW Procedure

This procedure sets the low boundary of the subscription window so that the subscription no longer sees any change data, effectively making the subscription window empty. The subscriber calls this procedure to notify Change Data Capture that the subscriber is finished processing a block of change data.

### Syntax

```
DBMS_CDC_SUBSCRIBE.PURGE_WINDOW (
    subscription_name  IN VARCHAR2,
    lower_bound        IN DATE DEFAULT NULL);
```

### Parameters

**Table 34–10** *PURGE\_WINDOW Procedure Parameters*

Parameter	Description
subscription_name	The name of the subscription that was specified for a previous call to the CREATE_SUBSCRIPTION procedure. Subscription names follow the Oracle schema object naming rules.
lower_bound	A date/timestamp to move the lower bound of the subscription window to.

### Exceptions

**Table 34–11** *PURGE\_WINDOW Procedure Exceptions*

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31429	Subscription has not been activated
ORA-31432	Invalid source table
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31514	Change set disabled due to capture error

### Usage Notes

- When finished with a set of changes, the subscriber purges the subscription window with the PURGE\_WINDOW procedure. By this action, the subscriber performs the following functions:
  - Informs Change Data Capture that the subscriber is finished with the current set of change data.
  - Enables Change Data Capture to remove change data that is no longer needed by any subscribers.

Change Data Capture manages the change data to ensure that it is available as long as there are subscribers who need it.

- When the underlying change set for a subscription has reached its end\_date parameter value, subsequent calls to the PURGE\_WINDOW procedure will not move the low boundary.

## SUBSCRIBE Procedure

This procedure specifies a source table and the source columns for which the subscriber wants to access change data. In addition, it specifies the subscriber view through which the subscriber sees change data for the source table.

### Syntax

There are two versions of syntax for the `SUBSCRIBE` procedure, as follow:

- Using source schema and source table

When this syntax is used, Change Data Capture will attempt to find a single publication ID that contains the specified `source_table` and `column_list`. If such a publication cannot be found, then Change Data Capture returns an error.

```
DBMS_CDC_SUBSCRIBE.SUBSCRIBE (
    subscription_name    IN VARCHAR2,
    source_schema       IN VARCHAR2,
    source_table        IN VARCHAR2,
    column_list         IN VARCHAR2,
    subscriber_view     IN VARCHAR2);
```

- Using publication IDs

When this syntax is used, Change Data Capture will use the publication ID to identify the change table. If the columns specified in the `column_list` parameter are not in the identified change table, then Change Data Capture returns an error.

```
DBMS_CDC_SUBSCRIBE.SUBSCRIBE (
    subscription_name    IN VARCHAR2,
    publication_id       IN NUMBER,
    column_list         IN VARCHAR2,
    subscriber_view     IN VARCHAR2);
```

### Parameters

**Table 34–12** *SUBSCRIBE Procedure Parameters*

Parameter	Description
<code>subscription_name</code>	The name of a subscription that was specified for, or returned by, a previous call to the <code>CREATE_SUBSCRIPTION</code> procedure. Subscription names follow the Oracle schema object naming rules.
<code>source_schema</code>	The name of the schema where the source table resides
<code>source_table</code>	The name of a published source table
<code>column_list</code>	A comma-delimited list of columns from the published source table or publication
<code>subscriber_view</code>	Unique name for the subscriber view for this source table or publication that must consist of 30 or fewer characters and must not have a prefix of <code>CDC\$</code> . Subscriber view names follow the Oracle schema object naming rules.
<code>publication_id</code>	A valid <code>publication_id</code> , which the subscriber can obtain from the <code>ALL_PUBLISHED_COLUMNS</code> view.

## Exceptions

**Table 34–13** *SUBSCRIBE Procedure Exceptions*

Exception	Description
ORA-31409	One or more values for input parameters are incorrect
ORA-31425	Subscription does not exist
ORA-31426	Cannot modify active subscriptions
ORA-31427	Publication string already subscribed
ORA-31428	No publication contains all the specified columns
ORA-31432	Invalid source table
ORA-31466	No publications found
ORA-31469	Cannot enable Change Data Capture for change set
ORA-31510	Name uses reserved prefix CDC\$
ORA-31511	Name exceeds maximum length of 30 characters

## Usage Notes

- The `SUBSCRIBE` procedure allows a subscriber to subscribe to one or more published source tables and to specific columns in each source table. Each call to the `SUBSCRIBE` procedure can specify only a single source table or publication ID. The subscriber can make multiple calls to the `SUBSCRIBE` procedure to include multiple source tables or publications IDs in a subscription.
- If the columns of interest are all in a single publication, the subscriber can call the `SUBSCRIBE` procedure using the `source_schema` and `source_table` parameters or using the `publication_id` parameter. However, if there are multiple publications on a single source table and these publications share some columns, and if any of the shared columns will be used by a single subscription, then the subscriber should call the `SUBSCRIBE` procedure using the `publication_id` parameter.
- The subscriber can subscribe to any valid publication ID on which the subscriber has privileges to access. The subscriber can find valid publication IDs on which the subscriber has access by querying the `ALL_PUBLISHED_COLUMNS` view.
- A subscriber can query the `ALL_PUBLISHED_COLUMNS` view to see all the published source table columns accessible to the subscriber.
- Subscriptions must be created before a subscriber calls the `SUBSCRIBE` procedure. Change Data Capture does not guarantee that there will be any change data available at the moment the subscription is created.
- Subscribers can subscribe only to published columns from the source table. All of the columns specified in a single call to the `SUBSCRIBE` procedure must come from the same publication. Any control columns associated with the underlying change table are added to the subscription automatically.
- All specified source tables or publications must be in the change set that is associated with the named subscription.
- A single source table can have more than one publication defined on it. A subscriber can subscribe to one or more of these publications. However a subscriber can subscribe to a particular publication only once.

- Each publication in a subscription has its own subscriber view. Subscriber views are used to query the change data encompassed by the subscription's current window. Subscriber views are created in the schema of the subscriber.
- A subscriber cannot subscribe to a publication within a change set that has reached its `end_date` parameter value.

---

---

## DBMS\_COMPARISON

The DBMS\_COMPARISON package provides interfaces to compare and converge database objects at different databases.

This chapter contains the following topics:

- [Using DBMS\\_COMPARISON](#)
  - Overview
  - Security Model
  - Constants
  - Views
  - Operational Notes
- [Data Structures](#)
- [Summary of DBMS\\_COMPARISON Subprograms](#)

**See Also:**

- *Oracle Database 2 Day + Data Replication and Integration Guide* for information about using the basic features of this package
- *Oracle Streams Replication Administrator's Guide* for information about using the advanced features of this package

---

## Using DBMS\_COMPARISON

This section contains topics which relate to using the DBMS\_COMPARISON package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)
- [Operational Notes](#)

## Overview

The DBMS\_COMPARISON package is an Oracle-supplied package that you can use to compare database objects at two databases. This package also enables you converge the database objects so that they are consistent at different databases. Typically, this package is used in environments that share a database object at multiple databases. When copies of the same database object exist at multiple databases, the database object is a **shared database object**. Several data dictionary views contain information about comparisons made with the DBMS\_COMPARISON package.

Shared database objects might be maintained by data replication. For example, materialized views or Oracle Streams components might replicate the database objects and maintain them at multiple databases. A custom application might also maintain shared database objects. When a database object is shared, it can diverge at the databases that share it. You can use this package to identify differences in the shared database objects. After identifying the differences, you can optionally use this package to synchronize the shared database objects.

To compare a database object that is shared at two different databases, complete the following general steps:

1. Run the CREATE\_COMPARE procedure in this package to create a **comparison**. The comparison identifies the database objects to compare and specifies parameters for the comparison.
2. Run the COMPARE function in this package to compare the database object at the two databases and identify differences. This function returns TRUE when no differences are found and FALSE when differences are found. This function also populates data dictionary views with comparison results. Separate comparison results are generated for each execution of the COMPARE function.
3. If you want to examine the comparison results, query the following data dictionary views:
  - DBA\_COMPARISON\_SCAN
  - USER\_COMPARISON\_SCAN
  - DBA\_COMPARISON\_SCAN\_VALUES
  - USER\_COMPARISON\_SCAN\_VALUES
  - DBA\_COMPARISON\_ROW\_DIF
  - USER\_COMPARISON\_ROW\_DIF
4. If there are differences, and you want to synchronize the database objects at the two databases, then run the CONVERGE procedure in this package.

After you create a comparison with the CREATE\_COMPARISON procedure in the DBMS\_COMPARISON package, you can run the comparison at any time using the COMPARE function. Each time you run the COMPARE function, it records comparison results in the appropriate data dictionary views. Comparison results might be modified when subprograms in this package are invoked and the scans in the comparison results are specified. For example, comparison results might be modified when you run the RECHECK function.

The comparison results for a single execution of the COMPARE function can include one or more **scans**. A scan checks for differences in some or all of the rows in a shared database object at a single point in time. You can compare database objects multiple times, and a unique scan ID identifies each scan in the comparison results.

A **bucket** is a range of rows in a database object that is being compared. Buckets improve performance by splitting the database object into ranges and comparing the ranges independently. Every comparison divides the rows being compared into an appropriate number of buckets, and each bucket is compared by a scan.

Each time the `COMPARE` function splits a bucket into smaller buckets, it performs new scans of the smaller buckets. The scan that analyzes a larger bucket is the **parent scan** of each scan that analyzes the smaller buckets into which the larger bucket was split. The **root scan** in the comparison results is the highest level parent scan. The root scan does not have a parent.

You can recheck a scan using the `RECHECK` function, and you can converge a scan using the `CONVERGE` procedure. When you want to recheck or converge all of the rows in comparison results, specify the root scan ID for the comparison results in the appropriate subprogram. When you want to recheck or converge a portion of the rows in comparison results, specify the scan ID of the scan that contains the differences.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about comparisons, including detailed information about scans, buckets, parent scans, and root scans



## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

Each subprogram in the DBMS\_COMPARISON package has a `comparison_name` parameter. The current user must be the owner of the specified comparison to run a subprogram in the DBMS\_COMPARISON package.

To run the COMPARE function, RECHECK function, or CONVERGE procedure, the following users must have SELECT privilege on each copy of the shared database object:

- The comparison owner at the local database
- When a database link is used, the user at the remote database to which the comparison owner connects through a database link

The CONVERGE procedure also requires additional privileges for one of these users at the database where it makes changes to the shared database object. The user must have INSERT, UPDATE, and DELETE privileges on the shared database object at this database.

In addition, when the CONVERGE procedure is run with either the `local_converge_tag` or `remote_converge_tag` parameter set to a non-NULL value, then the following additional requirements must be met:

- If the local table "wins," then the user at the remote database to which the invoker of the CONVERGE procedure connects through a database link must be granted either EXECUTE\_CATALOG\_ROLE or EXECUTE privilege on the DBMS\_STREAMS\_ADM package.
- If the remote table "wins," then the invoker of the CONVERGE procedure at the local database must be granted either EXECUTE\_CATALOG\_ROLE or EXECUTE privilege on the DBMS\_STREAMS\_ADM package.

## Constants

The `DBMS_COMPARISON` package defines several enumerated constants to use specifying parameter values. Enumerated constants must be prefixed with the package name. For example, `DBMS_COMPARISON.CMP_SCAN_MODE_FULL`.

[Table 35-1](#) lists the parameters and enumerated constants.

**Table 35-1** *DBMS\_COMPARISON Parameters with Enumerated Constants*

Parameter	Option	Type	Description
<code>comparison_mode</code>	<ul style="list-style-type: none"> <li>■ <code>CMP_COMPARE_MODE_OBJECT</code></li> </ul>	<code>VARCHAR2 (30)</code>	<code>CMP_COMPARE_MODE_OBJECT</code> is a database object. This constant can be specified as <code>'OBJECT'</code> .
<code>scan_mode</code>	<ul style="list-style-type: none"> <li>■ <code>CMP_SCAN_MODE_FULL</code></li> <li>■ <code>CMP_SCAN_MODE_RANDOM</code></li> <li>■ <code>CMP_SCAN_MODE_CYCLIC</code></li> <li>■ <code>CMP_SCAN_MODE_CUSTOM</code></li> </ul>	<code>VARCHAR2 (30)</code>	<p><code>CMP_SCAN_MODE_FULL</code> indicates that the entire database object is compared. This constant can be specified as <code>'FULL'</code>.</p> <p><code>CMP_SCAN_MODE_RANDOM</code> indicates that a random portion of the database object is compared. This constant can be specified as <code>'RANDOM'</code>.</p> <p><code>CMP_SCAN_MODE_CYCLIC</code> indicates that a portion of the database object is compared when you perform a single comparison. When you compare the database object again, another portion of the database object is compared, starting where the last comparison ended. This constant can be specified as <code>'CYCLIC'</code>.</p> <p><code>CMP_SCAN_MODE_CUSTOM</code> indicates that the user who runs the subprogram specifies the range to compare in the database object. This constant can be specified as <code>'CUSTOM'</code>.</p>
<code>converge_options</code>	<ul style="list-style-type: none"> <li>■ <code>CMP_CONVERGE_LOCAL_WINS</code></li> <li>■ <code>CMP_CONVERGE_REMOTE_WINS</code></li> </ul>	<code>VARCHAR2 (30)</code>	<p><code>CMP_CONVERGE_LOCAL_WINS</code> indicates that the column values at the local database replace the column values at the remote database when these column values are different. This constant can be specified as <code>'LOCAL'</code>.</p> <p><code>CMP_CONVERGE_REMOTE_WINS</code> indicates that the column values at the remote database replace the column values at the local database when these column values are different. This constant can be specified as <code>'REMOTE'</code>.</p>

**Table 35–1 (Cont.) DBMS\_COMPARISON Parameters with Enumerated Constants**

Parameter	Option	Type	Description
null_value	<ul style="list-style-type: none"> <li>■ CMP_NULL_VALUE_DEF</li> </ul>	VARCHAR2(100)	CMP_NULL_VALUE_DEF indicates that ORA\$STREAMS\$NV is substituted for NULL values in database objects during comparison. This constant can be specified as 'ORA\$STREAMS\$NV'.
max_num_buckets	<ul style="list-style-type: none"> <li>■ CMP_MAX_NUM_BUCKETS</li> </ul>	INTEGER	CMP_MAX_NUM_BUCKETS indicates that the maximum number of buckets is 1,000. This constant can be specified as 1000.
min_rows_in_bucket	<ul style="list-style-type: none"> <li>■ CMP_MIN_ROWS_IN_BUCKET</li> </ul>	INTEGER	CMP_MIN_ROWS_IN_BUCKET indicates that the minimum number of rows in a bucket is 10,000. This constant can be specified as 10000.

## Views

The `DBMS_COMPARISON` package uses the following views:

- `DBA_COMPARISON`
- `USER_COMPARISON`
- `DBA_COMPARISON_COLUMNS`
- `USER_COMPARISON_COLUMNS`
- `DBA_COMPARISON_SCAN`
- `USER_COMPARISON_SCAN`
- `DBA_COMPARISON_SCAN_VALUES`
- `USER_COMPARISON_SCAN_VALUES`
- `DBA_COMPARISON_ROW_DIF`
- `USER_COMPARISON_ROW_DIF`

**See Also:** *Oracle Database Reference*

## Operational Notes

This section contains the following operational notes for the DBMS\_COMPARISON package:

- [Oracle Database Release Requirements for the DBMS\\_COMPARISON Package](#)
- [Database Character Set Requirements for the DBMS\\_COMPARISON Package](#)
- [Database Object Requirements for the DBMS\\_COMPARISON Package](#)
- [Index Column Requirements for the DBMS\\_COMPARISON Package](#)
- [Data Type Requirements for the DBMS\\_COMPARISON Package](#)
- [Only Converge Rows That Are Not Being Updated](#)

### Oracle Database Release Requirements for the DBMS\_COMPARISON Package

Meet the following Oracle Database release requirements when running the subprograms in the DBMS\_COMPARISON package:

- The local database that runs the subprograms in the DBMS\_COMPARISON package must be an Oracle Database 11g Release 1 (11.1) database.
- The remote database must be an Oracle Database 10g Release 1 (10.1) or later database. Oracle databases before this release and non-Oracle databases are not supported.

### Database Character Set Requirements for the DBMS\_COMPARISON Package

The database character sets must be the same for the databases that contain the database objects being compared.

**See Also:** *Oracle Database Globalization Support Guide* for information about database character sets

### Database Object Requirements for the DBMS\_COMPARISON Package

The DBMS\_COMPARISON package can compare the following types of database objects:

- Tables
- Single-table views
- Materialized views
- Synonyms for tables, single-table views, and materialized views

Database objects of different types can be compared and converged at different databases. For example, a table at one database and a materialized view at another database can be compared and converged with this package.

To run the subprograms in the DBMS\_COMPARISON package, the specified database objects must have the same shape at each database. Specifically, the database objects must have the same number of columns at each database, and the data types of corresponding columns must match.

If a database object being compared contains columns that do not exist in the other database object, then you can compare the database objects by excluding the extra columns during comparison creation. Use the `column_list` parameter in the CREATE\_COMPARISON procedure to list only the columns that exist in both database objects.

**See Also:** [CREATE\\_COMPARISON Procedure](#) on page 35-20

## Index Column Requirements for the DBMS\_COMPARISON Package

This section discusses number, timestamp, and interval columns. These include the following data types:

- Number columns are of the following data types: NUMBER, FLOAT, BINARY\_FLOAT, and BINARY\_DOUBLE.
- Timestamp columns are of the following data types: TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE
- Interval columns are of the following data types: INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND.

For all scan modes to be supported by the DBMS\_COMPARISON package, the database objects must have one of the following types of indexes:

- A single-column index on a number, timestamp, interval, or DATE data type column
- A composite index that only includes number, timestamp, interval, or DATE data type columns. Each column in the composite index must either have a NOT NULL constraint or be part of the primary key.

For the scan modes CMP\_SCAN\_MODE\_FULL and CMP\_SCAN\_MODE\_CUSTOM to be supported, the database objects must have one of the following types of indexes:

- A single-column index on a number, timestamp, interval, DATE, VARCHAR2, or CHAR data type column
- A composite index that only includes number, timestamp, interval, DATE, VARCHAR2, or CHAR columns. Each column in the composite index must either have a NOT NULL constraint or be part of the primary key.

If the database objects do not have one of these types of indexes, then the DBMS\_COMPARISON package does not support the database objects. For example, if the database objects only have a single index on an NVARCHAR2 column, then the DBMS\_COMPARISON package does not support them. Or, if the database objects have only one index, and it is a composite index that includes a NUMBER column and an NCHAR column, then the DBMS\_COMPARISON package does not support them.

You can specify an index when you create a comparison using the `index_schema_name` and `index_name` parameters in the `CREATE_COMPARISON` procedure. If you specify an index, then make sure the columns in the index meet the requirements of the scan mode used for the comparison.

The index columns in a comparison must uniquely identify every row involved in a comparison. The following constraints satisfy this requirement:

- A primary key constraint
- A unique constraint on one or more non-NULL columns

If these constraints are not present on a table, then use the `index_schema_name` and `index_name` parameters in the `CREATE_COMPARISON` procedure to specify an index whose columns satisfy this requirement.

When a single index value identifies both a local row and a remote row, the two rows must be copies of the same row in the replicated tables. In addition, each pair of copies of the same row must always have the same index value.

The DBMS\_COMPARISON package can use an index only if all of the columns in the index are included in the `column_list` parameter when the comparison is created with the `CREATE_COMPARISON` procedure.

After a comparison is created, you can determine the index column or columns for it by running the following query:

```
SELECT COLUMN_NAME, COLUMN_POSITION FROM DBA_COMPARISON_COLUMNS
WHERE COMPARISON_NAME = 'COMPARE_CUSTOM' AND
      INDEX_COLUMN      = 'Y';
```

If there are multiple index columns, then the index column with 1 for the `COLUMN_POSITION` is the lead index column in the composite index.

**See Also:**

- ["Constants"](#) on page 35-6 for information about scan modes
- [CREATE\\_COMPARISON Procedure](#) on page 35-20 for information about specifying an index for a comparison

### Data Type Requirements for the DBMS\_COMPARISON Package

The DBMS\_COMPARISON package can compare data in columns of the following data types:

- VARCHAR2
- NVARCHAR2
- NUMBER
- FLOAT
- DATE
- BINARY\_FLOAT
- BINARY\_DOUBLE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- RAW
- CHAR
- NCHAR

If a column with data type `TIMESTAMP WITH LOCAL TIME ZONE` is compared, then the two databases must use the same time zone. Also, if a column with data type `NVARCHAR2` or `NCHAR` is compared, then the two databases must use the same national character set.

The DBMS\_COMPARISON package cannot compare data in columns of the following data types:

- LONG
- LONG RAW

- ROWID
- UROWID
- CLOB
- NCLOB
- BLOB
- BFILE
- User-defined types (including object types, REFS, varrays, and nested tables)
- Oracle-supplied types (including any types, XML types, spatial types, and media types)

You can compare database objects that contain unsupported columns by excluding the unsupported columns during comparison creation. Use the `column_list` parameter in the `CREATE_COMPARISON` procedure to list only the supported columns in a shared database object.

**See Also:**

- [CREATE\\_COMPARISON Procedure](#) on page 35-20
- *Oracle Database SQL Language Reference* for more information about data types
- *Oracle Database Globalization Support Guide* for information about national character sets

## Only Converge Rows That Are Not Being Updated

You should only converge rows that are not being updated on either database. For example, if the shared database object is updated by replication components, then only converge rows for which replication changes have been applied and make sure no new changes are in the process of being replicated for these rows. If you compare replicated database objects, then it is typically best to compare them during a time of little or no replication activity to identify persistent differences.

---

---

**Attention:** If a scan identifies that a row is different in the shared database object at two databases, and the row is modified after the scan, then it can result in unexpected data in the row after the `CONVERGE` procedure is run.

---

---

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about the `DBMS_COMPARISON` package in an Oracle Streams replication environment



## Data Structures

---

The DBMS\_COMPARISON package defines a RECORD type.

### RECORD Types

- [COMPARISON\\_TYPE Record Type](#)

## COMPARISON\_TYPE Record Type

Contains information returned by the COMPARE function or CONVERGE procedure in the DBMS\_COMPARISON package.

---

---

**Note:** The COMPARE function only returns a value for the scan\_id field.

---

---

### Syntax

```
TYPE COMPARISON_TYPE IS RECORD(  
    scan_id          NUMBER,  
    loc_rows_merged NUMBER,  
    rmt_rows_merged NUMBER,  
    loc_rows_deleted NUMBER,  
    rmt_rows_deleted NUMBER);
```

### Fields

**Table 35–2** *COMPARISON\_TYPE Attributes*

Field	Description
scan_id	The scan ID of the scan
loc_rows_merged	The number of rows in the local database object updated with information from the database object at the remote site
rmt_rows_merged	The number of rows in the database object updated at the remote site with information from the database object at the local site
loc_rows_deleted	The number of rows deleted from the local database object
rmt_rows_deleted	The number of rows deleted from the remote database object

---

## Summary of DBMS\_COMPARISON Subprograms

**Table 35–3** *DBMS\_COMPARISON Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">COMPARE Function</a> on page 35-16	Performs the specified comparison
<a href="#">CONVERGE Procedure</a> on page 35-18	Executes data manipulation language (DML) changes to synchronize the portion of the database object that was compared in the specified scan
<a href="#">CREATE_COMPARISON Procedure</a> on page 35-20	Creates a comparison
<a href="#">DROP_COMPARISON Procedure</a> on page 35-24	Drops a comparison
<a href="#">PURGE_COMPARISON Procedure</a> on page 35-25	Purges the comparison results, or a subset of the comparison results, for a comparison
<a href="#">RECHECK Function</a> on page 35-26	Rechecks the differences in a specified scan for a comparison

## COMPARE Function

This function performs the specified comparison.

Each time a comparison is performed, it results in at least one new scan, and each scan has a unique scan ID. You can define and name a comparison using the `CREATE_COMPARISON` procedure.

### See Also:

- ["Overview"](#) on page 35-3
- [CREATE\\_COMPARISON Procedure](#) on page 35-20

## Syntax

```
DBMS_COMPARISON.COMPARE(
    comparison_name IN  VARCHAR2,
    scan_info       OUT COMPARISON_TYPE,
    min_value       IN   VARCHAR2  DEFAULT NULL,
    max_value       IN   VARCHAR2  DEFAULT NULL,
    perform_row_dif IN   BOOLEAN    DEFAULT FALSE)
RETURN BOOLEAN;
```

## Parameters

**Table 35–4 COMPARE Function Parameters**

Parameter	Description
<code>comparison_name</code>	The name of the comparison.
<code>scan_info</code>	Information about the compare operation returned in the <code>COMPARISON_TYPE</code> data type. See <a href="#">COMPARISON_TYPE Record Type</a> on page 35-14.
<code>min_value</code>	When the scan mode for the comparison is set to <code>CMP_SCAN_MODE_CUSTOM</code> , specify the minimum index column value for the range of rows that are being compared. To determine the index column for a comparison, query the <code>DBA_COMPARISON_COLUMNS</code> data dictionary view. For a composite index, specify a value for the column with <code>column_position</code> equal to 1 in the <code>DBA_COMPARISON_COLUMNS</code> view. See <a href="#">"Index Column Requirements for the DBMS_COMPARISON Package"</a> on page 35-10.  If the scan mode is set to a value other than <code>CMP_SCAN_MODE_CUSTOM</code> , then this parameter must be set to <code>NULL</code> .  If <code>NULL</code> and the <code>scan_mode</code> parameter is set to <code>CMP_SCAN_MODE_CUSTOM</code> , then an error is raised.  To determine the scan mode for the comparison, query the <code>DBA_COMPARISON</code> data dictionary view.  See <a href="#">Constants</a> on page 35-6 for information about scan modes.

**Table 35–4 (Cont.) COMPARE Function Parameters**

Parameter	Description
max_value	<p>When the scan mode for the comparison is set to <code>CMP_SCAN_MODE_CUSTOM</code>, specify the maximum index column value for the range of rows that are being compared. To determine the index column for a comparison, query the <code>DBA_COMPARISON_COLUMNS</code> data dictionary view. For a composite index, specify a value for the column with <code>column_position</code> equal to 1 in the <code>DBA_COMPARISON_COLUMNS</code> view. See <a href="#">"Index Column Requirements for the DBMS_COMPARISON Package"</a> on page 35-10.</p> <p>If the scan mode is set to a value other than <code>CMP_SCAN_MODE_CUSTOM</code>, then this parameter must be set to <code>NULL</code>.</p> <p>If <code>NULL</code> and the <code>scan_mode</code> parameter is set to <code>CMP_SCAN_MODE_CUSTOM</code>, then an error is raised.</p> <p>To determine the scan mode for the comparison, query the <code>DBA_COMPARISON</code> data dictionary view.</p> <p>See <a href="#">Constants</a> on page 35-6 for information about scan modes.</p>
perform_row_dif	<p>If <code>TRUE</code>, then compares each row individually in the database object being compared after reaching the smallest possible bucket for the comparison.</p> <p>If <code>FALSE</code>, then compares buckets for differences but does not compare each row individually when differences are found in the smallest possible bucket.</p> <p>See <a href="#">"Overview"</a> on page 35-3 for information about buckets.</p>

## Return Values

This function returns `TRUE` when no differences are found in the database objects being compared. This function returns `FALSE` when differences are found in the database objects being compared.

## CONVERGE Procedure

This procedure executes data manipulation language (DML) changes to synchronize the portion of the database objects that was compared in the specified scan.

### Syntax

```
DBMS_COMPARISON.CONVERGE (
  comparison_name    IN   VARCHAR2,
  scan_id            IN   NUMBER,
  scan_info          OUT  COMPARISON_TYPE,
  converge_options   IN   VARCHAR2  DEFAULT CMP_CONVERGE_LOCAL_WINS,
  perform_commit     IN   BOOLEAN    DEFAULT TRUE,
  local_converge_tag IN   RAW        DEFAULT NULL,
  remote_converge_tag IN  RAW        DEFAULT NULL);
```

### Parameters

**Table 35–5 CONVERGE Procedure Parameters**

Parameter	Description
comparison_name	The name of the comparison.
scan_id	The identifier for the scan that contains the differences between the database objects being converged.  See <a href="#">"Overview"</a> on page 35-3 for more information about specifying a scan ID in this parameter.
scan_info	Information about the converge operation returned in the COMPARISON_TYPE data type.  See <a href="#">COMPARISON_TYPE Record Type</a> on page 35-14.
converge_options	Either the CMP_CONVERGE_LOCAL_WINS constant or the CMP_CONVERGE_REMOTE_WINS constant.  See <a href="#">"Constants"</a> on page 35-6 for information about these constants.
perform_commit	If TRUE, then performs a COMMIT periodically while making the DML changes. The CONVERGE procedure might perform more than one COMMIT when this parameter is set to TRUE.  If FALSE, then does not perform a COMMIT after making DML changes.
local_converge_tag	The Oracle Streams tag to set in the session on the local database before performing any changes to converge the data in the database objects being converged.  If non-NULL, then this parameter setting takes precedence over the local_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison.  If NULL, then this parameter is ignored, and the local_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison is used.  See <a href="#">"Security Model"</a> on page 35-5 for information about security requirement related to this parameter, and see the <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags.

**Table 35–5 (Cont.) CONVERGE Procedure Parameters**

Parameter	Description
remote_converge_tag	<p>The Oracle Streams tag to set in the session on the remote database before performing any changes to converge the data in the database objects being converged.</p> <p>If non-NULL, then this parameter setting takes precedence over the remote_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison.</p> <p>If NULL, then this parameter is ignored, and the remote_converge_tag parameter in the CREATE_COMPARISON procedure that created the comparison is used.</p> <p>See "<a href="#">Security Model</a>" on page 35-5 for information about security requirement related to this parameter, and see the <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags.</p>

### Usage Notes

If one of the database objects being converged is a read-only materialized view, then the converge\_options parameter must be set to ensure that the read-only materialized view "wins" in the converge operation. The CONVERGE procedure raises an error if it tries to make changes to a read-only materialized view.

## CREATE\_COMPARISON Procedure

This procedure creates a comparison.

### Syntax

```
DBMS_COMPARISON.CREATE_COMPARISON(
  comparison_name      IN  VARCHAR2,
  schema_name          IN  VARCHAR2,
  object_name          IN  VARCHAR2,
  dblink_name          IN  VARCHAR2,
  index_schema_name    IN  VARCHAR2  DEFAULT NULL,
  index_name           IN  VARCHAR2  DEFAULT NULL,
  remote_schema_name   IN  VARCHAR2  DEFAULT NULL,
  remote_object_name   IN  VARCHAR2  DEFAULT NULL,
  comparison_mode      IN  VARCHAR2  DEFAULT CMP_COMPARE_MODE_OBJECT,
  column_list          IN  VARCHAR2  DEFAULT '*',
  scan_mode            IN  VARCHAR2  DEFAULT CMP_SCAN_MODE_FULL,
  scan_percent         IN  NUMBER     DEFAULT NULL,
  null_value           IN  VARCHAR2  DEFAULT CMP_NULL_VALUE_DEF,
  local_converge_tag   IN  RAW        DEFAULT NULL,
  remote_converge_tag  IN  RAW        DEFAULT NULL,
  max_num_buckets     IN  NUMBER     DEFAULT CMP_MAX_NUM_BUCKETS,
  min_rows_in_bucket  IN  NUMBER     DEFAULT CMP_MIN_ROWS_IN_BUCKET);
```

### Parameters

**Table 35–6 CREATE\_COMPARISON Procedure Parameters**

Parameter	Description
comparison_name	The name of the comparison.
schema_name	The name of the schema that contains the local database object to compare.
object_name	The name of the local database object to compare.
dblink_name	Database link to the remote database. The specified database object in the remote database is compared with the database object in the local database.  If NULL, then the comparison is configured to compare two database objects in the local database. In this case, parameters that specify the remote database object apply to the second database object in the comparison and to operations on the second database object. For example, specify the second database object in this procedure by using the remote_schema_name and remote_object_name parameters.
index_schema_name	The name of the schema that contains the index.  If NULL, then the schema specified in the schema_name parameter is used.
index_name	The name of the index.  If NULL, then the system determines the index columns for the comparison automatically.  If the index_schema_name parameter is non-NULL, then the index_name parameter must also be non-NULL. Otherwise, an error is raised.  <b>See Also:</b> " <a href="#">Usage Notes</a> " on page 35-22 for more information about specifying an index



**Table 35–6 (Cont.) CREATE\_COMPARISON Procedure Parameters**

Parameter	Description
remote_schema_name	<p>The name of the schema that contains the database object at the remote database. Specify a non-NULL value if the schema names are different at the two databases.</p> <p>If NULL, then the schema specified in the schema_name parameter is used.</p>
remote_object_name	<p>The name of the database object at the remote database. Specify a non-NULL value if the database object names are different at the two databases.</p> <p>If NULL, then the database object specified in the object_name parameter is used.</p>
comparison_mode	<p>Specify the default value CMP_COMPARE_MODE_OBJECT. Additional modes might be added in future releases.</p>
column_list	<p>Specify '*' to include all of the columns in the database objects being compared.</p> <p>To compare a subset of columns in the database objects, specify a comma-delimited list of the columns to check. Any columns that are not in the list are ignored during a comparison and convergence.</p> <p>See <a href="#">"Usage Notes"</a> on page 35-22 for information about columns that are required in the column_list parameter.</p>
scan_mode	<p>Either CMP_SCAN_MODE_FULL, CMP_SCAN_MODE_RANDOM, CMP_SCAN_MODE_CYCLIC, or CMP_SCAN_MODE_CUSTOM.</p> <p>If you specify CMP_SCAN_MODE_CUSTOM, then make sure you specify an index using the index_schema_name and index_name parameters. Specifying an index ensures that you can specify the correct min_value and max_value for the lead index column when you run the COMPARE or RECHECK function.</p> <p>See <a href="#">"Constants"</a> on page 35-6 for information about these constants.</p>
scan_percent	<p>The percentage of the database object to scan for comparison when the scan_mode parameter is set to either CMP_SCAN_MODE_RANDOM or CMP_SCAN_MODE_CYCLIC. For these scan_mode settings, a non-NULL value that is greater than 0 (zero) and less than 100 is required.</p> <p>If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_FULL, then the entire database object is scanned for comparison.</p> <p>If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_CUSTOM, then the portion of the database object scanned for comparison is specified when the COMPARE function is run.</p> <p>If non-NULL and the scan_mode parameter is set to either CMP_SCAN_MODE_FULL or CMP_SCAN_MODE_CUSTOM, then the scan_percent parameter is ignored.</p> <p><b>Note:</b> When the scan_percent parameter is non-NULL, and the lead index column for the comparison does not distribute the rows in the database object evenly, the portion of the database object that is compared might be smaller or larger than the specified scan_percent value. See <a href="#">"Index Column Requirements for the DBMS_COMPARISON Package"</a> on page 35-10 for more information about the lead index column.</p>

**Table 35–6 (Cont.) CREATE\_COMPARISON Procedure Parameters**

Parameter	Description
<code>null_value</code>	<p>The value to substitute for each NULL in the database objects being compared. Specify a value or use the <code>CMP_NULL_VALUE_DEF</code> constant.</p> <p>If a column being compared can contain NULLs, then the value specified for this parameter must be different than any non-NULL value in the column. Otherwise, if the value specified for this parameter can appear in the column, some row differences might not be found.</p> <p>See "Constants" on page 35-6 for information about this constant.</p>
<code>local_converge_tag</code>	<p>The Oracle Streams tag to set in the session on the local database before performing any changes to converge the data in the database objects being compared.</p> <p>If the <code>local_converge_tag</code> parameter is non-NULL in the CONVERGE procedure when comparison results for this comparison are converged, then the setting in the CONVERGE procedure takes precedence. See <a href="#">CONVERGE Procedure</a> on page 35-18 for more information.</p> <p>See the <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags.</p>
<code>remote_converge_tag</code>	<p>The Oracle Streams tag to set in the session on the remote database before performing any changes to converge the data in the database objects being compared.</p> <p>If the <code>remote_converge_tag</code> parameter is non-NULL in the CONVERGE procedure when comparison results for this comparison are converged, then the setting in the CONVERGE procedure takes precedence. See <a href="#">CONVERGE Procedure</a> on page 35-18 for more information.</p> <p>See the <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags.</p>
<code>max_num_buckets</code>	<p>Specify the maximum number of buckets to use. Specify a value or use the <code>CMP_MAX_NUM_BUCKETS</code> constant. See "Constants" on page 35-6 for information about this constant.</p> <p>See "Overview" on page 35-3 for information about buckets.</p> <p><b>Note:</b> If an index column for a comparison is a VARCHAR2 or CHAR column, then the number of buckets might exceed the value specified for the <code>max_num_buckets</code> parameter.</p>
<code>min_rows_in_bucket</code>	<p>Specify the minimum number of rows in each bucket. Specify a value or use the <code>CMP_MIN_ROWS_IN_BUCKET</code> constant. See "Constants" on page 35-6 for information about this constant.</p> <p>See "Overview" on page 35-3 for information about buckets.</p>

## Usage Notes

This section contains usage notes for the CREATE\_COMPARISON procedure.

### Usage Notes for the `index_schema_name` and `index_name` Parameters

When you specify an index for a comparison with the `index_schema_name` and `index_name` parameters, the specified index determines the comparison's index columns and their ordering. The order of the columns in the index determines the index column ordering for the comparison. Therefore, the column in column position 1 in the index is the lead column for the comparison.

The index columns and their ordering affect the details of each SQL statement generated and executed for a comparison. For each SQL statement, the optimizer decides whether to use indexes. If the optimizer decides to use indexes, then the optimizer decides which particular indexes to use. An index specified in `column_list` parameter might or might not be used.

The columns in the specified index must meet the requirements described in "[Index Column Requirements for the DBMS\\_COMPARISON Package](#)" on page 35-10. If the index columns do not meet these requirements, then an error is raised.

---

---

**Note:** If you do not specify an index when you create a comparison, then the `CREATE_COMPARISON` procedure selects either the primary key, if it exists, or an existing unique index. The procedure never selects a non-unique index. However, if you specify an index, then the `CREATE_COMPARISON` procedure does not check its uniqueness. Therefore, if you specify a non-unique index, and duplicate index keys exist, then the results might be incorrect when the `CONVERGE` procedure synchronizes data.

---

---

### Usage Notes for the `column_list` Parameter

When the `column_list` parameter is set to a value other than `'*'`, the following columns are required in the `column_list` parameter:

- Any columns that are required to meet the index column requirements for the `DBMS_COMPARISON` package. If the `index_name` parameter is non-NULL, then the columns in the specified index must be in the column list. If the `index_name` parameter is NULL, then see "[Index Column Requirements for the DBMS\\_COMPARISON Package](#)" on page 35-10.
- If you plan to use the `CONVERGE` procedure to make changes to a database object based on the comparison, then any columns in this database object that have a NOT NULL constraint but no default value must be included in the column list. If these columns are not included, then the `CONVERGE` procedure returns an error. See [CONVERGE Procedure](#) on page 35-18.

## DROP\_COMPARISON Procedure

This procedure drops a comparison.

### Syntax

```
DBMS_COMPARISON.DROP_COMPARISON(  
    comparison_name IN VARCHAR2);
```

### Parameters

**Table 35–7** *DROP\_COMPARISON Procedure Parameters*

Parameter	Description
comparison_name	The name of the comparison.

## PURGE\_COMPARISON Procedure

This procedure purges the comparison results, or a subset of the comparison results, for a comparison.

---



---

**Note:** At least one of the following parameters must be set to NULL: `scan_id` or `purge_time`. If both the `scan_id` and `purge_time` parameters are NULL, then this procedure purges all comparison results for the comparison.

---



---

### Syntax

```
DBMS_COMPARISON.PURGE_COMPARISON(
  comparison_name IN VARCHAR2,
  scan_id         IN NUMBER     DEFAULT NULL,
  purge_time      IN TIMESTAMP  DEFAULT NULL);
```

### Parameters

**Table 35–8** *PURGE\_COMPARISON Procedure Parameters*

Parameter	Description
<code>comparison_name</code>	The name of the comparison.
<code>scan_id</code>	The scan ID of the scan for which results are purged. The scan ID must identify a root scan. If the scan ID does not identify a root scan, then an error is raised. When a root scan ID is specified, it is purged, and all direct and indirect child scans of the specified root scan are purged.  If NULL, then no scan ID is considered when purging comparison results for the comparison.  See <a href="#">"Overview"</a> on page 35-3 for information about scans.
<code>purge_time</code>	The date before which results are purged.  If NULL, then no date is considered when purging comparison results for the comparison.

## RECHECK Function

This function rechecks the differences in a specified scan for a comparison.

This function performs one of the following actions:

- If the specified scan completed successfully the last time it ran, then this function checks the previously identified differences in the scan.
- If the specified scan completed partially, then this function continues to check the database object from the point where the previous scan ended.

---

**Note:** This function does not compare the shared database object for differences that were not recorded in the specified comparison scan. To check for those differences, run the `COMPARE` function.

---

### See Also:

- *Oracle Streams Replication Administrator's Guide*
- [COMPARE Function](#) on page 35-16

## Syntax

```
DBMS_COMPARISON.RECHECK(
  comparison_name IN VARCHAR2,
  scan_id         IN NUMBER,
  perform_row_dif IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;
```

## Parameters

**Table 35–9 RECHECK Function Parameters**

Parameter	Description
<code>comparison_name</code>	The name of the comparison.
<code>scan_id</code>	The scan ID of the scan to recheck. See " <a href="#">Overview</a> " on page 35-3 for more information about specifying a scan ID in this parameter.
<code>perform_row_dif</code>	If <code>TRUE</code> , then compares each row individually in the database objects being compared after reaching the smallest possible bucket for the comparison.  If <code>FALSE</code> , then compares buckets for differences but does not compare each row individually when differences are found in the smallest possible bucket.  See " <a href="#">Overview</a> " on page 35-3 for information about buckets.

## Return Values

This function returns `TRUE` when no differences are found in the database objects being compared. This function returns `FALSE` when differences are found in the database objects being compared.

---

---

## DBMS\_COMPRESSION

The DBMS\_COMPRESSION package provides an interface to facilitate choosing the correct compression level for an application.

**See Also:**

- *Oracle Database Administrator's Guide*
- *Oracle Database Concepts*
- *Oracle Database SQL Language Reference*
- *Oracle Database 2 Day + Data Warehousing Guide*
- *Oracle Database Data Warehousing Guide*
- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database Reference*

This chapter contains the following topics:

- [Using DBMS\\_COMPRESSION](#)
  - Overview
  - Security Model
  - Constants
  - Views
- [Summary of DBMS\\_COMPRESSION Subprograms](#)

## Using DBMS\_COMPRESSION

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)



## Overview

The DBMS\_COMPRESSION package gathers compression-related information within a database environment. This includes tools for estimating compressibility of a table for both partitioned and non-partitioned tables, and gathering row-level compression information on previously compressed tables. This gives the user with adequate information to make compression-related decision.

## Security Model

The `DBMS_COMPRESSSION` package is owned by user `SYS`, and must be created by `SYS`. The execution privilege is granted to `PUBLIC`. Subprograms in this package are executed using the privileges of the current user.

## Constants

The DBMS\_COMPRESSION package uses the constants shown in [Table 36-1, "DBMS\\_COMPRESSION Constants - Compression Types"](#):

**Table 36-1 DBMS\_COMPRESSION Constants - Compression Types**

Constant	Type	Value	Description
COMP_NOCOMPRESS	NUMBER	1	No compression
COMP_FOR_OLTP	NUMBER	2	OLTP compression
COMP_FOR_QUERY_HIGH	NUMBER	4	High compression level for query operations
COMP_FOR_QUERY_LOW	NUMBER	8	Low compression level for query operations
COMP_FOR_ARCHIVE_HIGH	NUMBER	16	High compression level for archive operations
COMP_FOR_ARCHIVE_LOW	NUMBER	32	Low compression level for archive operations

---



---

**Note:** Hybrid columnar compression is a feature of certain Oracle storage systems. See *Oracle Database Concepts* for more information.

---



---

## Views

The DBMS\_DST package uses views described in the *Oracle Database Reference*. The twenty catalog views that contain a COMPRESS\_FOR or DEF\_COMPRESS\_FOR will have a list of valid displayed values to be one of the following:

- BASIC
- OLTP
- QUERY LOW
- QUERY HIGH
- ARCHIVE LOW
- ARCHIVE LOW

The affected views are:

- ALL\_ALL\_TABLES
- ALL\_OBJECT\_TABLES
- ALL\_PART\_TABLES
- ALL\_TABLES
- ALL\_TAB\_PARTITIONS
- ALL\_TAB\_SUBPARTITIONS
- DBA\_ALL\_TABLES
- DBA\_OBJECT\_TABLES
- DBA\_PART\_TABLES
- DBA\_TABLES
- DBA\_TAB\_PARTITIONS
- DBA\_TAB\_SUBPARTITIONS
- DBA\_TABLESPACES
- USER\_OBJECT\_TABLES
- USER\_PART\_TABLES
- USER\_TABLES
- USER\_TAB\_PARTITIONS
- USER\_TABLESPACES

## Summary of DBMS\_COMPRESSION Subprograms

**Table 36–2** *DBMS\_COMPRESSION Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_COMPRESSION_RATIO Procedure</a> on page 36-8	Analyzes the compression ratio of a table, and gives information about compressibility of a table
<a href="#">GET_COMPRESSION_TYPE Function</a> on page 36-10	Returns the compression type for a specified row

## GET\_COMPRESSION\_RATIO Procedure

This procedure analyzes the compression ratio of a table, and gives information about compressibility of a table. Various parameters can be provided by the user to selectively analyze different compression types.

The overload takes an index name and an optional partition name as IN arguments, and for every possible type of compression algorithm (currently Auto-Prefix or OLTP), it return the compression ratio as an OUT argument. In addition, it returns a count of the compressed blocks and uncompressed blocks which can be used to determine absolute space consumption of the index (partition) with or without compression.

### Syntax

```
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
    scratchtbsname      IN    VARCHAR2,
    tabowner            IN    VARCHAR2,
    ownname             IN    VARCHAR2,
    partname            IN    VARCHAR2,
    comptype            IN    NUMBER,
    blkcnt_cmp          OUT   PLS_INTEGER,
    blkcnt_uncmp        OUT   PLS_INTEGER,
    row_cmp             OUT   PLS_INTEGER,
    row_uncmp           OUT   PLS_INTEGER,
    cmp_ratio           OUT   NUMBER,
    comptype_str        OUT   VARCHAR2,
    subset_numrows      IN    NUMBER DEFAULT COMP_RATIO_MINROWS);
```

```
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
    scratchtbsname      IN    VARCHAR2,
    indexname           IN    VARCHAR2,
    tabowner            IN    VARCHAR2,
    tabname             IN    VARCHAR2,
    lobname             IN    VARCHAR2,
    partname            IN    VARCHAR2,
    comptype            IN    NUMBER,
    blkcnt_cmp          OUT   PLS_INTEGER,
    blkcnt_uncmp        OUT   PLS_INTEGER,
    row_cmp             OUT   PLS_INTEGER,
    row_uncmp           OUT   PLS_INTEGER,
    cmp_ratio           OUT   NUMBER,
    comptype_str        OUT   VARCHAR2,
    subset_numrows      IN    NUMBER DEFAULT COMP_RATIO_LOB_MAXROWS);
```

```
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
    scratchtbsname      IN    VARCHAR2,
    indexname           IN    VARCHAR2,
    tabowner            IN    VARCHAR2,
    tabname             IN    VARCHAR2,
    partname            IN    VARCHAR2,
    comptype            IN    NUMBER,
    blkcnt_cmp          OUT   PLS_INTEGER,
    blkcnt_uncmp        OUT   PLS_INTEGER,
    row_cmp             OUT   PLS_INTEGER,
    row_uncmp           OUT   PLS_INTEGER,
    cmp_ratio           OUT   NUMBER,
    comptype_str        OUT   VARCHAR2,
    subset_numrows      IN    NUMBER DEFAULT COMP_RATIO_MINROWS);
```

## Parameters

**Table 36–3** *GET\_COMPRESSION\_RATIO Procedure Parameters*

Parameter	Description
scratchtbsname	Temporary scratch tablespace that can be used for analysis
ownname / tabowner	Schema of the table to analyze
tabname	Name of the table to analyze
partname	In case of partitioned tables, the related partition name
comptype	Compression types for which analysis should be performed
blkcnt_cmp	Number of blocks used by compressed sample of the table
blkcnt_uncmp	Number of blocks used by uncompressed sample of the table
row_cmp	Number of rows in a block in compressed sample of the table
row_uncmp	Number of rows in a block in uncompressed sample of the table
cmp_ratio	Compression ratio, blkcnt_uncmp divided by blkcnt_cmp
comptype_str	String describing the compression type
indexname	Name of index
tabowner	Owner of table to analyze
row_cmp	Number of rows in compressed sample of the table
row_uncmp	Number of rows in uncompressed sample of the table
lobname	
subset_numrows	

## Usage Notes

The procedure creates different tables in the scratch tablespace and runs analysis on these objects. It does not modify anything in the user-specified tables.

## GET\_COMPRESSION\_TYPE Function

This function returns the compression type for a specified row. If the row is chained, the function returns the compression type of the head piece only, and does not examine the intermediate or the tail piece since head pieces can be differently compressed.

### Syntax

```
DBMS_COMPRESSION.GET_COMPRESSION_TYPE (  
    ownname    IN    VARCHAR2,  
    tabname    IN    VARCHAR2,  
    row_id     IN    ROWID)  
RETURN NUMBER;
```

### Parameters

**Table 36–4** GET\_COMPRESSION\_TYPE Function Parameters

Parameter	Description
ownname	Schema name of the table
tabname	Name of table
rowid	Rowid of the row

### Return Values

Flag to indicate the compression type (see [DBMS\\_COMPRESSION Constants - Compression Types](#)).



---

---

## DBMS\_CONNECTION\_POOL

The DBMS\_CONNECTION\_POOL package provides an interface to manage Database Resident Connection Pool.

**See Also:** *Oracle Database Concepts* for more information on "Database Resident Connection Pooling"

This chapter contains the following topic:

- [Summary of DBMS\\_CONNECTION\\_POOL Subprograms](#)

## Summary of DBMS\_CONNECTION\_POOL Subprograms

**Table 37-1 DBMS\_CONNECTION\_POOL Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ALTER_PARAM Procedure</a> on page 37-3	Alters a specific configuration parameter as a standalone unit and does not affect other parameters
<a href="#">CONFIGURE_POOL Procedure</a> on page 37-4	Configures the pool with advanced options
<a href="#">START_POOL Procedure</a> on page 37-6	Starts the pool for operations. It is only after this call that the pool could be used by connection clients for creating sessions
<a href="#">STOP_POOL Procedure</a> on page 37-7	Stops the pool and makes it unavailable for the registered connection clients
<a href="#">RESTORE_DEFAULTS Procedure</a> on page 37-8	Restores the pool to default settings

## ALTER\_PARAM Procedure

This procedure alters a specific configuration parameter as a standalone unit and does not affect other parameters.

### Syntax

```
DBMS_CONNECTION_POOL.ALTER_PARAM (
  pool_name      IN  VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL',
  param_name     IN  VARCHAR2,
  param_value    IN  VARCHAR2);
```

### Parameters

**Table 37-2 ALTER\_PARAM Procedure Parameters**

Parameter	Description
pool_name	Pool to be configured. Currently only the default pool name is supported.
param_name	Any parameter name from CONFIGURE_POOL
param_value	Parameter value for param_name.

### Exceptions

**Table 37-3 ALTER\_PARAM Procedure Exceptions**

Exception	Description
ORA-56500	Connection pool not found
ORA-56504	Invalid connection pool configuration parameter name
ORA-56505	Invalid connection pool configuration parameter value
ORA-56507	Connection pool alter configuration failed

### Examples

```
DBMS_CONNECTION_POOL.ALTER_PARAM(
  'SYS_DEFAULT_CONNECTION_POOL', 'MAX_LIFETIME_SESSION', '120');
```

## CONFIGURE\_POOL Procedure

This procedure configures the pool with advanced options.

### Syntax

```
DBMS_CONNECTION_POOL.CONFIGURE_POOL (
  pool_name          IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL',
  minsize            IN NUMBER   DEFAULT 4,
  maxsize            IN NUMBER   DEFAULT 40,
  incrsz            IN NUMBER   DEFAULT 2,
  session_cached_cursors IN NUMBER DEFAULT 20,
  inactivity_timeout IN NUMBER   DEFAULT 300,
  max_think_time     IN NUMBER   DEFAULT 120,
  max_use_session    IN NUMBER   DEFAULT 500000,
  max_lifetime_session IN NUMBER   DEFAULT 86400);
```

### Parameters

**Table 37-4 CONFIGURE\_POOL Procedure Parameters**

Parameter	Description
pool_name	Pool to be configured. Currently only the default pool name is supported.
minsize	Minimum number of pooled servers in the pool
maxsize	Maximum allowed pooled servers in the pool
incrsz	Pool would increment by this number of pooled server when pooled server are unavailable at application request time
session_cached_cursors	Turn on SESSION_CACHED_CURSORS for all connections in the pool. This is an existing init.ora parameter
inactivity_timeout	TTL (Time to live) for an idle session in the pool. This parameter helps to shrink the pool when it is not used to its maximum capacity. If a connection remains in the pool idle for this time, it is killed.
max_think_time	Maximum time of inactivity by the client after getting a session from the pool. If the client does not issue a database call after grabbing a server from the pool, the client will be forced to relinquish control of the pooled server and will get an ORA-xxxxxx error. The freed up server may or may not be returned to the pool.
max_use_session	Maximum number of times a connection can be taken and released to the pool
max_lifetime_session	TTL (Time to live) for a pooled session

### Exceptions

**Table 37-5 CONFIGURE\_POOL Procedure Exceptions**

Exception	Description
ORA-56500	Connection pool not found
ORA-56507	Connection pool alter configuration failed

## Usage Notes

- All expressions of time are in seconds
- All of the parameters should be set based on statistical request patterns.
- `minsize` should be set keeping in mind that it puts a lower bound on server resource consumption. This is to prevent the timeout from dragging the pool too low, because of a brief period of inactivity.
- `maxsize` should be set keeping in mind that it puts an upper bound on concurrency and response-times and also server resource consumption.
- `session_cached_cursors` is typically set to the number of most frequently used statements. It occupies cursor resource on the server
- In doubt, do not set the `increment` and `inactivity_timeout`. The pool will have reasonable defaults.
- `max_use_session` and `max_lifetime_session` allow for software rejuvenation or defensive approaches to potential bugs, leaks, accumulations, and like problems, by getting brand new sessions once in a while.
- The connection pool reserves 5% of the pooled servers for authentication, and at least one pooled server is always reserved for authentication. When setting the `maxsize` parameter, ensure that there are enough pooled servers for both authentication and connections.

## START\_POOL Procedure

This procedure starts the pool for operations. It is only after this call that the pool could be used by connection classes for creating sessions.

### Syntax

```
DBMS_CONNECTION_POOL.START_POOL (  
    pool_name IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL');
```

### Parameters

**Table 37-6** *START\_POOL Procedure Parameters*

Parameter	Description
pool_name	Pool to be started. Currently only the default pool name is supported.

### Exceptions

**Table 37-7** *START\_POOL Procedure Exceptions*

Exception	Description
ORA-56500	Connection pool not found
ORA-56501	Connection pool startup failed

### Usage Notes

If the instance is restarted (shutdown followed by startup), the pool is automatically started.

## STOP\_POOL Procedure

This procedure stops the pool and makes it unavailable for the registered connection classes.

### Syntax

```
DBMS_CONNECTION_POOL.STOP_POOL (
    pool_name IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL');
```

### Parameters

**Table 37–8 STOP\_POOL Procedure Parameters**

Parameter	Description
pool_name	Pool to be stopped. Currently only the default pool name is supported.

### Exceptions

**Table 37–9 STOP\_POOL Procedure Exceptions**

Exception	Description
ORA-56500	Connection pool not found
ORA-56506	Connection pool shutdown failed

### Usage Notes

This stops the pool and takes it offline. This does not destroy the persistent data (such as, the pool name and configuration parameters) associated with the pool.

## RESTORE\_DEFAULTS Procedure

This procedure restores the pool to default settings.

### Syntax

```
DBMS_CONNECTION_POOL.RESTORE_DEFAULTS (  
    pool_name IN VARCHAR2 DEFAULT 'SYS_DEFAULT_CONNECTION_POOL');
```

### Parameters

**Table 37–10** *RESTORE\_DEFAULTS Procedure Parameters*

Parameter	Description
pool_name	Pool to be restored. Currently only the default pool name is supported.

### Exceptions

**Table 37–11** *RESTORE\_DEFAULTS Procedure Exceptions*

Exception	Description
ORA-56500	Connection pool not found
ORA-56507	Connection pool alter configuration failed

### Usage Notes

If the instance is restarted (shutdown followed by startup), the pool is automatically started.



---

---

## DBMS\_CQ\_NOTIFICATION

The `DBMS_CQ_NOTIFICATION` package is part of the database change notification feature that provides the functionality to create registration on queries designated by a client application and so to receive notifications in response to DML or DDL changes on the objects associated with the queries. The notifications are published by the database when the DML or DDL transaction commits.

**See Also:** *Oracle Database Advanced Application Developer's Guide* regarding implementing database change notification.

This chapter contains the following topics:

- [Using DBMS\\_CQ\\_NOTIFICATION](#)
  - Overview
  - Security Model
  - Constants
  - Operational Notes
  - Examples
- [Data Structures](#)
  - OBJECT Types
- [Summary of DBMS\\_CQ\\_NOTIFICATION Subprograms](#)

## Using DBMS\_CQ\_NOTIFICATION

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Operational Notes](#)
- [Examples](#)

## Overview

The `DBMS_CQ_NOTIFICATION` package provides PL/SQL based registration interfaces. A client can use this interface to create registrations on queries based on objects of interest and specify a PL/SQL callback handler to receive notifications. In case of object level registration, when a transaction changes any of the objects associated with the registered queries and `|COMMIT|s`, the PL/SQL callback, specified during registration for those objects, is invoked. The application can define client-specific processing inside the implementation of its PL/SQL callback handler.

The interface lets you define a registration block (using a mechanism similar to a `BEGIN-END` block). The recipient of notifications namely the name of the PL/SQL callback handler and a few other registration properties like time-outs can be specified during the `BEGIN` phase. Any queries executed subsequently (inside the registration block) are considered "interesting queries" and objects referenced by those queries during query execution are registered. The registration is completed by `ENDING` the registration block.

The registration block lets you create new registrations or add objects to existing registrations.

When a registration is created through the PL/SQL interface, a unique registration ID is assigned to the registration by the RDBMS. The client application can use the registration ID to keep track of registrations created by it. When a notification is published by the RDBMS, the registration ID will be part of the notification.

### Typical Applications

This functionality is useful for example to applications that cache query result sets on mostly read-only objects in the mid-tier to avoid network round trips to the database. Such an application can create a registration on the queries it is interested in caching. On changes to objects referenced inside those queries, the database publishes a notification when the underlying transaction commits. In response to the notification, the mid-tier application can refresh its cache by re-executing the query/queries.

## Security Model

The `DBMS_CQ_NOTIFICATION` package requires that the user have the `CHANGE NOTIFICATION` system privilege in order to receive notifications, and be granted `EXECUTE` privilege on the `DBMS_CQ_NOTIFICATION` package.

In addition the user is required to have `SELECT` privileges on all objects to be registered. Note that if the `SELECT` privilege on an object was granted at the time of registration creation but lost subsequently (due to a revoke), then the registration will be purged and a notification to that effect will be published.

## Constants

The DBMS\_CQ\_NOTIFICATION package uses the constants shown in [Table 38–1](#). The constants are used as flag parameters either during registration or when received during the notification.

The DBMS\_CQ\_NOTIFICATION package has sets of constants:

- EVENT\_STARTUP, EVENT\_SHUTDOWN, EVENT\_SHUTDOWN\_ANY, EVENT\_DEREG describe the type of the notification published by the database.
- INSERTOP, DELETEOP, UPDATEOP, ALTEROP, DROPOP and UNKNOWNOP describe the type of operation on a table (during a notification published by the database).
- QOS\_RELIABLE, QOS\_DEREG\_NFY, QOS\_ROWIDS describe registration Quality of Service properties that the client requires. These are specified during registration.

**Table 38–1 DBMS\_CQ\_NOTIFICATION Constants**

Name	Type	Value	Description
ALL_OPERATIONS	BINARY_INTEGER	0	Interested in being notified on all operations, specified as a parameter during registration
ALL_ROWS	BINARY_INTEGER	1	All rows within the table may have been potentially modified
EVENT_STARTUP	BINARY_INTEGER	1	Instance startup notification
EVENT_SHUTDOWN	BINARY_INTEGER	2	Instance shutdown notification
EVENT_SHUTDOWN_ANY	BINARY_INTEGER	3	Any instance shutdown when running Oracle Real Application Clusters (Oracle RAC)
EVENT_DEREG	BINARY_INTEGER	5	Registration has been removed
EVENT_OBJCHANGE	BINARY_INTEGER	6	Notification for object change
EVENT_QUERYCHANGE	BINARY_INTEGER	7	Notification for query result set change
INSERTOP	BINARY_INTEGER	2	Insert operation
UPDATEOP	BINARY_INTEGER	4	Update operation
DELETEOP	BINARY_INTEGER	8	Delete operation
ALTEROP	BINARY_INTEGER	16	Table altered
DROPOP	BINARY_INTEGER	32	Table dropped
UNKNOWNOP	BINARY_INTEGER	64	Unknown operation
QOS_RELIABLE	BINARY_INTEGER	1	Reliable or persistent notification. Also implies that the notifications will be inserted into the persistent storage atomically with the committing transaction that results in an object change.
QOS_DEREG_NFY	BINARY_INTEGER	2	Purge registration on first notification
QOS_ROWIDS	BINARY_INTEGER	4	Require rowids of modified rows
QOS_QUERY	BINARY_INTEGER	8	Register at query granularity
QOS_BEST_EFFORT	BINARY_INTEGER	16	Best effort evaluation

**Table 38–1 (Cont.) DBMS\_CQ\_NOTIFICATION Constants**

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
NTFN_GROUPING_ CLASS_TIME	BINARY_INTEGER	1	Group notifications by time
NTFN_GROUPING_ TYPE_SUMMARY	BINARY_INTEGER	1	Summary grouping of notifications
NTFN_GROUPING_ TYPE_LAST	BINARY_INTEGER	2	Last notification in the group
NTFN_GROUPING_ FOREVER	BINARY_INTEGER	-1	Repeat notifications forever

## Operational Notes

With regard to object level registration:

- The notifications are published by the database when a transaction changes the registered objects and `COMMITs`.
- All objects referenced in the queries executed inside the registration block starting from the previous `NEW_REG_START` or `ENABLE_REG` to `REG_END` are considered interesting objects and added to the registration.

With regard to query result change registration:

- The notifications are published by the database when a transaction changes the result set of the registered query and `COMMITs`.

### Troubleshooting

If you have created a registration and seem to not receive notifications when the underlying tables are changed, please check the following.

- Is the `job_queue_processes` parameter set to a nonzero value? This parameter needs to be configured to a nonzero value in order to receive PL/SQL notifications through the handler.
- Are the registrations being created as a non-SYS user?
- If you are attempting DML changes on the registered object, are you `COMMITting` the transaction? Please note that the notifications are transactional and will be generated when the transaction `COMMITs`.
- It maybe possible that there are run-time errors during the execution of the PL/SQL callback due to implementation errors. If so, they would be logged to the trace file of the `JOBQ` process that attempts to execute the procedure. The trace file would be usually named `<ORACLE_SID>_j*_<PID>.trc.`

For example, if the `ORACLE_SID` is 'dbs1' and the process is 12483, the trace file might be named 'dbs1\_j000\_12483.trc.

Suppose a registration is created with `chnf_callback` as the notification handler and with `registration_id` 100. Let us suppose the user forgets to define the `chnf_callback` procedure. Then the `JOBQ` trace file might contain a message of the following form.

```
Runtime error during execution of PL/SQL cbk chnf_callback for reg CHNF100
Error in PLSQL notification of msgid:
Queue :
Consumer Name :
PLSQL function :chnf_callback
Exception Occured, Error msg:
ORA-00604: error occurred at recursive SQL level 2
ORA-06550: line 1, column 7:
PLS-00201: identifier 'CHNF_CALLBACK' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
```

**See Also:** For more information about troubleshooting Database Change Notification, see *Oracle Database Advanced Application Developer's Guide*.

## Examples

### Object Change Registration Example

Suppose that a mid-tier application has a lot of queries on the HR.EMPLOYEES table. If the EMPLOYEES table is infrequently updated, it can obtain better performance by caching rows from the table because that would avoid a round-trip to the backend database server and server side execution latency. Let us assume that the application has implemented a mid-tier HTTP listener that listens for notifications and updates the mid-tier cache in response to a notification.

The DBMS\_CQ\_NOTIFICATION package can be utilized in this scenario to send notifications about changes to the table by means of the following steps:

1. Implement a mid-tier listener component of the cache management system (for example, using HTTP) that listens to notification messages sent from the database and refreshes the mid-tier cache in response to the notification.
2. Create a server side stored procedure to process notifications

```
CONNECT system;
Enter password: password
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO hr;

Rem Enable job queue processes to receive notifications.
ALTER SYSTEM SET "job_queue_processes"=2;

CONNECT hr;
Enter password: password
Rem Create a table to record notification events
CREATE TABLE nfevents(regid number, event_type number);

Rem create a table to record changes to registered tables
CREATE TABLE nftablechanges(regid number, table_name varchar2(100),
                             table_operation number);

Rem create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(regid number, table_name varchar2(100),
                           row_id varchar2(30));

Rem Create a PL/SQL callback handler to process notifications.
CREATE OR REPLACE PROCEDURE chnf_callback(ntfnds IN SYS.CHNF$_DESC) IS
  regid          NUMBER;
  tbname         VARCHAR2(60);
  event_type     NUMBER;
  numtables      NUMBER;
  operation_type NUMBER;
  numrows        NUMBER;
  row_id         VARCHAR2(20);
BEGIN
  regid          := ntfnds.registration_id;
  numtables      := ntfnds.numtables;
  event_type     := ntfnds.event_type;

  INSERT INTO nfevents VALUES(regid, event_type);
  IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_OBJCHANGE) THEN
    FOR i IN 1..numtables LOOP
      tbname       := ntfnds.table_desc_array(i).table_name;
      operation_type := ntfnds.table_desc_array(i).Opflags;
      INSERT INTO nftablechanges VALUES(regid, tbname, operation_type);
```



```

        /* Send the table name and operation_type to client side listener using
UTL_HTTP */
        /* If interested in the rowids, obtain them as follows */
        IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0) THEN
            numrows := ntfnds.table_desc_array(i).numrows;
        ELSE
            numrows :=0; /* ROWID INFO NOT AVAILABLE */
        END IF;

        /* The body of the loop is not executed when numrows is ZERO */
        FOR j IN 1..numrows LOOP
            Row_id := ntfnds.table_desc_array(i).row_desc_array(j).row_id;
            INSERT INTO nfrowchanges VALUES(regid, tname, Row_id);
            /* optionally Send out row_ids to client side listener using UTL_HTTP;
*/
        END LOOP;

    END LOOP;
END IF;
COMMIT;
END;
/

```

In Step 2 we can send as much information about the invalidation as the mid-tier application needs based on the information obtained from the notification descriptor.

#### Notes

- a. In the above example, a registration was created on the EMPLOYEES table with 'chnf\_callback' as the PL/SQL handler for notifications. During registration, the client specified reliable notifications (QOS\_RELIABLE) and rowid notifications (QOS\_ROWIDS)
- b. The handler accesses the table descriptor array from the notification descriptor only if the notification type is of EVENT\_OBJCHANGE. In all other cases (e.g EVENT\_DEREG, EVENT\_SHUTDOWN), the table descriptor array should not be accessed.
- c. The handler accesses the row descriptor array from the table notification descriptor only if the ALL\_ROWS bit is not set in the table operation flag. If the ALL\_ROWS bit is set in the table operation flag, then it means that all rows within the table may have been potentially modified. In addition to operations like TRUNCATE that affect all rows in the tables, this bit may also be set if individual rowids have been rolled up into a FULL table invalidation.

This can occur if too many rows were modified on a given table in a single transaction (more than 80) or the total shared memory consumption due to rowids on the RDBMS is determined too large (exceeds 1% of the dynamic shared pool size). In this case, the recipient must conservatively assume that the entire table has been invalidated and the callback/application must be able to handle this condition.

Also note that the implementation of the user defined callback is up to the developer. In the above example, the callback was used to record event details into database tables. The application can additionally send the notification details to a mid-tier HTTP listener of its cache management system (as in the example) using UTL\_HTTP. The listener could then refresh its cache by querying from the back-end database.

3. Create a registrations on the tables that we wish to be notified about. We pass in the previously defined procedure name (chnf\_callback) as the name of the server side PL/SQL procedure to be executed when a notification is generated.

```

Rem Create a REGISTRATION on the EMPLOYEES TABLE
DECLARE
  REGDS      SYS.CHNF$_REG_INFO;
  regid      NUMBER;
  mgr_id     NUMBER;
  dept_id    NUMBER;
  qosflags   NUMBER;
BEGIN
  qosflags := DBMS_CQ_NOTIFICATION.QOS_RELIABLE +
             DBMS_CQ_NOTIFICATION.QOS_ROWIDS;
  REGDS := SYS.CHNF$_REG_INFO ('chnf_callback', qosflags, 0,0,0);
  regid := DBMS_CQ_NOTIFICATION.NEW_REG_START (REGDS);
  SELECT manager_id INTO mgr_id FROM EMPLOYEES WHERE employee_id = 200;
  DBMS_CQ_NOTIFICATION.REG_END;
END;
/

```

Once the registration is created in Step 3 above, the server side PL/SQL procedure defined in Step 2 is executed in response to any COMMITted changes to the HR.EMPLOYEES table. As an example, let us assume that the following update is performed on the employees table.

```

UPDATE employees SET salary=salary*1.05 WHERE employee_id=203;
COMMIT;

```

Once the notification is processed, you will find rows which might look like the following in the nfevents, nftablechanges and nfrowchanges tables.

```
SQL> SELECT * FROM nfevents;
```

REGID	EVENT_TYPE
20045	6

```
SQL> SELECT * FROM nftablechanges;
```

REGID	TABLE_NAME	TABLE_OPERATION
20045	HR.EMPLOYEES	4

```
SQL> select * from nfrowchanges;
```

REGID	TABLE_NAME	ROW_ID
20045	HR.EMPLOYEES	AAAKB/AABAAAJ8zAAF

## Query Result Change Registration Example

### 1. Creating a Callback

```

CONNECT system;
Enter password: password
GRANT CHANGE NOTIFICATION TO hr;
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO hr;
CONNECT hr;
Enter password: password
Rem Create a table to record notification events
CREATE TABLE nfevents(regid NUMBER, event_type NUMBER);

```

```

Rem Create a table to record notification queries
CREATE TABLE nfqueries (qid NUMBER, qop NUMBER);

Rem Create a table to record changes to registered tables
CREATE TABLE nftablechanges(
    qid                NUMBER,
    table_name         VARCHAR2(100),
    table_operation    NUMBER);

Rem Create a table to record rowids of changed rows.
CREATE TABLE nfrowchanges(
    qid                NUMBER,
    table_name         VARCHAR2(100),
    row_id             VARCHAR2(2000));

CREATE OR REPLACE PROCEDURE chnf_callback
    (ntfnds IN CQ_NOTIFICATION$_DESCRIPTOR)
IS
    regid            NUMBER;
    tbname           VARCHAR2(60);
    event_type       NUMBER;
    numtables        NUMBER;
    operation_type   NUMBER;
    numrows          NUMBER;
    row_id           VARCHAR2(2000);
    numqueries       NUMBER;
    qid NUMBER;
    qop NUMBER;

BEGIN
    regid := ntfnds.registration_id;
    event_type := ntfnds.event_type;
    INSERT INTO nfevents VALUES(regid, event_type);
    numqueries :=0;
    IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_QUERYCHANGE) THEN
        numqueries := ntfnds.query_desc_array.count;
        FOR i in 1..numqueries LOOP
            qid := ntfnds.QUERY_DESC_ARRAY(i).queryid;
            qop := ntfnds.QUERY_DESC_ARRAY(i).queryop;
            INSERT INTO nfqueries VALUES(qid, qop);
            numtables := 0;
            numtables := ntfnds.QUERY_DESC_ARRAY(i).table_desc_array.count;
            FOR j IN 1..numtables LOOP
                tbname := ntfnds.QUERY_DESC_ARRAY(i).table_desc_array(j).table_name;
                operation_type := ntfnds.QUERY_DESC_ARRAY(i).table_desc_
array(j).Opflags;
                INSERT INTO nftablechanges VALUES(qid, tbname, operation_type);
                IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0)
                THEN
                    numrows := ntfnds.query_desc_array(i).table_desc_array(j).numrows;
                ELSE
                    numrows :=0; /* ROWID INFO NOT AVAILABLE */
                END IF;

                /* The body of the loop is not executed when numrows is ZERO */
                FOR k IN 1..numrows LOOP
                    Row_id := ntfnds.query_desc_array(i).table_desc_array(j).row_
desc_array(k).row_id;
                    INSERT INTO nfrowchanges VALUES(qid, tbname, Row_id);

```

```

        END LOOP; /* loop over rows */
    END LOOP; /* loop over tables */
END LOOP; /* loop over queries */
END IF;
COMMIT;
END;
/

```

## 2. Creates a query registration

```

DECLARE
    reginfo    cq_notification$_reg_info;
    mgr_id     NUMBER;
    dept_id    NUMBER;
    v_cursor   SYS_REFCURSOR;
    regid      NUMBER;
    qosflags   NUMBER;

BEGIN
    /* Register two queries for result-set-change notifications: */

    /* 1. Construct registration information.
       'chnf_callback' is name of notification handler.
       QOS_QUERY specifies result-set-change notifications. */

    qosflags := DBMS_CQ_NOTIFICATION.QOS_QUERY +
                DBMS_CQ_NOTIFICATION.QOS_ROWIDS;

    reginfo := cq_notification$_reg_info('chnf_callback', qosflags,0, 0, 0);

    /* 2. Create registration */

    regid := DBMS_CQ_NOTIFICATION.NEW_REG_START(reginfo);

    OPEN v_cursor FOR
        SELECT DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID, manager_id
        FROM HR.employees
        WHERE employee_id = 7902;
    CLOSE v_cursor;

    OPEN v_cursor for
        SELECT DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID, department_id
        FROM HR.departments
        WHERE department_name = 'IT';
    CLOSE v_cursor;

    DBMS_CQ_NOTIFICATION.REG_END;
END;
/

```

## 3. After creating the query registrations, the output from USER\_CQ\_NOTIFICATION\_QUERIES would appear as follows.

```

SQL> SELECT queryid, regid, to_char(querytext)
       FROM user_cq_notification_queries;

```

```

      QUERYID      REGID
-----
TO_CHAR (QUERYTEXT)

```

```
-----
-
          22          41
SELECT HR.DEPARTMENTS.DEPARTMENT_ID FROM HR.DEPARTMENTS WHERE
HR.DEPARTMENTS.
DEPARTMENT_NAME = 'IT'
```

```
          21          41
SELECT HR.EMPLOYEES.MANAGER_ID FROM HR.EMPLOYEES WHERE
HR.EMPLOYEES.EMPLOYEE_
ID = 7902
```

Now, let us perform an UPDATE that changes the result of the query with queryid 22 by renaming the department with name 'IT' to FINANCE.

```
SQL> update departments set department_name = 'FINANCE' where department_name
= 'IT';
```

1 row updated.

```
SQL> commit;
```

Commit complete.

Now we can query the notifications that we recorded in the callback.

```
SQL> select * from nfevents;
```

```
          REGID EVENT_TYPE
-----
          61          7
```

Event type 7 corresponds to EVENT\_QUERYCHANGE

```
SQL> select * from nfqueries;
```

```
          QID          QOP
-----
          42          7
```

Event type 7 corresponds to EVENT\_QUERYCHANGE

```
SQL> select * from nftablechanges;
```

```
SQL> select * from nftablechanges;
```

```
          REGID
-----
TABLE_NAME
-----
-
TABLE_OPERATION
-----
          42
HR.DEPARTMENTS
          4
```

TABLE\_OPERATION 4 corresponds to UPDATEOP

```
SQL> select * from nfrowchanges;
```

```
      REGID
-----
TABLE_NAME
-----
-
ROW_ID
-----
-
          61
HR.DEPARTMENTS
AAANKdAABAAALinAAF
```

## Data Structures

The DBMS\_CQ\_NOTIFICATION package defines the following OBJECT types.

### OBJECT Types

- CQ\_NOTIFICATION\$\_DESCRIPTOR Object Type
- CQ\_NOTIFICATION\$\_QUERY Object Type
- CQ\_NOTIFICATION\$\_QUERY\_ARRAY Object (Array) Type
- CQ\_NOTIFICATION\$\_TABLE Object Type
- CQ\_NOTIFICATION\$\_TABLE\_ARRAY Object (Array) Type
- CQ\_NOTIFICATION\$\_ROW Object Type
- CQ\_NOTIFICATION\$\_ROW\_ARRAY Object (Array) Type
- CQ\_NOTIFICATION\$\_REG\_INFO Object Type

## CQ\_NOTIFICATION\$\_DESCRIPTOR Object Type

This is the top level change notification descriptor type. It is a synonym for the SYS.CHNF\$\_DESC type.

### Syntax

```
TYPE SYS.CHNF$_DESC IS OBJECT(
  registration_id    NUMBER,
  transaction_id    RAW(8),
  dbname            VARCHAR2(30),
  event_type        NUMBER,
  numtables         NUMBER,
  table_desc_array  CQ_NOTIFICATION$_TABLE_ARRAY,
  query_desc_array  CQ_NOTIFICATION$_QUERY_ARRAY);
```

### Attributes

**Table 38–2 CQ\_NOTIFICATION\$\_DESCRIPTOR Object Type**

Attribute	Description
registration_id	Registration ID returned during registration
transaction_id	Transaction ID. transaction_id of the transaction that made the change. Will be NULL unless the event_type is EVENT_OBJCHANGE or EVENT_QUERYCHANGE.
dbname	Name of database
event_type	Database event associated with the notification. Can be one of EVENT_OBJCHANGE (change to a registered object), EVENT_STARTUP, or EVENT_QUERYCHANGE, EVENT_SHUTDOWN or EVENT_DEREG (registration has been removed due to a timeout or other reason)
numtables	Number of modified tables. Will be NULL unless the event_type is EVENT_OBJCHANGE.
table_desc_array	Array of table descriptors. Will be NULL unless the event_type is EVENT_OBJCHANGE.
query_desc_array	Array of queries changed. This will be NULL unless event_type is EVENT_QUERYCHANGE



## CQ\_NOTIFICATION\$\_QUERY Object Type

The object type describes the changes to a query result caused by an event such as a transaction commit. An array of CQ\_NOTIFICATION\$\_QUERY descriptors is embedded inside the top level notification descriptor (CQ\_NOTIFICATION\$\_DESCRIPTOR) for events of type EVENT\_QUERYCHANGE. The array corresponds to the SET of queryids which were invalidated as a result of the event.

This is a synonym for the base type SYS.CHNF\$\_QDESC.

### Syntax

```
TYPE SYS.CHNF$_QDESC IS OBJECT (
  queryid          NUMBER,
  queryop         NUMBER,
  table_desc_array CQ_NOTIFICATION$_TABLE_ARRAY);
```

### Attributes

**Table 38-3** TYPE SYS.CQ\_NOTIFICATION\$\_QUERY Object Type

Attribute	Description
queryid	QueryId of the changed query
queryop	Operation describing change to the query
table_desc_array	Array of table changes which contributed to the query Result Set change

## CQ\_NOTIFICATION\$\_QUERY\_ARRAY Object (Array) Type

This type corresponds to an array of CQ\_NOTIFICATION\$\_QUERY objects. It is a synonym for the SYS.CHNF\$\_QUERY\_ARRAY type.

### Syntax

```
TYPE CQ_NOTIFICATION$_TABLE_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_TABLE;
```

## CQ\_NOTIFICATION\$\_TABLE Object Type

This descriptor type describes a change to a table and is embedded inside the top level change notification descriptor type for events of type `EVENT_OBJCHANGE`. For query result set changes (event type will be set to `EVENT_QUERYCHANGE`), the array of table descriptors is embedded inside each query change descriptor.

Note that this is a synonym for the type previously named `SYS.CHNF$_TDESC`.

### Syntax

```
TYPE SYS.CHNF$_TDESC IS OBJECT (
  opflags          NUMBER,
  table_name       VARCHAR2(2*M_IDEN+1),
  numrows          NUMBER,
  row_desc_array   CQ_NOTIFICATION$_ROW_ARRAY)
```

### Attributes

**Table 38-4** TYPE `SYS.CQ_NOTIFICATION$_TABLE` Object Type

Attribute	Description
<code>opflags</code>	Table level operation flags. This is a flag field (bit-vector) that describes the operations that occurred on the table. It can be an OR of the following bit fields - <code>INSERTOP</code> , <code>UPDATEOP</code> , <code>DELETEOP</code> , <code>DROPOP</code> , <code>ALTEROP</code> , <code>ALL_ROWS</code> . If the <code>ALL_ROWS</code> (0x1) bit is set it means that either the entire table is modified (for example, <code>DELETE * FROM t</code> ) or row level granularity of information is not requested or not available in the notification and the receiver has to conservatively assume that the entire table has been invalidated.
<code>table_name</code>	Name of modified table
<code>numrows</code>	Number of modified rows within the table. <code>numrows</code> will be <code>NULL</code> and hence should not be accessed if the <code>ALL_ROWS</code> bit is set in the table change descriptor.
<code>row_desc_array</code>	Array of row descriptors. This field will be <code>NULL</code> if the <code>ALL_ROWS</code> bit is set in <code>opflags</code> .

## CQ\_NOTIFICATION\$\_TABLE\_ARRAY Object (Array) Type

This type corresponds to an array of CQ\_NOTIFICATION\$\_TABLE objects. It is a synonym for the SYS.CHNF\$\_TDESC\_ARRAY type.

### Syntax

```
TYPE CQ_NOTIFICATION$_TABLE_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_TABLE;
```

## CQ\_NOTIFICATION\$\_ROW Object Type

An array of CQ\_NOTIFICATION\$\_ROW is embedded inside a CQ\_NOTIFICATION\$\_TABLE (table change descriptor) if the QOS\_ROWIDS option was chosen at the time of registration and the ALL\_ROWS bit is not set in the opflags field of the table change descriptor.

Note that this is a synonym for the type previously named SYS.CHNF\$\_RDESC.

### Syntax

```
TYPE SYS.CHNF$_RDESC IS OBJECT (
  opflags          NUMBER,
  row_id           VARCAHR2 (2000));
```

### Attributes

**Table 38-5** TYPE SYS.CQ\_NOTIFICATION\$\_ROW Object Type

Attribute	Description
opflags	Row level operation flags. The flag field (bit vector) describes the operations in the row (could be INSERTOP, UPDATEOP or DELETEOP).
row_id	The rowid of the modified row

## CQ\_NOTIFICATION\$\_ROW\_ARRAY Object (Array) Type

This object type corresponds to an array of CQ\_NOTIFICATION\$\_ROW objects and is embedded inside the CQ\_NOTIFICATION\$\_TABLE if QOS\_ROWIDS was specified during registration and the ALL\_ROWS bit is not set in the opflags field of the table change descriptor.

This type is a synonym for the SYS.CHNF\$\_RDESC\_ARRAY type.

### Syntax

```
TYPE CQ_NOTIFICATION$_ROW_ARRAY IS VARRAY (1073741824) OF CQ_NOTIFICATION$_ROW;
```

## CQ\_NOTIFICATION\$\_REG\_INFO Object Type

The object type describes the attributes associated with creating a new registration. It is a synonym for the type previously named `SYS.CHNF$_REG_INFO`.

### Syntax

```
TYPE SYS.CHNF$_REG_INFO IS OBJECT (
  callback                VARCHAR2(20) ,
  quosflags               NUMBER,
  timeout                 NUMBER,
  operations_filter       NUMBER,
  transaction_lag         NUMBER,
  ntfn_grouping_class     NUMBER,
  ntfn_grouping_value     NUMBER,
  ntfn_grouping_type      NUMBER,
  ntfn_grouping_start_time  TIMESTAMP WITH TIME ZONE,
  ntfn_grouping_repeat_count NUMBER);
```

### Attributes

**Table 38–6** TYPE CQ\_NOTIFICATION\$\_REG\_INFO Object Type

Attribute	Description
callback	Name of the server side PL/SQL procedure to be executed on a notification. Prototype is <code>&lt;callbackname&gt;(ntfnds IN SYS.chnf\$_desc)</code>
quosflags	Quality of service flags. Can be set to an OR of the following values: <ul style="list-style-type: none"> <li>▪ <code>QOS_RELIABLE (0x1)</code>: Notifications are reliable (persistent) and survive instance death. This means that on an instance death in an Oracle RAC cluster, surviving instances will be able to deliver any queued invalidations. Similarly, pending invalidations can be delivered on instance restart, in a single instance configuration. The disadvantage is that there is a CPU cost/ latency involved in inserting the invalidation message to a persistent store. If this parameter is false, then server side CPU and latency are minimized, because invalidations are buffered into an in memory queue but the client could lose invalidation messages on an instance shutdown.</li> <li>▪ <code>QOS_DEREG_NFY (0x2)</code>: The registration will be expunged on the first notification</li> <li>▪ <code>QOS_ROWIDS (0x4)</code>: The notification needs to include information about the rowids that were modified</li> <li>▪ <code>QOS_QUERY (0x8)</code>: specifies query result change notification as opposed to object change notification</li> <li>▪ <code>QOS_BEST_EFFORT (0x16)</code>: can register simplified versions of queries and minimizes evaluation with some false positives.</li> </ul>

**Table 38–6 (Cont.) TYPE CQ\_NOTIFICATIONS\$ REG\_INFO Object Type**

Attribute	Description
timeout	If set to a nonzero value, specifies the time in seconds after which the registration is automatically expunged by the database. If zero / NULL, the registration lives until explicitly deregistered. Note that the <code>timeout</code> option can be combined with the <code>purge on notification (QOS_DEREG_NFY)</code> option as well.
operations_filter	<p>if nonzero, specifies a filter to be selectively notified on certain operations. These flags can be used to filter based on specific operation types:</p> <ul style="list-style-type: none"> <li>▪ 0: Notify on all operations (<code>DBMS_CQ_NOTIFICATION.ALL_OPERATIONS</code>)</li> <li>▪ 0x2: Notify on every INSERT (<code>DBMS_CQ_NOTIFICATION.INSERTOP</code>)</li> <li>▪ 0x4: Notify on every UPDATE (<code>DBMS_CQ_NOTIFICATION.UPDATEOP</code>)</li> <li>▪ 0x8: Notify on every DELETE (<code>DBMS_CQ_NOTIFICATION.DELETEOP</code>)</li> </ul> <p>A combination of operations can be specified by using a bitwise OR.</p> <p>Caution: This parameter will be honored for object level registrations but ignored for query result change registrations. To implement notification flow control in 11g, the applications can use the "GROUPING notification" option.</p>
transaction_lag	<p>Lag between consecutive notifications in units of transactions. Can be used to specify the number of transactions/database changes, by which the client is willing to lag behind the database. If 0, it means that the client needs to receive an invalidation message as soon as it is generated.</p> <p><b>Caution:</b> This parameter will be honored for object level registrations but ignored for query result change notification registrations.</p>
ntfn_grouping_class	When grouping notifications, the class based on which the group is derived. Currently, the only allowed value is <code>DBMS_CQ_NOTIFICATION.NTFN_GROUPING_CLASS_TIME</code> by which notifications are grouped by time.
ntfn_grouping_value	The grouping value. This describes the time interval that defines the group in seconds. For example, if this were set to 900, it would mean that notifications that were generated in each 15 minute interval would be grouped together.
ntfn_grouping_type	<p>The type of grouping desired. It can be one of two allowed values</p> <ul style="list-style-type: none"> <li>▪ <code>DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_SUMMARY</code> - all notifications in the group are summarized into a single notification</li> <li>▪ <code>DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_LAST</code> - only the last notification in the group is published and the earlier ones discarded</li> </ul>
ntfn_grouping_start_time	When to start generating notifications. If specified as NULL, it defaults to the current system generated time.



**Table 38–6 (Cont.) TYPE CQ\_NOTIFICATION\$\_REG\_INFO Object Type**

Attribute	Description
ntfn_grouping_repeat_count	How many times the notification should be repeated. Set this to DBMS_CQ_NOTIFICATION.NTFN_GROUPING_FOREVER to receive notifications for the life time of the registration. Set to a nonzero value if only a certain number of notifications are desired for the life time of the registration.

## Usage Notes

- The type declaration incorporates three other alternative constructors. In the first case all other parameters default to their default values.

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
  callback          VARCHAR2(20) ,
  quosflags         NUMBER,
  timeout           NUMBER);
```

The second option applies to the type constructor defined in a previous release, and which is retained for backward compatibility:

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
  callback          VARCHAR2(20) ,
  quosflags         NUMBER,
  timeout           NUMBER,
  operations_filter NUMBER,
  transaction_lag   NUMBER);
```

The third definition contains all the members of the type except `transaction_lag` which is being deprecated:

```
TYPE CQ_NOTIFICATION$_REG_INFO IS OBJECT (
  callback          VARCHAR2(20) ,
  quosflags         NUMBER,
  timeout           NUMBER,
  operations_filter NUMBER,
  ntfn_grouping_class      NUMBER,
  ntfn_grouping_value      NUMBER,
  ntfn_grouping_type       NUMBER,
  ntfn_grouping_start_time  TIMESTAMP WITH TIME ZONE,
  ntfn_grouping_repeat_count NUMBER);
```

- In response to a database change, the server side PL/SQL procedure specified by "callback" is executed. The PL/SQL procedure name has to be specified in the format `schema_name.procedure_name`. The procedure must have the following signature:

```
PROCEDURE <procedure_name>(ntfnds IN SYS.chnf$_desc)
```

CHNF\$\_DESC describes the change notification descriptor.

- The `init.ora` parameter `job_queue_processes` must be set to a nonzero value to receive PL/SQL notifications, because the specified procedure is executed inside a job queue process when a notification is generated.

## Summary of DBMS\_CQ\_NOTIFICATION Subprograms

**Table 38–7** *DBMS\_CQ\_NOTIFICATION Package Subprograms*

Subprogram	Description
<a href="#">CQ_NOTIFICATION_QUERYID Function</a> on page 38-27	Returns the queryid of the most recent query that was attempted to be registered in a registration block
<a href="#">DEREGISTER Procedure</a> on page 38-28	De-subscribes the client with the supplied registration identifier (ID)
<a href="#">ENABLE_REG Procedure</a> on page 38-29	Begins a registration block using an existing registration identifier (ID)
<a href="#">NEW_REG_START Function</a> on page 38-30	Begins a new registration block
<a href="#">REG_END Procedure</a> on page 38-31	Ends the registration boundary
<a href="#">SET_ROWID_THRESHOLD Procedure</a> on page 38-32	Configures the maximum number of rows of a table published in a change notification if the rows of the table are modified in a transaction

## **CQ\_NOTIFICATION\_QUERYID Function**

This function returns the queryid of the most recent query that was attempted to be registered in a registration block.

### **Syntax**

```
DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID  
RETURN NUMBER;
```

### **Return Values**

Returns the queryid of the most recently registered query.

## DEREGISTER Procedure

This procedure desubscribes the client with the specified registration identifier (ID).

### Syntax

```
DBMS_CQ_NOTIFICATION.DEREGISTER (  
    regid IN NUMBER);
```

### Parameters

**Table 38–8** *DEREGISTER Procedure Parameters*

Parameter	Description
regid	Client registration ID

### Usage Notes

Only the user that created the registration (or the SYS user) will be able to desubscribe the registration.

## ENABLE\_REG Procedure

This procedure adds objects to an existing registration identifier (ID). It is similar to the interface for creating a new registration, except that it takes an existing `regid` to which to add objects.

Subsequent execution of queries causes the objects referenced in the queries to be added to the specified `regid`, and the registration is completed on invoking the [REG\\_END Procedure](#).

### Syntax

```
DBMS_CQ_NOTIFICATION.ENABLE_REG (  
    regid IN NUMBER);
```

### Parameters

**Table 38–9** *ENABLE\_REG Procedure Parameters*

Parameter	Description
<code>regid</code>	Client registration ID

### Usage Notes

Only the user that created the registration will be able to add further objects to the registration.

## NEW\_REG\_START Function

This procedure begins a new registration block. Any objects referenced by queries executed within the registration block are considered interesting objects and added to the registration. The registration block ends upon calling the REG\_END procedure.

### Syntax

```
DBMS_CQ_NOTIFICATION.NEW_REG_START (
  regds IN sys.chnf$_reg_info)
RETURN NUMBER;
```

### Parameters

**Table 38–10 NEW\_REG\_START Function Parameters**

Parameter	Description
sys.chnf\$_reg_info	Registration descriptor describing the notification handler and other properties of the registration

### Return Values

The procedure returns a registration-id which is a unique integer assigned by the database to this registration. The registration-id will be echoed back in every notification received for this registration.

### Usage Notes

- The only operations permitted inside a registration block are queries (the ones the user wishes to register). DML and DDL operations are not permitted.
- The registration block is a session property and implicitly terminates upon exiting the session. While the registration block is a session property, the registration itself is a persistent database entity. Once created, the registration survives until explicitly deregistered by the client application or timed-out or removed by the database for some other reason (such as loss of privileges).
- The user must have the CHANGE\_NOTIFICATION system privilege and SELECT privileges on any objects to be registered.
- The SYS user will not be permitted to create new registrations.
- Nesting of registration block is not permitted.

## **REG\_END Procedure**

This procedure marks the end of the registration block. No newly executed queries are tracked.

### **Syntax**

```
DBMS_CQ_NOTIFICATION.REG_END;
```

## SET\_ROWID\_THRESHOLD Procedure

This procedure configures the maximum number of rows of a table published in a change notification if the rows of the table are modified in a transaction.

### Syntax

```
DBMS_CQ_NOTIFICATION.SET_ROWID_THRESHOLD (  
  tbnname      IN VARCHAR2,  
  threshold    IN NUMBER);
```

### Parameters

**Table 38–11** SET\_ROWID\_THRESHOLD Procedure Parameters

Parameter	Description
tbnname	Table name qualified by the schema name in the form <code>schemaname.tablename</code>
threshold	Maximum number of modified rows of the table to be published in the change notification

### Usage Notes

- The table needs to be registered for change notification either at object change granularity or at query result set granularity.
- The threshold set by means of this subprogram applies to that instance only and does not persist across instance startup/shutdown.



DBMS\_CRYPTO provides an interface to encrypt and decrypt stored data, and can be used in conjunction with PL/SQL programs running network communications. It provides support for several industry-standard encryption and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES has been approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES).

**See Also:** *Oracle Database Security Guide* for further information about using this package and about encrypting data in general.

This chapter contains the following topics:

- [Using the DBMS\\_CRYPTO Subprograms](#)
  - Overview
  - Security Model
  - Types
  - Algorithms
  - Restrictions
  - Exceptions
  - Operational Notes
- [Summary of DBMS\\_CRYPTO Subprograms](#)

---

## Using the DBMS\_CRYPTO Subprograms

- [Overview](#)
- [Security Model](#)
- [Types](#)
- [Algorithms](#)
- [Restrictions](#)
- [Exceptions](#)
- [Operational Notes](#)

## Overview

DBMS\_CRYPT0 contains basic cryptographic functions and procedures. To use this package correctly and securely, a general level of security expertise is assumed.

The DBMS\_CRYPT0 package enables encryption and decryption for common Oracle datatypes, including RAW and large objects (LOBs), such as images and sound. Specifically, it supports BLOBs and CLOBs. In addition, it provides Globalization Support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES, 2-key and 3-key)
- Advanced Encryption Standard (AES)
- MD5, MD4, and SHA-1 cryptographic hashes
- MD5 and SHA-1 Message Authentication Code (MAC)

Block cipher modifiers are also provided with DBMS\_CRYPT0. You can choose from several padding options, including PKCS (Public Key Cryptographic Standard) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC).

Table 39–1 lists the DBMS\_CRYPT0 package features in comparison to the other PL/SQL encryption package, the DBMS\_OBFUSCATION\_TOOLKIT.

**Table 39–1 DBMS\_CRYPT0 and DBMS\_OBFUSCATION\_TOOLKIT Feature Comparison**

Package Feature	DBMS_CRYPT0	DBMS_OBFUSCATION_TOOLKIT
Cryptographic algorithms	DES, 3DES, AES, RC4, 3DES_2KEY	DES, 3DES
Padding forms	PKCS5, zeroes	none supported
Block cipher chaining modes	CBC, CFB, ECB, OFB	CBC
Cryptographic hash algorithms	MD5, SHA-1, MD4	MD5
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1	none supported
Cryptographic pseudo-random number generator	RAW, NUMBER, BINARY_INTEGER	RAW, VARCHAR2
Database types	RAW, CLOB, BLOB	RAW, VARCHAR2

DBMS\_CRYPT0 is intended to replace the DBMS\_OBFUSCATION\_TOOLKIT, providing greater ease of use and support for a range of algorithms to accommodate new and existing systems. Specifically, 3DES\_2KEY and MD4 are provided for backward compatibility. It is not recommended that you use these algorithms because they do not provide the same level of security as provided by 3DES, AES, MD5, or SHA-1.

## Security Model

Oracle Database installs this package in the `SYS` schema. You can then grant package access to existing users and roles as needed.

## Types

Parameters for the DBMS\_CRYPT0 subprograms use these datatypes:

**Table 39–2 DBMS\_CRYPT0 Datatypes**

Type	Description
BLOB	A source or destination binary LOB
CLOB	A source or destination character LOB (excluding NCLOB)
PLS_INTEGER	Specifies a cryptographic algorithm type (used with BLOB, CLOB, and RAW datatypes)
RAW	A source or destination RAW buffer

## Algorithms

The following cryptographic algorithms, modifiers, and cipher suites are predefined in this package.

**Table 39–3 DBMS\_CRYPTO Cryptographic Hash Functions**

Name	Description
HASH_MD4	Produces a 128-bit hash, or message digest of the input message
HASH_MD5	Also produces a 128-bit hash, but is more complex than MD4
HASH_SH1	Secure Hash Algorithm (SHA). Produces a 160-bit hash.

**Table 39–4 DBMS\_CRYPTO MAC (Message Authentication Code) Functions**

Name	Description
HMAC_MD5 <sup>1</sup>	Same as MD5 hash function, except it requires a secret key to verify the hash value.
HMAC_SH1 <sup>1</sup>	Same as SHA hash function, except it requires a secret key to verify the hash value.

<sup>1</sup> Complies with IETF RFC 2104 standard

**Table 39–5 DBMS\_CRYPTO Encryption Algorithms**

Name	Description
ENCRYPT_DES	Data Encryption Standard. Block cipher. Uses key length of 56 bits.
ENCRYPT_3DES_2KEY	Data Encryption Standard. Block cipher. Operates on a block 3 times with 2 keys. Effective key length of 112 bits.
ENCRYPT_3DES	Data Encryption Standard. Block cipher. Operates on a block 3 times.
ENCRYPT_AES128	Advanced Encryption Standard. Block cipher. Uses 128-bit key size.
ENCRYPT_AES192	Advanced Encryption Standard. Block cipher. Uses 192-bit key size.
ENCRYPT_AES256	Advanced Encryption Standard. Block cipher. Uses 256-bit key size.
ENCRYPT_RC4	Stream cipher. Uses a secret, randomly generated key unique to each session.

**Table 39–6 DBMS\_CRYPTO Block Cipher Suites**

Name	Description
DES_CBC_PKCS5	ENCRYPT_DES <sup>1</sup> + CHAIN_CBC <sup>2</sup> + PAD_PKCS5 <sup>3</sup>
DES3_CBC_PKCS5	ENCRYPT_3DES <sup>1</sup> + CHAIN_CBC <sup>2</sup> + PAD_PKCS5 <sup>3</sup>

<sup>1</sup> See Table 39–5, "DBMS\_CRYPTO Encryption Algorithms"

<sup>2</sup> See Table 39–7, "DBMS\_CRYPTO Block Cipher Chaining Modifiers"

<sup>3</sup> See Table 39–8, "DBMS\_CRYPTO Block Cipher Padding Modifiers"

**Table 39–7 DBMS\_CRYPT0 Block Cipher Chaining Modifiers**

<b>Name</b>	<b>Description</b>
CHAIN_ECB	Electronic Codebook. Encrypts each plaintext block independently.
CHAIN_CBC	Cipher Block Chaining. Plaintext is XORed with the previous ciphertext block before it is encrypted.
CHAIN_CFB	Cipher-Feedback. Enables encrypting units of data smaller than the block size.
CHAIN_OFB	Output-Feedback. Enables running a block cipher as a synchronous stream cipher. Similar to CFB, except that <i>n</i> bits of the previous output block are moved into the right-most positions of the data queue waiting to be encrypted.

**Table 39–8 DBMS\_CRYPT0 Block Cipher Padding Modifiers**

<b>Name</b>	<b>Description</b>
PAD_PKCS5	Provides padding which complies with the PKCS #5: Password-Based Cryptography Standard
PAD_NONE	Provides option to specify no padding. Caller must ensure that blocksize is correct, else the package returns an error.
PAD_ZERO	Provides padding consisting of zeroes.

## Restrictions

The `VARCHAR2` datatype is not directly supported by `DBMS_CRYPTO`. Before you can perform cryptographic operations on data of the type `VARCHAR2`, you must convert it to the uniform database character set `AL32UTF8`, and then convert it to the `RAW` datatype. After performing these conversions, you can then encrypt it with the `DBMS_CRYPTO` package.

**See Also:** ["Conversion Rules"](#) on page 39-11 for information about converting datatypes.



## Exceptions

Table 39–9 lists exceptions that have been defined for DBMS\_CRYPT0.

**Table 39–9 DBMS\_CRYPT0 Exceptions**

Exception	Code	Description
CipherSuiteInvalid	28827	The specified cipher suite is not defined.
CipherSuiteNull	28829	No value has been specified for the cipher suite to be used.
KeyNull	28239	The encryption key has not been specified or contains a NULL value.
KeyBadSize	28234	DES keys: Specified key size is too short. DES keys must be at least 8 bytes (64 bits). AES keys: Specified key size is not supported. AES keys must be 128, 192, or 256 bits in length.
DoubleEncryption	28233	Source data was previously encrypted.

## Operational Notes

- [When to Use Encrypt and Decrypt Procedures or Functions](#)
- [When to Use Hash or Message Authentication Code \(MAC\) Functions](#)
- [About Generating and Storing Encryption Keys](#)
- [Conversion Rules](#)

### When to Use Encrypt and Decrypt Procedures or Functions

This package includes both `ENCRYPT` and `DECRYPT` procedures and functions. The procedures are used to encrypt or decrypt LOB datatypes (overloaded for CLOB and BLOB datatypes). In contrast, the `ENCRYPT` and `DECRYPT` functions are used to encrypt and decrypt RAW datatypes. Data of type `VARCHAR2` must be converted to RAW before you can use `DBMS_CRYPTO` functions to encrypt it.

### When to Use Hash or Message Authentication Code (MAC) Functions

This package includes two different types of one-way hash functions: the `HASH` function and the `MAC` function. Hash functions operate on an arbitrary-length input message, and return a fixed-length hash value. One-way hash functions work in one direction only. It is easy to compute a hash value from an input message, but it is extremely difficult to generate an input message that hashes to a particular value. Note that hash values should be at least 128 bits in length to be considered secure.

You can use hash values to verify whether data has been altered. For example, before storing data, Laurel runs `DBMS_CRYPTO.HASH` against the stored data to create a hash value. When she returns the stored data at a later date, she can again run the hash function against it, using the same algorithm. If the second hash value is identical to the first one, then the data has not been altered. Hash values are similar to "file fingerprints" and are used to ensure data integrity.

The `HASH` function included with `DBMS_CRYPTO`, is a one-way hash function that you can use to generate a hash value from either RAW or LOB data. The `MAC` function is also a one-way hash function, but with the addition of a secret key. It works the same way as the `DBMS_CRYPTO.HASH` function, except only someone with the key can verify the hash value.

MACs can be used to authenticate files between users. They can also be used by a single user to determine if her files have been altered, perhaps by a virus. A user could compute the MAC of his files and store that value in a table. If the user did not use a MAC function, then the virus could compute the new hash value after infection and replace the table entry. A virus cannot do that with a MAC because the virus does not know the key.

### About Generating and Storing Encryption Keys

The `DBMS_CRYPTO` package can generate random material for encryption keys, but it does not provide a mechanism for maintaining them. Application developers must take care to ensure that the encryption keys used with this package are securely generated and stored. Also note that the encryption and decryption operations performed by `DBMS_CRYPTO` occur on the server, not on the client. Consequently, if the key is sent over the connection between the client and the server, the connection must be protected by using network encryption. Otherwise, the key is vulnerable to capture over the wire.

Although DBMS\_CRYPT0 cannot generate keys on its own, it does provide tools you can use to aid in key generation. For example, you can use the RANDOMBYTES function to generate random material for keys. (Calls to the RANDOMBYTES function behave like calls to the DESGETKEY and DES3GETKEY functions of the DBMS\_OBFUSCATION\_TOOLKIT package.)

When generating encryption keys for DES, it is important to remember that some numbers are considered weak and semiweak keys. Keys are considered weak or semiweak when the pattern of the algorithm combines with the pattern of the initial key value to produce ciphertext that is more susceptible to cryptanalysis. To avoid this, filter out the known weak DES keys. Lists of the known weak and semiweak DES keys are available on several public Internet sites.

**See Also:**

- *Oracle Database Advanced Security Administrator's Guide* for information about configuring network encryption and SSL.
- ["Key Management"](#) on page 96-5 for a full discussion about securely storing encryption keys
- ["RANDOMBYTES Function"](#) on page 39-20

## Conversion Rules

- To convert VARCHAR2 to RAW, use the UTL\_I18N.STRING\_TO\_RAW function to perform the following steps:
  1. Convert VARCHAR2 in the current database character set to VARCHAR2 in the AL32UTF8 database character.
  2. Convert VARCHAR2 in the AL32UTF8 database character set to RAW.

Syntax example:

```
UTL_I18N.STRING_TO_RAW (string, 'AL32UTF8');
```

- To convert RAW to VARCHAR2, use the UTL\_I18N.RAW\_TO\_CHAR function to perform the following steps:
  1. Convert RAW to VARCHAR2 in the AL32UTF8 database character set.
  2. Convert VARCHAR2 in the AL32UTF8 database character set to VARCHAR2 in the database character set you wish to use.

Syntax example:

```
UTL_I18N.RAW_TO_CHAR (data, 'AL32UTF8');
```

**See Also:** [Chapter 225, "UTL\\_I18N"](#) for information about using the UTL\_I18N PL/SQL package.

- If you want to store encrypted data of the RAW datatype in a VARCHAR2 database column, then use RAWTOHEX or UTL\_ENCODE.BASE64\_ENCODE to make it suitable for VARCHAR2 storage. These functions expand data size by 2 and 4/3, respectively.

## Examples

The following listing shows PL/SQL block encrypting and decrypting pre-defined 'input\_string' using 256-bit AES algorithm with Cipher Block Chaining and PKCS#5 compliant padding.

```
DECLARE
    input_string      VARCHAR2 (200) := 'Secret Message';
    output_string     VARCHAR2 (200);
    encrypted_raw     RAW (2000);           -- stores encrypted binary text
    decrypted_raw     RAW (2000);           -- stores decrypted binary text
    num_key_bytes     NUMBER := 256/8;     -- key length 256 bits (32 bytes)
    key_bytes_raw     RAW (32);            -- stores 256-bit encryption key
    encryption_type   PLS_INTEGER :=      -- total encryption type
        DBMS_CRYPTO.ENCRYPT_AES256
        + DBMS_CRYPTO.CHAIN_CBC
        + DBMS_CRYPTO.PAD_PKCS5;
BEGIN
    DBMS_OUTPUT.PUT_LINE ( 'Original string: ' || input_string);
    key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
    encrypted_raw := DBMS_CRYPTO.ENCRYPT
    (
        src => UTL_I18N.STRING_TO_RAW (input_string, 'AL32UTF8'),
        typ => encryption_type,
        key => key_bytes_raw
    );
    -- The encrypted value "encrypted_raw" can be used here

    decrypted_raw := DBMS_CRYPTO.DECRYPT
    (
        src => encrypted_raw,
        typ => encryption_type,
        key => key_bytes_raw
    );
    output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');

    DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);
END;
```

---

## Summary of DBMS\_CRYPTO Subprograms

**Table 39–10 DBMS\_CRYPTO Package Subprograms**

Subprogram	Description
<a href="#">DECRYPT Function</a> on page 39-14	Decrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector)
<a href="#">DECRYPT Procedures</a> on page 39-15	Decrypts LOB data using a stream or block cipher with a user supplied key and optional IV
<a href="#">ENCRYPT Function</a> on page 39-16	Encrypts RAW data using a stream or block cipher with a user supplied key and optional IV
<a href="#">ENCRYPT Procedures</a> on page 39-17	Encrypts LOB data using a stream or block cipher with a user supplied key and optional IV
<a href="#">HASH Function</a> on page 39-18	Applies one of the supported cryptographic hash algorithms (MD4, MD5, or SHA-1) to data
<a href="#">MAC Function</a> on page 39-19	Applies Message Authentication Code algorithms (MD5 or SHA-1) to data to provide keyed message protection
<a href="#">RANDOMBYTES Function</a> on page 39-20	Returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, and can be used to generate random material for encryption keys
<a href="#">RANDOMINTEGER Function</a> on page 39-21	Returns a random BINARY_INTEGER
<a href="#">RANDOMNUMBER Function</a> on page 39-22	Returns a random 128-bit integer of the NUMBER datatype

## DECRYPT Function

This function decrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

### Syntax

```
DBMS_CRYPTO.DECRYPT (
  src IN RAW,
  typ IN PLS_INTEGER,
  key IN RAW,
  iv IN RAW DEFAULT NULL)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references (decrypt, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39–11 DECRYPT Function Parameters**

Parameter Name	Description
src	RAW data to be decrypted.
typ	Stream or block cipher type and modifiers to be used.
key	Key to be used for decryption.
iv	Optional initialization vector for block ciphers. Default is NULL.

### Usage Notes

- To retrieve original plaintext data, DECRYPT must be called with the same cipher, modifiers, key, and IV that was used to encrypt the data originally.
 

**See Also:** ["Usage Notes"](#) for the ENCRYPT function on page 39-16 for additional information about the ciphers and modifiers available with this package.
- If VARCHAR2 data is converted to RAW before encryption, then it must be converted back to the appropriate database character set by using the UTL\_I18N package.
 

**See Also:** ["Conversion Rules"](#) on page 39-11 for a discussion of the VARCHAR2 to RAW conversion process.

## DECRYPT Procedures

These procedures decrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

### Syntax

```
DBMS_CRYPTO.DECRYPT (
  dst IN OUT NOCOPY BLOB,
  src IN          BLOB,
  typ IN          PLS_INTEGER,
  key IN          RAW,
  iv IN           RAW          DEFAULT NULL);
```

```
DBMS_CRYPT.DECRYPT (
  dst IN OUT NOCOPY CLOB          CHARACTER SET ANY_CS,
  src IN          BLOB,
  typ IN          PLS_INTEGER,
  key IN          RAW,
  iv IN           RAW          DEFAULT NULL);
```

### Pragmas

```
pragma restrict_references (decrypt, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39–12** *DECRYPT Procedure Parameters*

Parameter Name	Description
dst	LOB locator of output data. The value in the output LOB <dst> will be overwritten.
src	LOB locator of input data.
typ	Stream or block cipher type and modifiers to be used.
key	Key to be used for decryption.
iv	Optional initialization vector for block ciphers. Default is all zeroes.

## ENCRYPT Function

This function encrypts RAW data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

### Syntax

```
DBMS_CRYPTO.ENCRYPT (
  src IN RAW,
  typ IN PLS_INTEGER,
  key IN RAW,
  iv IN RAW          DEFAULT NULL)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references (encrypt, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39–13 ENCRYPT Function Parameters**

Parameter Name	Description
src	RAW data to be encrypted.
typ	Stream or block cipher type and modifiers to be used.
key	Encryption key to be used for encrypting data.
iv	Optional initialization vector for block ciphers. Default is NULL.

### Usage Notes

- Block ciphers may be modified with chaining and padding type modifiers. The chaining and padding type modifiers are added to the block cipher to produce a cipher suite. Cipher Block Chaining (CBC) is the most commonly used chaining type, and PKCS #5 is the recommended padding type. See [Table 39–7](#) and [Table 39–8](#) on page 39-7 for block cipher chaining and padding modifier constants that have been defined for this package.

- To improve readability, you can define your own package-level constants to represent the cipher suites you use for encryption and decryption. For example, the following example defines a cipher suite that uses DES, cipher block chaining mode, and no padding:

```
DES_CBC_NONE CONSTANT PLS_INTEGER := DBMS_CRYPTO.ENCRYPT_DES
                                     + DBMS_CRYPTO.CHAIN_CBC
                                     + DBMS_CRYPTO.PAD_NONE;
```

See [Table 39–6](#) on page 39-6 for the block cipher suites already defined as constants for this package.

- To encrypt VARCHAR2 data, it should first be converted to the AL32UTF8 character set.

**See Also:** ["Conversion Rules"](#) on page 39-11 for a discussion of the conversion process.

- Stream ciphers, such as RC4, are not recommended for stored data encryption.



## ENCRYPT Procedures

These procedures encrypt LOB data using a stream or block cipher with a user supplied key and optional IV (initialization vector).

### Syntax

```
DBMS_CRYPT0.ENCRYPT (
  dst IN OUT NOCOPY BLOB,
  src IN          BLOB,
  typ IN          PLS_INTEGER,
  key IN          RAW,
  iv IN           RAW          DEFAULT NULL);
```

```
DBMS_CRYPT0.ENCRYPT (
  dst IN OUT NOCOPY BLOB,
  src IN          CLOB          CHARACTER SET ANY_CS,
  typ IN          PLS_INTEGER,
  key IN          RAW,
  iv IN           RAW          DEFAULT NULL);
```

### Pragmas

```
pragma restrict_references (encrypt, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39–14 ENCRYPT Procedure Parameters**

Parameter Name	Description
dst	LOB locator of output data. The value in the output LOB <dst> will be overwritten.
src	LOB locator of input data.
typ	Stream or block cipher type and modifiers to be used.
key	Encryption key to be used for encrypting data.
iv	Optional initialization vector for block ciphers. Default is NULL.

### Usage Notes

See "[Conversion Rules](#)" on page 39-11 for usage notes about using the ENCRYPT procedure.

## HASH Function

A one-way hash function takes a variable-length input string, the data, and converts it to a fixed-length (generally smaller) output string called a *hash value*. The hash value serves as a unique identifier (like a fingerprint) of the input data. You can use the hash value to verify whether data has been changed or not.

Note that a one-way hash function is a hash function that works in one direction. It is easy to compute a hash value from the input data, but it is hard to generate data that hashes to a particular value. Consequently, one-way hash functions work well to ensure data integrity. Refer to ["When to Use Hash or Message Authentication Code \(MAC\) Functions"](#) on page 39-10 for more information about using one-way hash functions.

This function applies to data one of the supported cryptographic hash algorithms listed in [Table 39-3](#) on page 39-6.

### Syntax

```
DBMS_CRYPTO.Hash (
    src IN RAW,
    typ IN PLS_INTEGER)
RETURN RAW;

DBMS_CRYPTO.Hash (
    src IN BLOB,
    typ IN PLS_INTEGER)
RETURN RAW;

DBMS_CRYPTO.Hash (
    src IN CLOB CHARACTER SET ANY_CS,
    typ IN PLS_INTEGER)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references (hash, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39-15 HASH Function Parameters**

Parameter Name	Description
src	The source data to be hashed.
typ	The hash algorithm to be used.

### Usage Note

Oracle recommends that you use the SHA-1 (Secure Hash Algorithm), specified with the constant, `HASH_SH1`, because it is more resistant to brute-force attacks than MD4 or MD5. If you must use a Message Digest algorithm, then MD5 provides greater security than MD4.

## MAC Function

A Message Authentication Code, or MAC, is a key-dependent one-way hash function. MACs have the same properties as the one-way hash function described in "[HASH Function](#)" on page 39-18, but they also include a key. Only someone with the identical key can verify the hash. Also refer to "[When to Use Hash or Message Authentication Code \(MAC\) Functions](#)" on page 39-10 for more information about using MACs.

This function applies MAC algorithms to data to provide keyed message protection. See [Table 39-4](#) on page 39-6 for a list of MAC algorithms that have been defined for this package.

### Syntax

```
DBMS_CRYPT0.MAC (
    src IN RAW,
    typ IN PLS_INTEGER,
    key IN RAW)
RETURN RAW;

DBMS_CRYPT0.MAC (
    src IN BLOB,
    typ IN PLS_INTEGER
    key IN RAW)
RETURN RAW;

DBMS_CRYPT0.MAC (
    src IN CLOB CHARACTER SET ANY_CS,
    typ IN PLS_INTEGER
    key IN RAW)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references (mac, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39-16 MAC Function Parameters**

Parameter Name	Description
src	Source data to which MAC algorithms are to be applied.
typ	MAC algorithm to be used.
key	Key to be used for MAC algorithm.

## RANDOMBYTES Function

This function returns a RAW value containing a cryptographically secure pseudo-random sequence of bytes, which can be used to generate random material for encryption keys. The RANDOMBYTES function is based on the RSA X9.31 PRNG (Pseudo-Random Number Generator).

### Syntax

```
DBMS_CRYPTO.RANDOMBYTES (  
    number_bytes IN POSITIVE)  
RETURN RAW;
```

### Pragmas

```
pragma restrict_references (randombytes, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 39–17** RANDOMBYTES Function Parameter

Parameter Name	Description
number_bytes	The number of pseudo-random bytes to be generated.

### Usage Note

The number\_bytes value should not exceed the maximum length of a RAW variable.

## RANDOMINTEGER Function

This function returns an integer in the complete range available for the Oracle `BINARY_INTEGER` datatype.

### Syntax

```
DBMS_CRYPT0.RANDOMINTEGER  
RETURN BINARY_INTEGER;
```

### Pragmas

```
pragma restrict_references (randominteger, WNDS, RNDS, WNPS, RNPS);
```

## RANDOMNUMBER Function

This function returns an integer in the Oracle NUMBER datatype in the range of  $[0..2^{128}-1]$ .

### Syntax

```
DBMS_CRYPTO.RANDOMNUMBER  
RETURN NUMBER;
```

### Pragmas

```
pragma restrict_references (randomnumber, WNDS, RNDS, WNPS, RNPS);
```

---

---

## DBMS\_CSX\_ADMIN

The DBMS\_CSX\_ADMIN package provides an interface to customize the setup when transporting a tablespace containing binary XML data.

The chapter contains the following topics:

- [Using DBMS\\_CSX\\_ADMIN](#)
  - Overview
  - Security Model
  - Constants
- [Summary of DBMS\\_CSX\\_ADMIN](#)

## Using DBMS\_CSX\_ADMIN

- [Overview](#)
- [Security Model](#)
- [Constants](#)



## Overview

This package can be used by DBAs to customize the setup when transporting a tablespace containing binary XML data. The use of the package is not required in order for a transportable tablespace job to run.

By default, all binary XML tables will use the default token table set, which will be replicated during transport on the target database. To avoid the cost of transporting a potentially large token table set, the DBA may opt for registering a new set of token tables for a given tablespace. The package provides routines for token table set registration and lookup.

## Security Model

Owned by XDB, the DBMS\_CSX\_ADMIN package must be created by SYS or XDB. The EXECUTE privilege is granted to SYS or XDB or DBA. Subprograms in this package are executed using the privileges of the current user.

## Constants

The DBMS\_CSX\_ADMIN package uses the constants shown in [Table 40-1](#):

**Table 40-1 DBMS\_CSX\_ADMIN Constants**

Name	Type	Value	Description
DEFAULT_LEVEL	BINARY_INTEGER	0	Default token table
TAB_LEVEL	BINARY_INTEGER	1	Token table set associated with tables, not tablespaces
TBS_LEVEL	BINARY_INTEGER	2	Token table set associated with a tablespace
NO_CREATE	BINARY_INTEGER	0	Token tables already exist, associate them with the given table/tablespace
NO_INDEXES	BINARY_INTEGER	1	Do not create indexes on the new set of token tables
WITH_INDEXES	BINARY_INTEGER	2	Create indexes on the token tables
DEFAULT_TOKS	BINARY_INTEGER	0	Prepopulate the token tables with default token mappings
NO_DEFAULT_TOKS	BINARY_INTEGER	1	Do not prepopulate the token tables with default token mappings

## Examples

### Example 1: Register a New Set of Token Tables for the Tablespace

To prepare a tablespace CSXTS for export, the DBA can register a new set of token tables for the tablespace with the following PL/SQL:

```
DECLARE
    tsno    number;
    stmt    varchar2(2000);
BEGIN
    stmt := 'SELECT ts# FROM ts$ WHERE (name = '' || 'CSXTS' || '' )';
    EXECUTE IMMEDIATE stmt INTO tsno;
    DBMS_CSX_ADMIN.REGISTER_TOKENTABLESET(
        tstabno => tsno,
        guid    => NULL,
        flags   => DBMS_CSX_ADMIN.TBS_LEVEL,
        tocreate => DBMS_CSX_ADMIN.WITH_INDEXES,
        defaulttoks => DBMS_CSX_ADMIN.DEFAULT_TOKS);
    COMMIT;
END;
/
```

In this example, the new token tables are indexed, populated with default token mappings, and associated with all tables in the CSXTS tablespace. All subsequent loads of binary XML data in CSXTS will make use of the new set of token tables. The advantage is that no loading of binary XML data in other tablespaces will affect the size of the token tables used by CSXTS and exported during a tablespace export of CSXTS. This setup is suited to the case in which the tablespace does not yet contain binary XML data.

### Example 2: Copying the Default Token Tables in a New Set

If binary XML data already exists in the tablespace to be exported, the DBA has the option of copying the default token tables in a new set.

```
DECLARE
    tsno    number;
    stmt    varchar2(2000);
    qntab   varchar2(34);
    nmtab   varchar2(34);
    pttab   varchar2(34);
BEGIN
    stmt := 'select ts# from ts$ where (name = '' || 'CSXTS' || '' )';
    EXECUTE IMMEDIATE stmt INTO tsno;
    DBMS_CSX_ADMIN.COPYDEFAULT_TOKENTABLESET(
        tstabno => tsno, qntab, nmtab, pttab);
    COMMIT;
END;
/
```

This setup is suited to the case in which the DBA wishes to optimize the export of a tablespace that already contains binary XML data, and does not have associated a token table set. After cloning the default token table set, all subsequent loads of binary XML data in CSXTS will make use of the new set of token tables.

---

## Summary of DBMS\_CSX\_ADMIN

**Table 40–2 DBMS\_CSX\_ADMIN Package Subprograms**

Subprogram	Description
<a href="#">COPYDEFAULTTOKENTABLESET Procedure</a> on page 40-8	Creates a new token table set associated with a given tablespace, and populates the token tables with the token mappings from the default token tables
<a href="#">GETTOKENTABLEINFO Procedure &amp; Function</a> on page 40-9	Returns the GUID of the token table set where token mappings for this table
<a href="#">GETTOKENTABLEINFOBYTABLESPACE Procedure</a> on page 40-10	Returns the GUID and the token table names for this tablespace
<a href="#">NAMESPACEIDTABLE Function</a> on page 40-11	Returns default namespace-ID token table
<a href="#">PATHIDTABLE Function</a> on page 40-12	Returns the default path-ID token table
<a href="#">QNAMEIDTABLE Function</a> on page 40-13	Returns the default qname-ID token table.
<a href="#">REGISTERTOKENTABLESET Procedure</a> on page 40-14	Registers a new token table set, creates (if required) the token tables (with the corresponding indexes)

## COPYDEFAULTTOKENTABLESET Procedure

This procedure creates a new token table set associated with a given tablespace, and populates the token tables with the token mappings from the default token tables.

### Syntax

```
DBMS_CSX_ADMIN.COPYDEFAULTTOKENTABLESET (
    tsno          IN    NUMBER,
    qnametable    OUT   VARCHAR2,
    nmspctable    OUT   VARCHAR2,
    pttable       OUT   VARCHAR2);
```

### Parameters

**Table 40-3** *COPYDEFAULTTOKENTABLESET Procedure Parameters*

Parameter	Description
tsno	Tablespace number the new set of token tables will be associated with
qnametable	Name of the qname-ID table in the new set
nmspctable	Name of the namespace-ID table in the new set
pttable	Name of the path-id table in the new set

## GETTOKENTABLEINFO Procedure & Function

Given the table name and the owner, the first overload of the procedure returns the globally unique identifier (GUID) of the token table set where token mappings for this table can be found. The procedure returns also the names of the token tables, and whether the token table set is the default one.

Given the object number of a table, the second overload of the procedure returns the GUID of the token table set used by the table, and whether this is the default token table set.

### Syntax

```
DBMS_CSX_ADMIN.GETTOKENTABLEINFO (
  ownername      IN  VARCHAR2,
  tablename      IN  VARCHAR2,
  guid           OUT RAW,
  qnametable     OUT  VARCHAR2,
  nmspctable    OUT  VARCHAR2,
  level         OUT  NUMBER,
  tabno         OUT  NUMBER);
```

```
DBMS_CSX_ADMIN.GETTOKENTABLEINFO (
  tabno         IN  NUMBER,
  guid         OUT  RAW);
RETURN BOOLEAN;
```

### Parameters

**Table 40–4** GETTOKENTABLEINFO Procedure & Function Parameters

Parameter	Description
ownername	Owner of the table
tablename	Name of the table
guid	GUID of the token table set used by the given table
qnametable	Name of the qname-ID table in the new set
nmspctable	Name of the namespace-ID table in the new set
level	DEFAULT_LEVEL if default token table set, TBS_LEVEL if same token table set is used by all tables in the same tablespace as the given table, TAB_LEVEL otherwise
tabno	Table object number

## GETTOKENTABLEINFOBYTABLESPACE Procedure

Given a tablespace number, this procedure returns the GUID and the token table names for this tablespace.

### Syntax

```
DBMS_CSX_ADMIN.GETTOKENTABLEINFOBYTABLESPACE (
    tsname          IN  VARCHAR2,
    tablespaceno    IN  NUMBER,
    guid            OUT RAW,
    qnametable      OUT VARCHAR2,
    nmspctable      OUT VARCHAR2,
    isdefault       OUT BOOLEAN,
    containTokTab   OUT BOOLEAN);
```

### Parameters

**Table 40–5** GETTOKENTABLEINFOBYTABLESPACE Procedure Parameters

Parameter	Description
tsname	Tablespace name
tablespaceno	Tablespace number
guid	GUID of the token table set associated with this tablespace (if any)
qnametable	Name of the qname-ID table
nmspctable	Name of the namespace-ID table
isdefault	TRUE if the token table is the default one
containTokTab	TRUE if the tablespace contains its own token table set



## **NAMESPACEIDTABLE Function**

This procedure returns default namespace-ID token table.

### **Syntax**

```
DBMS_CSX_ADMIN.NAMESPACEIDTABLE  
RETURN VARCHAR2;
```

## PATHIDTABLE Function

This procedure returns the default path-ID token table. This is used for granting permissions on the default path-ID token table for a user before executing `EXPLAIN PLAN` for a query on an XML table with an XML index.

### Syntax

```
DBMS_CSX_ADMIN.PATHIDTABLE  
RETURN VARCHAR2;
```

## QNAMEIDTABLE Function

This procedure returns the default qname-ID token table.

### Syntax

```
DBMS_CSX_ADMIN.QNAMEIDTABLE  
RETURN VARCHAR2;
```

## REGISTERTOKENTABLESET Procedure

This procedure registers a new token table set, creates (if required) the token tables (with the corresponding indexes).

### Syntax

```
DBMS_CSX_ADMIN.REGISTERTOKENTABLESET (
  tstabno      IN  NUMBER  DEFAULT NULL,
  guid         IN  RAW      DEFAULT NULL,
  flags        IN  NUMBER  DEFAULT TBS_LEVEL,
  tocreate     IN  NUMBER  DEFAULT WITH_INDEXES,
  defaulttoks IN  NUMBER  DEFAULT DEFAULT_TOKS);
```

### Parameters

**Table 40–6** *REGISTERTOKENTABLESET Procedure Parameters*

Parameter	Description
tstabno	Tablespace/table number of the tablespace/table using the set of token table we register
guid	GUID of the token table set. If NULL, a new identifier is created, provided the user is SYS.
flags	TAB_LEVEL for table level, TBS_LEVEL for tablespace level
tocreate	Possible values: <ul style="list-style-type: none"> <li>■ NO_CREATE if no token tables are created</li> <li>■ NO_INDEXES if token tables are created, but no indexes</li> <li>■ WITH_INDEXES if token tables and corresponding indexes are created</li> </ul>
defaulttoks	If DEFAULT_TOKS, insert default token mappings

DBMS\_CUBE contains subprograms that create OLAP cubes and dimensions, and that load and process the data for querying.

**See Also:** *Oracle OLAP User's Guide* regarding use of the OLAP option to support business intelligence and analytical applications

This chapter contains the following topics:

- [Using DBMS\\_CUBE](#)
- [Using SQL Aggregation Management](#)
- [Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g](#)
- [Summary of DBMS\\_CUBE Subprograms](#)

## Using DBMS\_CUBE

Cubes and cube dimensions are first class data objects that support multidimensional analytics. They are stored in a container called an analytic workspace. Multidimensional objects and analytics are available with the OLAP option to Oracle Database.

Cubes can be enabled as cube materialized views for automatic refresh of the cubes and dimensions, and for query rewrite. Several DBMS\_CUBE subprograms support the creation and maintenance of cube materialized views as a replacement for relational materialized views. These subprograms are discussed in ["Using SQL Aggregation Management"](#) on page 41-4.

The metadata for cubes and dimensions is defined in XML documents, called *templates*, which you can derive from relational materialized views using the CREATE\_CUBE or DERIVE\_FROM\_MVIEW functions. Using a graphical tool named Analytic Workspace Manager, you can enhance the cube with analytic content or create the metadata for new cubes and cube dimensions from scratch.

Several other DBMS\_CUBE subprograms provide a SQL alternative to Analytic Workspace Manager for creating an analytic workspace from an XML template and for refreshing the data stored in cubes and dimensions. The IMPORT\_XML procedure creates an analytic workspace with its cubes and cube dimensions from an XML template. The BUILD procedure loads data into the cubes and dimensions from their data sources and performs whatever processing steps are needed to prepare the data for querying.

## Security Model

The following roles and system privileges are required to use this package:

**To create dimensional objects in the user's own schema:**

- OLAP\_USER role
- CREATE SESSION privilege

**To create dimensional objects in different schemas:**

- OLAP\_DBA role
- CREATE SESSION privilege

**To create cube materialized views in the user's own schema:**

- CREATE MATERIALIZED VIEW privilege
- CREATE DIMENSION privilege
- ADVISOR privilege

**To create cube materialized views in different schemas:**

- CREATE ANY MATERIALIZED VIEW privilege
- CREATE ANY DIMENSION privilege
- ADVISOR privilege

If the source tables are in a different schema, then the owner of the dimensional objects needs SELECT object privileges on those tables.

## Using SQL Aggregation Management

SQL Aggregation Management is a group of PL/SQL subprograms in `DBMS_CUBE` that supports the rapid deployment of cube materialized views from existing relational materialized views. Cube materialized views are cubes that have been enhanced to use the automatic refresh and query rewrite features of Oracle Database. A single cube materialized view can replace many of the relational materialized views of summaries on a fact table, providing uniform response time to all summary data.

Cube materialized views bring the fast update and fast query capabilities of the OLAP option to applications that query summaries of detail relational tables. The summary data is generated and stored in a cube, and query rewrite automatically redirects queries to the cube materialized views. Applications experience excellent querying performance.

In the process of creating the cube materialized views, `DBMS_CUBE` also creates a fully functional analytic workspace including a cube and the cube dimensions. The cube stores the data for a cube materialized view instead of the table that stores the data for a relational materialized view. A cube can also support a wide range of analytic functions that enhance the database with information-rich content.

Cube materialized views are registered in the data dictionary along with all other materialized views. A `CB$` prefix identifies a cube materialized view.

The `DBMS_CUBE` subprograms also support life-cycle management of cube materialized views.

**See Also:** *Oracle OLAP User's Guide* for more information about cube materialized views and enhanced OLAP analytics.



## Subprograms in SQL Aggregation Management

These subprograms are included in SQL Aggregation Management:

- [CREATE\\_MVIEW](#) Function
- [DERIVE\\_FROM\\_MVIEW](#) Function
- [DROP\\_MVIEW](#) Procedure
- [REFRESH\\_MVIEW](#) Procedure

## Requirements for the Relational Materialized View

SQL Aggregation Management uses an existing relational materialized view to derive all the information needed to generate a cube materialized view. The relational materialized view determines the detail level of data that is stored in the cube materialized view. The related relational dimension objects determine the scope of the aggregates, from the lowest level specified in the `GROUP BY` clause of the materialized view subquery, to the highest level of the dimension hierarchy.

The relational materialized view must conform to these requirements:

- Explicit `GROUP BY` clause for one or more columns.
  - No expressions in the select list or `GROUP BY` clause.
  - At least one of these numeric aggregation methods: `SUM`, `MIN`, `MAX`, or `AVG`.
  - No outer joins.
  - Summary keys with at least one simple column associated with a relational dimension.
- or*
- Summary keys with at least one simple column and no hierarchies or levels.
- Numeric data type of any type for the fact columns. All facts are converted to `NUMBER`.
  - Eligible for rewrite. `REWRITE_CAPABILITY` should be `GENERAL`; it cannot be `NONE`. Refer to the `ALL_MVIEWS` entry in the *Oracle Database Reference*.
  - Cannot use the `DISTINCT` or `UNIQUE` keywords with an aggregate function in the defining query. For example, `AVG (DISTINCT units)` causes an error in `STRICT` mode and is ignored in `LOOSE` mode.

You can choose between two modes when rendering the cube materialized view, `LOOSE` and `STRICT`. In `STRICT` mode, any deviation from the requirements raises an exception and prevents the materialized view from being created. In `LOOSE` mode (the default), some deviations are allowed, but they affect the content of the materialized view. These elements in the relational materialized view generate warning messages:

- Complex expressions in the defining query are ignored and do not appear in the cube materialized view.
- The `AVG` function is changed to `SUM` and `COUNT`.
- The `COUNT` function without a `SUM`, `MIN`, `MAX`, or `AVG` function is ignored.
- The `STDDEV` and `VARIANCE` functions are ignored.

You can also choose how conditions in the `WHERE` clause are filtered. When filtering is turned off, the conditions are ignored. When turned on, valid conditions are rendered in the cube materialized view, but asymmetric conditions among dimension levels raise an exception.

## Permissions for Managing and Querying Cube Materialized Views

To create cube materialized views, you must have these privileges:

- CREATE [ANY] MATERIALIZED VIEW privilege
- CREATE [ANY] DIMENSION privilege
- ADVISOR privilege

To access cube materialized views from another schema using query rewrite, you must have these privileges:

- GLOBAL QUERY REWRITE privilege
- SELECT privilege on the relational source tables
- SELECT privilege on the analytic workspace (*AW\$name*) that supports the cube materialized view
- SELECT privilege on the cube
- SELECT privilege on the dimensions of the cube

Note that you need SELECT privileges on the database objects that *support* the cube materialized views, but not on the cube materialized views.

## Example of SQL Aggregation Management

All examples for the SQL Aggregate Management subprograms use the sample Sales History schema, which is installed in Oracle Database with two relational materialized views: CAL\_MONTH\_SALES\_MV and FWEEK\_PSCAT\_SALES\_MV.

### About Relational Materialized View CAL\_MONTH\_SALES\_MV

This example uses CAL\_MONTH\_SALES\_MV as the basis for creating a cube materialized view. The following query was used to create CAL\_MONTH\_SALES\_MV. CAL\_MONTH\_SALES\_MV summarizes the daily sales data stored in the SALES table by month.

```
SELECT query FROM user_mviews
      WHERE mview_name='CAL_MONTH_SALES_MV';
```

```
QUERY
```

```
-----
SELECT  t.calendar_month_desc
      ,   sum(s.amount_sold) AS dollars
FROM    sales s
      ,   times t
WHERE   s.time_id = t.time_id
GROUP BY t.calendar_month_desc
```

DBMS\_CUBE uses relational dimensions to derive levels and hierarchies for the cube materialized view. The SH schema has relational dimensions for most dimension tables in the schema, as shown by the following query.

```
SELECT dimension_name FROM user_dimensions;
```

```
DIMENSION_NAME
```

```
-----
CUSTOMERS_DIM
PRODUCTS_DIM
TIMES_DIM
CHANNELS_DIM
PROMOTIONS_DIM
```

### Creating the Cube Materialized View

This PL/SQL script uses the CREATE\_MVIEW function to create a cube materialized view from CAL\_MONTH\_SALES\_MV. CREATE\_MVIEW sets the optional BUILD parameter to refresh the cube materialized view immediately.

```
SET serverout ON format wrapped
```

```
DECLARE
```

```
    salesaw varchar2(30);
```

```
BEGIN
```

```
    salesaw := dbms_cube.create_mview('SH', 'CAL_MONTH_SALES_MV',
                                     'build=immediate');
```

```
END;
```

```
/
```

These messages confirm that the script created and refreshed CB\$CAL\_MONTH\_SALES successfully:

```
Completed refresh of cube mview "SH"."CB$CAL_MONTH_SALES" at 20081112 08:42:58.0
```

03.  
 Created cube organized materialized view "CB\$CAL\_MONTH\_SALES" for rewrite at 20081112 08:42:58.004.

The following query lists the materialized views in the SH schema:

```
SELECT mview_name FROM user_mvviews;
```

```
MVIEW_NAME
-----
CB$CAL_MONTH_SALES
CB$TIMES_DIM_D1_CAL_ROLLUP
CAL_MONTH_SALES_MV
FWEEK_PSCAT_SALES_MV
```

Two new materialized views are registered in the data dictionary:

- CB\$CAL\_MONTH\_SALES: Cube materialized view
- CB\$TIMES\_DIM\_D1\_CAL\_ROLLUP: Cube dimension materialized view for the TIME\_DIM Calendar Rollup hierarchy

Cube dimension materialized views support refresh of the cube materialized view. You do not directly administer dimension materialized views.

## Disabling the Relational Materialized Views

After creating a cube materialized view, disable query rewrite on all relational materialized views for the facts now supported by the cube materialized view. You can drop them when you are sure that you created the cube materialized view with the optimal parameters.

```
ALTER MATERIALIZED VIEW cal_month_sales_mv DISABLE QUERY REWRITE;
```

Materialized view altered.

You can also use the `DISABLEQRW` parameter in the `CREATE_MVIEW` function, which disables query rewrite on the source materialized view as described in [Table 41-7](#).

## Creating Execution Plans for Cube Materialized Views

You can create execution plans for cube materialized views the same as for relational materialized views. The following command generates an execution plan for a query against the `SALES` table, which contains data at the day level. The answer set requires data summarized by quarter. Query rewrite would not use the original relational materialized view for this query, because its data is summarized by month. However, query rewrite can use the new cube materialized view for summary data for months, quarters, years, and all years.

```
EXPLAIN PLAN FOR SELECT
    t.calendar_quarter_desc,
    sum(s.amount_sold) AS dollars
FROM    sales s,
        times t
WHERE   s.time_id = t.time_id
AND     t.calendar_quarter_desc LIKE '2001%'
GROUP BY t.calendar_quarter_desc
ORDER BY t.calendar_quarter_desc;
```

The query returns these results:

```
CALENDAR_QUARTER_DESC    DOLLARS
```

```

-----
2001-01                6547097.44
2001-02                6922468.39
2001-03                7195998.63
2001-04                7470897.52

```

The execution plan shows that query rewrite returned the summary data from the cube materialized view, CB\$CAL\_MONTH\_SALES, instead of recalculating it from the SALES table.

```
SELECT plan_table_output FROM TABLE(dbms_xplan.display());
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 2999729407
```

```

-----
| Id | Operation                               | Name                | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT                         |                     |     1 |    30 |    3 (34) | 00:00:01 |
|  1 | SORT GROUP BY                             |                     |     1 |    30 |    3 (34) | 00:00:01 |
|*  2 |  MAT_VIEW REWRITE CUBE ACCESS            | CB$CAL_MONTH_SALES |     1 |    30 |    2 (0)  | 00:00:01 |
-----

```

```
Predicate Information (identified by operation id):
```

```

-----
2 - filter("CB$CAL_MONTH_SALES"."D1_CALENDAR_QUARTER_DESC" LIKE '2001%' AND
          "CB$CAL_MONTH_SALES"."SYS_GID"=63)

```

```
15 rows selected.
```

## Maintaining Cube Materialized Views

You can create a cube materialized view that refreshes automatically. However, you can force a refresh at any time using the [REFRESH\\_MVIEW Procedure](#):

```

BEGIN
    dbms_cube.refresh_mview('SH', 'CB$CAL_MONTH_SALES');
END;
/

```

```
Completed refresh of cube mview "SH"."CB$CAL_MONTH_SALES" at 20081112
14:30:59.534.
```

If you want to drop a cube materialized view, use the [DROP\\_MVIEW Procedure](#) so that all supporting database objects (analytic workspace, cube, cube dimensions, and so forth) are also dropped:

```

BEGIN
    dbms_cube.drop_mview('SH', 'CB$CAL_MONTH_SALES');
END;
/

```

```
Dropped cube organized materialized view "SH"."CAL_MONTH_SALES" including
container analytic workspace "SH"."CAL_MONTH_SALES_AW" at 20081112 13:38:47.878.
```

## New Database Objects

The CREATE\_MVIEW function creates several first class database objects in addition to the cube materialized views. You can explore these objects through the data dictionary by querying views such as ALL\_CUBES and ALL\_CUBE\_DIMENSIONS.

This example created the following supporting objects:

- Analytic workspace CAL\_MONTH\_SALES\_AW (AW\$CAL\_MONTH\_SALES\_AW table)
- Cube CAL\_MONTH\_SALES
- Cube dimension TIMES\_DIM\_D1
- Dimension hierarchy CAL\_ROLLUP
- Dimension levels ALL\_TIMES\_DIM, YEAR, QUARTER, and MONTH
- Numerous attributes for levels in the CAL\_ROLLUP hierarchy

## Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g

You can upgrade an Oracle OLAP 10g analytic workspace to OLAP 11g by saving the metadata in an XML template and using it to create a new analytic workspace. The original analytic workspace remains accessible and unchanged by the upgrade process.

**Tip:** Oracle recommends using Analytic Workspace Manager for performing upgrades. See the *Oracle OLAP User's Guide*.

These subprograms in DBMS\_CUBE support the upgrade process:

- [CREATE\\_EXPORT\\_OPTIONS Procedure](#)
- [CREATE\\_IMPORT\\_OPTIONS Procedure](#)
- [EXPORT\\_XML Procedure](#)
- [EXPORT\\_XML\\_TO\\_FILE Procedure](#)
- [IMPORT\\_XML Procedure](#)
- [INITIALIZE\\_CUBE\\_UPGRADE Procedure](#)
- [UPGRADE\\_AW Procedure](#)

### Prerequisites:

- The OLAP 10g analytic workspace can use CWM metadata or OLAP standard form (AWXML) metadata.
- Customizations to the OLAP 10g analytic workspace may not be exported to the XML template. You must re-create them in OLAP 11g.
- The original relational source data must be available to load into the new analytic workspace. If the data is in a different schema or the table names are different, then you must remap the dimensional objects to the new relational sources after the upgrade.
- You can create the OLAP 11g analytic workspace in the same schema as the OLAP 10g analytic workspace. However, if you prefer to create it in a different schema, then create a new user with the following privileges:
  - SELECT privileges on the OLAP 10g analytic workspace (`GRANT SELECT ON schema.AW$analytic_workspace`).
  - SELECT privileges on all database tables and views that contain the source data for the OLAP 10g analytic workspace.
  - Appropriate privileges for an OLAP administrator.
  - Same default tablespace as the Oracle 10g user.

See the *Oracle OLAP User's Guide*.



## Correcting Naming Conflicts

The namespaces are different in OLAP 11g and OLAP 10g. For a successful upgrade, you must identify any 10g object names that are used multiple times under the 11g naming rules and provide unique names for them.

The following namespaces control the uniqueness of OLAP object names in Oracle 11g:

- **Schema:** The names of cubes, dimensions, and measure folders must be unique within a schema. They cannot conflict with the names of tables, views, indexes, relational dimensions, or any other first class objects. However, these OLAP 11g object names do not need to be distinct from 10g object names, because they are in different namespaces.
- **Cube:** The names of measures must be unique within a cube.
- **Dimension:** The names of hierarchies, levels, and attributes must be unique within a dimension. For example, a dimension cannot have a hierarchy named Customers and a level named Customers.

You can use an initialization table and a rename table to rename objects in the upgraded 11g analytic workspace.

### Initialization Table

The `INITIALIZE_CUBE_UPGRADE` procedure identifies ambiguous names under the OLAP 11g naming rules. For example, a 10g dimension might have a hierarchy and a level with the same name. Because hierarchies and levels are in the same 11g namespace, the name is not unique in 11g; to an 11g client, the hierarchy and the level cannot be differentiated by name.

`INITIALIZE_CUBE_UPGRADE` creates and populates a table named `CUBE_UPGRADE_INFO` with unique names for these levels, hierarchies, and attributes. By using the unique names provided in the table, an 11g client can browse the OLAP 11g metadata. You cannot attach an OLAP 11g client to the analytic workspace or perform an upgrade without a `CUBE_UPGRADE_INFO` table, if the 10g metadata contains ambiguous names.

You can edit `CUBE_UPGRADE_INFO` to change the default unique names to names of your choosing. You can also add rows to change the names of any other objects. When using an 11g client, you see the new object names. When using an 10g client, you see the original names. However, the `INITIALIZE_CUBE_UPGRADE` procedure overwrites this table, so you may prefer to enter customizations in a rename table.

During an upgrade from OLAP 10g, the unique object names in `CUBE_UPGRADE_INFO` are used as the names of 11g objects in the new analytic workspace. However, `INITIALIZE_CUBE_UPGRADE` does not automatically provide unique names for cubes, dimensions, and measure folders. To complete an upgrade, you must assure that these objects have unique names within the 11g namespace. You can provide these objects with new names in the `CUBE_UPGRADE_INFO` table or in a rename table.

OLAP 11g clients automatically use `CUBE_UPGRADE_INFO` when it exists in the same schema as the OLAP 10g analytic workspace.

**See Also:** ["INITIALIZE\\_CUBE\\_UPGRADE Procedure"](#) on page 41-52

## Rename Table

You can create a rename table that contains new object names for an OLAP 11g analytic workspace. You can then use the rename table in the `CREATE_IMPORT_OPTIONS` and `UPGRADE_AW` procedures.

When upgrading within the same schema, you must provide a unique name for the 11g analytic workspace. The `UPGRADE_AW` procedure provides a parameter for this purpose; otherwise, you must provide the new name in the rename table. The duplication of cube names does not create ambiguity because the 11g cubes are created in a different namespace than the 10g cubes.

The names provided in a rename table are used only during an upgrade and overwrite any names entered in the `CUBE_UPGRADE_INFO` table.

### To create a rename table:

1. Open SQL\*Plus or another SQL client, and connect to Oracle Database as the owner of the 10g analytic workspace.
2. Issue a command like the following:

```
CREATE TABLE table_name (
    source_id    VARCHAR2(300),
    new_name     VARCHAR2(30),
    object_type  VARCHAR2(30));
```

3. Populate the rename table with the appropriate values, as follows.

`table_name` is the name of the rename table.

`source_id` is the identifier for an object described in the XML document supplied to `IMPORT_XML`. The identifier must have this format:

```
schema_name.object_name[.subobject_name]
```

`new_name` is the object name given during the import to the object specified by `source_id`.

`object_type` is the object type as described in the XML, such as `StandardDimension` or `DerivedMeasure`.

For example, these SQL statements populate the table with new names for the analytic workspace, a cube, and four dimensions:

```
INSERT INTO my_object_map VALUES('GLOBAL_AW.GLOBAL10.AW', 'GLOBAL11', 'AW');
INSERT INTO my_object_map VALUES('GLOBAL_AW.UNITS_CUBE', 'UNIT_SALES_CUBE',
'Cube');
INSERT INTO my_object_map VALUES('GLOBAL_AW.CUSTOMER', 'CUSTOMERS',
'StandardDimension');
INSERT INTO my_object_map VALUES('GLOBAL_AW.CHANNEL', 'CHANNELS',
'StandardDimension');
INSERT INTO my_object_map VALUES('GLOBAL_AW.PRODUCT', 'PRODUCTS',
'StandardDimension');
INSERT INTO my_object_map VALUES('GLOBAL_AW.TIME', 'TIME_PERIODS',
'TimeDimension');
```

**See Also:** ["CREATE\\_IMPORT\\_OPTIONS Procedure"](#) on page 41-34

## Simple Upgrade

A simple upgrade creates an OLAP 11g analytic workspace from an OLAP 10g analytic workspace.

### To perform a simple upgrade of an Oracle OLAP 10g analytic workspace:

1. Open SQL\*Plus or a similar SQL command-line interface and connect to Oracle Database 11g as the schema owner for the OLAP 11g analytic workspace.
2. To rename any objects in the 11g analytic workspace, create a rename table as described in ["Rename Table"](#) on page 41-14. (Optional)
3. Perform the upgrade, as described in ["UPGRADE\\_AW Procedure"](#) on page 41-56.
4. Use the `DBMS_CUBE.BUILD` procedure to load data into the cube.

### **Example 41–1 Performing a Simple Upgrade to the GLOBAL Analytic Workspace**

This example creates an OLAP 11g analytic workspace named `GLOBAL11` from an OLAP 10g analytic workspace named `GLOBAL10`. `GLOBAL10` contains no naming conflicts between cubes, dimensions, measure folders, or tables in the schema, so a rename table is not needed in this example.

```
BEGIN

  -- Upgrade the analytic workspace
  dbms_cube.upgrade_aw(sourceaw =>'GLOBAL10', destaw => 'GLOBAL11');

  -- Load and aggregate the data
  dbms_cube.build(script => 'UNITS_CUBE, PRICE_AND_COST_CUBE');

END;
/
```

## Custom Upgrade

A custom upgrade enables you to set the export and import options.

### To perform a custom upgrade of an Oracle OLAP 10g analytic workspace:

1. Open SQL\*Plus or a similar SQL command-line interface and connect to Oracle Database 11g as the schema owner of the OLAP 11g analytic workspace.
2. Generate an initialization table, as described in ["Initialization Table"](#) on page 41-13. Review the new, default object names and modify them as desired.
3. Create a rename table, as described in ["Rename Table"](#) on page 41-14. If you are upgrading in the same schema, you must use a rename table to provide a unique name for the 11g analytic workspace. Otherwise, a rename table is needed only if names are duplicated among the cubes, dimensions, and measure folders of the analytic workspace, or between those names and the existing cubes, dimensions, measure folders, or tables of the destination schema.
4. Create a SQL script that does the following:
  - a. Create an XML document for the export options, as described in ["CREATE\\_EXPORT\\_OPTIONS Procedure"](#) on page 41-31. The `SUPPRESS_NAMESPACE` option must be set to `TRUE` for the upgrade to occur.
  - b. Create an XML document for the import options, as described in ["CREATE\\_IMPORT\\_OPTIONS Procedure"](#) on page 41-34.
  - c. Create an XML template in OLAP 11g format, as described in ["EXPORT\\_XML Procedure"](#) on page 41-45.
  - d. Create an OLAP 11g analytic workspace from the XML template, as described in ["IMPORT\\_XML Procedure"](#) on page 41-49.
5. Load and aggregate the data in the new analytic workspace, as described in ["BUILD Procedure"](#) on page 41-19.

### **Example 41–2 Performing a Custom Upgrade to the GLOBAL Analytic Workspace**

This example upgrades the `GLOBAL10` analytic workspace from OLAP 10g metadata to OLAP 11g metadata in the `GLOBAL_AW` schema.

The rename table provides the new name of the analytic workspace. These commands define the rename table.

```
CREATE TABLE my_object_map(
    source_id    VARCHAR2(300),
    new_name     VARCHAR2(30),
    object_type  VARCHAR2(30));

INSERT INTO my_object_map VALUES('GLOBAL_AW.GLOBAL10.AW', 'GLOBAL11', 'AW');
COMMIT;
```

Following is the script for performing the upgrade.

```
set serverout on

DECLARE
    importClob    clob;
    exportClob    clob;
    exportOptClob clob;
    importOptClob clob;
```

```
BEGIN

  -- Create table of reconciled names
  dbms_cube.initialize_cube_upgrade;

  -- Create a CLOB containing the export options
  dbms_lob.createtemporary(exportOptClob, TRUE);
  dbms_cube.create_export_options(out_options_xml=>exportOptClob, suppress_
namespace=>TRUE, preserve_table_owners=>TRUE);

  -- Create a CLOB containing the import options
  dbms_lob.createtemporary(importOptClob, TRUE);
  dbms_cube.create_import_options(out_options_xml=>importOptClob, rename_table =>
'MY_OBJECT_MAP');

  -- Create CLOBs for the metadata
  dbms_lob.createtemporary(importClob, TRUE);
  dbms_lob.createtemporary(exportClob, TRUE);

  -- Export metadata from a 10g analytic workspace to a CLOB
  dbms_cube.export_xml(object_ids=>'GLOBAL_AW', options_xml=>exportOptClob, out_
xml=>exportClob);

  -- Import metadata from the CLOB
  dbms_cube.import_xml(in_xml => exportClob, options_xml=>importOptClob, out_
xml=>importClob);

  -- Load and aggregate the data
  dbms_cube.build('UNITS_CUBE, PRICE_AND_COST_CUBE');

END;
/
```

---

## Summary of DBMS\_CUBE Subprograms

**Table 41–1 DBMS\_CUBE Subprograms**

Subprogram	Description
<a href="#">BUILD Procedure</a> on page 41-19	Loads data into one or more cubes and dimensions, and prepares the data for querying.
<a href="#">CREATE_EXPORT_OPTIONS Procedure</a> on page 41-31	Creates an input XML document of processing options for the EXPORT_XML procedure.
<a href="#">CREATE_IMPORT_OPTIONS Procedure</a> on page 41-34	Creates an input XML document of processing options for the IMPORT_XML procedure.
<a href="#">CREATE_MVIEW Function</a> on page 41-36	Creates a cube materialized view from the definition of a relational materialized view.
<a href="#">DERIVE_FROM_MVIEW Function</a> on page 41-41	Creates an XML template for a cube materialized view from the definition of a relational materialized view.
<a href="#">DROP_MVIEW Procedure</a> on page 41-43	Drops a cube materialized view.
<a href="#">EXPORT_XML Procedure</a> on page 41-45	Exports the XML of an analytic workspace to a CLOB.
<a href="#">EXPORT_XML_TO_FILE Procedure</a> on page 41-47	Exports the XML of an analytic workspace to a file.
<a href="#">IMPORT_XML Procedure</a> on page 41-49	Creates, modifies, or drops an analytic workspace by using an XML template
<a href="#">INITIALIZE_CUBE_UPGRADE Procedure</a> on page 41-52	Processes Oracle OLAP 10g objects with naming conflicts to enable Oracle 11g clients to access them.
<a href="#">REFRESH_MVIEW Procedure</a> on page 41-54	Refreshes a cube materialized view.
<a href="#">UPGRADE_AW Procedure</a> on page 41-56	Upgrades an analytic workspace from Oracle OLAP 10g to 11g.
<a href="#">VALIDATE_XML Procedure</a> on page 41-58	Checks the XML to assure that it is valid, without committing the results to the database.

## BUILD Procedure

This procedure loads data into one or more cubes and dimensions, and generates aggregate values in the cubes. The results are automatically committed to the database.

### Syntax

```
DBMS_CUBE.BUILD (
    script           IN  VARCHAR2,
    method          IN  VARCHAR2          DEFAULT NULL,
    refresh_after_errors IN  BOOLEAN      DEFAULT FALSE,
    parallelism     IN  BINARY_INTEGER  DEFAULT 0,
    atomic_refresh  IN  BOOLEAN          DEFAULT FALSE,
    automatic_order IN  BOOLEAN          DEFAULT TRUE,
    add_dimensions  IN  BOOLEAN          DEFAULT TRUE,
    scheduler_job   IN  VARCHAR2        DEFAULT NULL,
    master_build_id IN  BINARY_INTEGER  DEFAULT 0,
    nested          IN  BOOLEAN          DEFAULT FALSE);
```

### Parameters

**Table 41–2 BUILD Procedure Parameters**

Parameter	Description
script	A list of cubes and dimensions and their build options (see <a href="#">"SCRIPT Parameter"</a> on page 41-20).
method	<p>A full or a fast (partial) refresh. In a fast refresh, only changed rows are inserted in the cube and the affected areas of the cube are re-aggregated.</p> <p>You can specify a method for each cube and dimension in sequential order, or a single method to apply to all cubes and dimensions. If you list more objects than methods, then the last method applies to the additional objects.</p> <ul style="list-style-type: none"> <li>■ C: Complete refresh clears all dimension values before loading. (Default)</li> <li>■ F: Fast refresh of a cube materialized view, which performs an incremental refresh and re-aggregation of only changed rows in the source table.</li> <li>■ ?: Fast refresh if possible, and otherwise a complete refresh.</li> <li>■ P: Recomputes rows in a cube materialized view that are affected by changed partitions in the detail tables.</li> <li>■ S: Fast solve of a compressed cube. A fast solve reloads all the detail data and re-aggregates only the changed values.</li> </ul> <p>See the <a href="#">"Usage Notes"</a> on page 41-26 for additional details.</p> <p>Methods do not apply to dimensions.</p>
refresh_after_errors	<p>TRUE to roll back just the cube or dimension with errors, and then continue building the other objects.</p> <p>FALSE to roll back all objects in the build.</p>
parallelism	Number of parallel processes to allocate to this job (see <a href="#">Usage Notes</a> ).

**Table 41–2 (Cont.) BUILD Procedure Parameters**

Parameter	Description
<code>atomic_refresh</code>	<p><code>TRUE</code> prevents users from accessing intermediate results during a build. It freezes the current state of an analytic workspace at the beginning of the build to provide current sessions with consistent data. This option thaws the analytic workspace at the end of the build to give new sessions access to the refreshed data. If an error occurs during the build, then all objects are rolled back to the frozen state.</p> <p><code>FALSE</code> enables users to access intermediate results during a build.</p>
<code>automatic_order</code>	<p><code>TRUE</code> enables optimization of the build order. Dimensions are loaded before cubes.</p> <p><code>FALSE</code> builds objects in the order you list them in the script.</p>
<code>add_dimensions</code>	<p><code>TRUE</code> automatically includes all the dimensions of the cubes in the build, whether or not you list them in the script. If a cube materialized view with a particular dimension is fresh, then that dimension is not reloaded. You can list a cube once in the script.</p> <p><code>FALSE</code> includes only dimensions specifically listed in the script.</p>
<code>scheduler_job</code>	Any text identifier for the job, which will appear in the log table. The string does not need to be unique.
<code>master_build_id</code>	A unique name for the build.
<code>nested</code>	<p><code>TRUE</code> performs nested refresh operations for the specified set of cube materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.</p> <p>All objects must reside in a single analytic workspace.</p>

## SCRIPT Parameter

The `SCRIPT` parameter identifies the objects that will be included in the build, and specifies the type of processing that will be performed on each one. The parameter has this syntax:

```
[VALIDATE | NO COMMIT] objects [ USING ( commands ) ][,...]
```

*Where:*

`VALIDATE` checks all steps of the build and sends the planned steps to `CUBE_BUILD_LOG` without executing the steps. You can view all generated SQL in the `OUTPUT` column of the log table.

`NO COMMIT` builds the objects in the current attach mode (or Read Only when the analytic workspace is not attached) but does not commit the changes. This option supports what-if analysis, since it enables you to change data values temporarily. See ["SCRIPT Parameter: USING Clause: SET command"](#) on page 41-25.

`objects` is the qualified name of one or more cubes or dimensions, separated by commas, in the form `[aw_name. ]object`, such as `UNITS_CUBE` or `GLOBAL.UNITS_CUBE`.

## SCRIPT Parameter: USING Clause

The `USING` clause specifies the processing options. It consists of one or more commands separated by commas.



---



---

**Note:** A cube with a rewrite materialized view cannot have a USING clause, except for the ANALYZE command. It uses the default build options.

---



---

The commands can be any of the following.

- AGGREGATE USING [MEASURE]

Generates aggregate values using the syntax described in "[SCRIPT Parameter: USING Clause: AGGREGATE command](#)".

- ANALYZE

Runs DBMS\_AW\_STATS .ANALYZE, which generates and stores optimizer statistics for cubes and dimensions.

- CLEAR [VALUES | LEAVES | AGGREGATES] [SERIAL | PARALLEL]

Prepares the cube for a data refresh. It can also be used on dimensions, but CLEAR removes all dimension keys, and thus deletes all data values for cubes that use the dimension.

These optional arguments control the refresh method. If you omit the argument, then the behavior of CLEAR depends on the refresh method. The 'C' (complete) refresh method runs CLEAR VALUES, and all other refresh methods run CLEAR LEAVES.

- VALUES: Clears all data in the cube. All facts must be reloaded and all aggregates must be recomputed. This option supports the COMPLETE refresh method. (Default for the C and F methods)
- LEAVES: Clears the detail data and retains the aggregates. All facts must be reloaded, and the aggregates for any new or changed facts must be computed. This option supports the FAST refresh method. (Default for the ? method)
- AGGREGATES: Retains the detail data and clears the aggregates. All aggregates must be recomputed.

These optional arguments control the load method, and can be combined with any of the refresh options:

- PARALLEL: Each partition is cleared separately. (Default)
- SERIAL: All partitions are cleared together.

If you omit the CLEAR command, DBMS\_CUBE loads new and updated facts, but does not delete any old detail data. This is equivalent to a LOAD NO SYNC for dimensions.

- COMPILE [SORT | NO SORT | SORT ONLY]

Creates the supporting structures for the dimension. (Dimensions only)

These options control the use of a sort order attribute:

- SORT: The user-defined sort order attribute populates the sort column in the embedded-total (ET) view. (Default)
- NO SORT: Any sort order attribute is ignored. This option is for very large dimensions where sorting could consume too many resources.
- SORT ONLY: The compile step only runs the sort.

- EXECUTE PLSQL string

Executes a PL/SQL command or script in the database.

- EXECUTE OLAP DML *string* [PARALLEL | SERIAL]

Executes an OLAP DML command or program in the analytic workspace. The options control execution of the command or program:

- PARALLEL: Execute the command or program once for each partition. This option can be used to provide a performance boost to complex DML operations, such as forecasts and models.
- SERIAL: Execute the command or program once for the entire cube. (Default)

- [INSERT | MERGE] INTO [ALL HIERARCHIES | HIERARCHIES (*dimension.hierarchy*)] VALUES (*dim\_key*, *level\_name*, *parent*)

Adds a dimension member to one or more hierarchies. INSERT throws an error if the member already exists, while MERGE does not. See "[Dimension Maintenance Example](#)" on page 41-29.

*dimension.hierarchy*: The name of a hierarchy the new member belongs to. Enclose each part of the name in double quotes, for example, "PRODUCT" . "PRIMARY".

*dim\_key*: The DIM\_KEY value of the dimension member.

*level\_name*: The level of the dimension key.

*parent*: The parent of the dimension key.

- UPDATE [ALL HIERARCHIES | HIERARCHIES (*dimension.hierarchy*)] SET PARENT = *parent*, LEVEL=*level\_name* WHERE MEMBER = *dim\_key*

Alters the level or parent of an existing dimension member. See INSERT for a description of the options. Also see "[Dimension Maintenance Example](#)" on page 41-29.

- DELETE FROM DIMENSION WHERE MEMBER=*dim\_key*

Deletes a dimension member. See "[Dimension Maintenance Example](#)" on page 41-29.

*dim\_key*: The DIM\_KEY value of the dimension member to be deleted.

- SET *dimension.attribute[qdr]* = CAST('attribute\_value' AS VARCHAR2)

Sets the value of an attribute for a dimension member. See "[Dimension Maintenance Example](#)" on page 41-29.

*dimension.attribute*: The name of the attribute. Enclose each part of the name in double quotes, for example, "PRODUCT" . "LONG\_DESCRIPTION".

*qdr*: The dimension member being given an attribute value in the form of a qualified data reference, such as "PRODUCT"='OPT MOUSE'.

*attribute\_value*: The value of the attribute, such as 'Optical Mouse'.

- FOR *dimension\_clause* *measure\_clause* BUILD (*commands*)

Restricts the build to particular measures and dimension values, using the following arguments. See "[FOR Clause Example](#)" on page 41-28.

- *dimension\_clause*:

*dimension* ALL | NONE | WHERE *condition* | LEVELS (*level* [, *level*...])

*dimension* is the name of a dimension of the cube.

ALL sets the dimension status to all members before executing the list of commands.

NONE loads values for no dimension members.

WHERE loads values for those dimension members that match the condition.

LEVELS loads values for dimension members in the named levels.

*level* is a level of the named dimension.

– *measure\_clause*:

MEASURES (*measure* [, *measure* . . . ])

*measure* is the name of a measure in the cube.

– *commands*: Any of the other USING commands.

- LOAD [SYNCH | NO SYNCH] [PRUNE | PARALLEL | SERIAL] [WHERE *condition*]

Loads data into the dimension or cube.

– WHERE limits the load to those values in the mapped relational table that match *condition*.

– *condition* is a valid predicate based on the columns of the mapped table. See the "Examples" on page 41-27.

These optional arguments apply only to dimensions:

- SYNCH matches the dimension keys to the relational data source. (Default)
- NO SYNCH loads new dimension keys but does not delete old keys.

These optional arguments apply only to cubes:

- PRUNE: Runs a full table scan on the fact table to determine which partitions to load. For example, if a cube is partitioned by month and the fact table has values only for the last two months, then jobs are only started to load the partitions for the last two months.
- PARALLEL: Each partition is loaded separately. (Default)
- SERIAL: All partitions are loaded in one SELECT statement.

- MODEL *model\_name* [PARALLEL | SERIAL]

Executes a model previously created for the cube. It accepts these arguments:

- PARALLEL: The model runs separately on each partition.
- SERIAL: The model runs on all cubes at the same time. (Default)

- SET

Supports write-back to the cube using the syntax described in "[SCRIPT Parameter: USING Clause: SET command](#)" on page 41-25. (Cubes only)

- SOLVE [PARALLEL | SERIAL]

Aggregates the cube using the rules defined for the cube, including the aggregation operator and the precompute specifications. (Cubes only)

It accepts these arguments:

- PARALLEL: Each partition is solved separately. (Default)

- SERIAL: All partitions are solved at the same time.

**SCRIPT Parameter: USING Clause: AGGREGATE command**

The AGGREGATE command in a script specifies the aggregation rules for one or more measures.

---



---

**Note:** The AGGREGATE command is available only for uncompressed cubes.

---



---

AGGREGATE has the following syntax:

```
{ AGGREGATE USING MEASURE
  WHEN measure1 THEN operator1
  WHEN measure2 THEN operator2...
  ELSE default_operator
|
  [AGGREGATE USING] operator_clause }
processing_options
OVER { ALL | dimension | dimension HIERARCHIES (hierarchy)}
```

**USING MEASURE Clause**

This clause enables you to specify different aggregation operators for different measures in the cube.

**Operator Clause**

The operator\_clause has this syntax:

```
operator(WEIGHTBY expression | SCALEBY expression)
```

WEIGHTBY multiplies each data value by an expression before aggregation.

SCALEBY adds the value of an expression to each data value before aggregation.

**Table 41-3 Aggregation Operators**

Operator	Option	Description
AVG	WEIGHTBY	Adds data values, then divides the sum by the number of data values that were added together.
FIRST	WEIGHTBY	The first real data value.
HIER_AVG	WEIGHTBY	Adds data values, then divides the sum by the number of the children in the dimension hierarchy. Unlike AVERAGE, which counts only non-NA children, HAVERAGE counts all of the logical children of a parent, regardless of whether each child does or does not have a value.
HIER_FIRST	WEIGHTBY	The first data value in the hierarchy, even when that value is NA.
HIER_LAST	WEIGHTBY	The last data value in the hierarchy, even when that value is NA.
LAST	WEIGHTBY	The last real data value.
MAX	WEIGHTBY	The largest data value among the children of each parent.
MIN	WEIGHTBY	The smallest data value among the children of each parent.
SUM	SCALEBY   WEIGHTBY	Adds data values. (Default)

## Processing Options

You can specify these processing options for aggregation:

- (ALLOW | DISALLOW) OVERFLOW
 

Specifies whether to allow decimal overflow, which occurs when the result of a calculation is very large and can no longer be represented by the exponent portion of the numerical representation.

  - ALLOW: A calculation that generates overflow executes without error and produces null results. (Default)
  - DISALLOW: A calculation involving overflow stops executing and generates an error message.
- (ALLOW | DISALLOW) DIVISION BY ZERO
 

Specifies whether to allow division by zero.

  - ALLOW: A calculation involving division by zero executes without error but returns a null value. (Default)
  - DISALLOW: A calculation involving division by zero stops executing and generates an error message.
- (CONSIDER | IGNORE) NULLS
 

Specifies whether nulls are included in the calculations.

  - CONSIDER: Nulls are included in the calculations. A calculation that includes a null value returns a null value.
  - IGNORE: Only actual data values are used in calculations. Nulls are treated as if they do not exist. (Default)
- MAINTAIN COUNT
 

Stores an up-to-date count of the number of dimension members for use in calculating averages. Omit this option to count the members on the fly.

## SCRIPT Parameter: USING Clause: SET command

The SET command in a script assigns values to one or more cells in a stored measure. It has this syntax:

```
SET target = expression
```

Where:

*target* is a measure or a qualified data reference.

*expression* returns values of the appropriate data type for *target*.

### Qualified Data References

Qualified data references (QDRs) limit a dimensional object to a single member in one or more dimensions for the duration of a query.

A QDR has the following syntax:

```
expression [ { dimension = member } [ , { dimension = member } ... ] ]
```

Where:

*expression* is a dimensional expression, typically the name of a measure.

*dimension* is a primary dimension of expression.

*member* is a value of dimension.

The outside square brackets shown in bold are literal syntax elements; they do not indicate an optional argument. The inside square brackets shown in regular text delimit an optional argument and are not syntax elements.

This example returns Sales values for calendar year 2007:

```
global.sales[global.time = 'CY2007'  
]
```

The next example returns Sales values only for the United States in calendar year 2007:

```
sales[customer = 'US', time = 'CY2007'  
]
```

See the Examples for qualified data references in SET commands.

## Usage Notes

### Build Methods

The *C*, *S*, and *?* methods always succeed and can be used on any cube.

The *F* and *P* methods require that the cube have a materialized view that was created as a fast or a rewrite materialized view.

### Parallelism

Partitioned cubes can be loaded and aggregated in parallel processes. For example, a cube with five partitions can use up to five processes. Dimensions are always loaded serially.

The number of parallel processes actually allocated by a build is controlled by the smallest of these factors:

- Number of cubes in the build and the number of partitions in each cube.
- Setting of the `PARALLELISM` argument of the `BUILD` procedure.
- Setting of the `JOB_QUEUE_PROCESSES` database initialization parameter.

Suppose `UNITS_CUBE` has 12 partitions, `PARALLELISM` is set to 10, and `JOB_QUEUE_PROCESSES` is set to 4. OLAP uses four processes, which appear as slave processes in the build log.

The SQL engine may allocate additional processes when the `PARALLEL_DEGREE_POLICY` database initialization parameter is set to `AUTO` or `LIMITED`. For example, if OLAP allocates four processes, the SQL engine might determine that two of those processes should be done by four processes instead, for a total of six processes.

### Build Logs

OLAP generates three logs that provide diagnostic information about builds:

- Cube build log
- Rejected values log
- Cube dimension compile log

Analytic Workspace Manager creates these logs automatically as tables in the same schema as the analytic workspace. If you do not use Analytic Workspace Manager, you can create and manage the logs in PL/SQL using the `DBMS_CUBE_LOG` package.

You can also create the cube log file by running `$ORACLE_HOME/olap/admin/utl_olaplog.sql`. This script creates three additional views:

- CUBE\_BUILD\_LATEST: Returns rows only from the last build.
- CUBE\_BUILD\_REPORT: Returns one row for each command with elapsed times.
- CUBE\_BUILD\_REPORT\_LATEST: Returns a report like CUBE\_BUILD\_REPORT only from the last build.

This report shows a successfully completed build of the objects in the GLOBAL analytic workspace, which has four dimensions and two cubes.

```
SELECT command, status, build_object, build_object_type type
FROM cube_build_report_latest;
```

COMMAND	STATUS	BUILD_OBJECT	TYPE
BUILD	COMPLETED		BUILD
FREEZE	COMPLETED		BUILD
LOAD NO SYNCH	COMPLETED	CHANNEL	DIMENSION
COMPILE	COMPLETED	CHANNEL	DIMENSION
UPDATE/COMMIT	COMPLETED	CHANNEL	DIMENSION
LOAD NO SYNCH	COMPLETED	CUSTOMER	DIMENSION
COMPILE	COMPLETED	CUSTOMER	DIMENSION
UPDATE/COMMIT	COMPLETED	CUSTOMER	DIMENSION
LOAD NO SYNCH	COMPLETED	PRODUCT	DIMENSION
COMPILE	COMPLETED	PRODUCT	DIMENSION
UPDATE/COMMIT	COMPLETED	PRODUCT	DIMENSION
LOAD NO SYNCH	COMPLETED	TIME	DIMENSION
COMPILE	COMPLETED	TIME	DIMENSION
UPDATE/COMMIT	COMPLETED	TIME	DIMENSION
COMPILE AGGMAP	COMPLETED	PRICE_CUBE	CUBE
UPDATE/COMMIT	COMPLETED	PRICE_CUBE	CUBE
COMPILE AGGMAP	COMPLETED	UNITS_CUBE	CUBE
UPDATE/COMMIT	COMPLETED	UNITS_CUBE	CUBE
DBMS_SCHEDULER.CREATE_JOB	COMPLETED	PRICE_CUBE	CUBE
DBMS_SCHEDULER.CREATE_JOB	COMPLETED	UNITS_CUBE	CUBE
BUILD	COMPLETED		BUILD
LOAD	COMPLETED	PRICE_CUBE	CUBE
SOLVE	COMPLETED	PRICE_CUBE	CUBE
UPDATE/COMMIT	COMPLETED	PRICE_CUBE	CUBE
BUILD	COMPLETED		BUILD
LOAD	COMPLETED	UNITS_CUBE	CUBE
SOLVE	COMPLETED	UNITS_CUBE	CUBE
UPDATE/COMMIT	COMPLETED	UNITS_CUBE	CUBE
ANALYZE	COMPLETED	PRICE_CUBE	CUBE
ANALYZE	COMPLETED	UNITS_CUBE	CUBE
THAW	COMPLETED		BUILD

31 rows selected.

## Examples

This example uses the default parameters to build UNITS\_CUBE.

```
EXECUTE DBMS_CUBE.BUILD('GLOBAL.UNITS_CUBE');
```

The next example builds UNITS\_CUBE and explicitly builds two of its dimensions, TIME and CHANNEL. The dimensions use the complete (C) method, and the cube uses the fast solve (S) method.

```
BEGIN
  DBMS_CUBE.BUILD(
    script=>'GLOBAL."TIME", GLOBAL.CHANNEL, GLOBAL.UNITS_CUBE',
```

```

        method=>'CCS',
        parallelism=>2);
END;
/

```

The following example loads only the selection of data identified by the WHERE clause:

```

BEGIN
  DBMS_CUBE.BUILD(q'!
  GLOBAL."TIME",
  GLOBAL.CHANNEL,
  GLOBAL.CUSTOMER,
  GLOBAL.PRODUCT,
  GLOBAL.UNITS_CUBE USING (LOAD NO SYNCH
  WHERE UNITS_FACT.MONTH_ID LIKE '2006%'
  AND UNITS_FACT.SALES > 5000)!');
END;
/

```

### FOR Clause Example

In this example, the Time dimension is partitioned by calendar year, and DBMS\_CUBE builds only the partition identified by CY2006. The HIER\_ANCESTOR is an analytic function in the OLAP expression syntax.

```

BEGIN
  dbms_cube.build(q'!
  UNITS_CUBE USING
  (
  FOR "TIME"
  WHERE HIER_ANCESTOR(WITHIN "TIME".CALENDAR LEVEL "TIME".CALENDAR_YEAR) =
  'CY2006'
  BUILD (LOAD, SOLVE)
  )!',
  parallelism=>1);
END;
/

```

The next example uses a FOR clause to limit the build to the SALES measure in 2006. All objects are built using the complete (C) method.

```

BEGIN
  DBMS_CUBE.BUILD(
  script => '
  GLOBAL."TIME",
  GLOBAL.CHANNEL,
  GLOBAL.CUSTOMER,
  GLOBAL.PRODUCT,
  GLOBAL.UNITS_CUBE USING
  (
  FOR MEASURES(GLOBAL.UNITS_CUBE.SALES)
  BUILD(LOAD NO SYNCH WHERE GLOBAL.UNITS_FACT.MONTH_ID LIKE ''2006%'')
  )',
  method => 'C',
  parallelism => 2);
END;
/

```

### Write-Back Examples

The following examples show various use of the SET command in a USING clause.

This example sets Sales Target to Sales increased by 5%:



```
DBMS_CUBE.BUILD('UNITS_CUBE USING (
  SET UNITS_CUBE.SALES_TARGET = UNITS_CUBE.SALES * 1.05, SOLVE)');
```

This example sets the price of the Deluxe Mouse in May 2007 to \$29.99:

```
DBMS_CUBE.BUILD('PRICE_CUBE USING (
  SET PRICE_CUBE.UNIT_PRICE["TIME"='2007.05', "PRODUCT"='DLX MOUSE']
  = 29.99, SOLVE)');
```

The next example contains two SET commands, but does not reaggregate the cube:

```
DBMS_CUBE.BUILD('PRICE_CUBE USING (
  SET PRICE_CUBE.UNIT_PRICE["TIME"='2006.12', "PRODUCT"='DLX MOUSE']
  = 29.49,
  SET PRICE_CUBE.UNIT_PRICE["TIME"='2007.05', "PRODUCT"='DLX MOUSE']
  = 29.99)');
```

### Dimension Maintenance Example

This script shows dimension maintenance. It adds a new dimension member named OPT MOUSE to all hierarchies, alters its position in the Primary hierarchy, assigns it a long description, then deletes it from the dimension.

```
BEGIN
dbms_output.put_line('Add optical mouse');
dbms_cube.build(q'!
  "PRODUCT" using (MERGE INTO ALL HIERARCHIES
    VALUES ('ITEM_OPT MOUSE', 'CLASS_SFT', "PRODUCT"."FAMILY"))
!');

dbms_output.put_line('Alter optical mouse');
dbms_cube.build(q'!
  "PRODUCT" using (UPDATE HIERARCHIES ("PRODUCT"."PRIMARY")
    SET PARENT = 'FAMILY_ACC', LEVEL = "PRODUCT"."ITEM"
    WHERE MEMBER = 'ITEM_OPT MOUSE')
!');

dbms_output.put_line('Provide attributes to optical mouse');
dbms_cube.build(q'!
  "PRODUCT" USING (SET "PRODUCT"."LONG_DESCRIPTION"["PRODUCT" = 'ITEM_OPT MOUSE']
    = CAST('Optical Mouse' AS VARCHAR2))
!');

dbms_output.put_line('Delete optical mouse');
dbms_cube.build(q'!
  "PRODUCT" USING (DELETE FROM DIMENSION WHERE MEMBER='ITEM_OPT MOUSE')
!');

END;
/
```

### OLAP DML Example

This example uses the OLAP DML to add comments to the cube build log:

```
BEGIN
DBMS_CUBE.BUILD(q'!
global.units_cube USING (
EXECUTE OLAP DML 'SHOW STATLEN(units_cube_prt_list)' PARALLEL,
EXECUTE OLAP DML 'SHOW LIMIT(units_cube_prt_list KEEP ALL)' PARALLEL,
EXECUTE OLAP DML 'SHOW STATLEN(time)' parallel,
EXECUTE OLAP DML 'SHOW LIMIT(time KEEP time_levelrel 'CALENDAR_YEAR')'
```

```
parallel!'),
  parallelism=>2,
  add_dimensions=>false);
END;
/
```

This query shows the comments in the cube build log:

```
SELECT partition, slave_number, TO_CHAR(output) output
FROM cube_build_log
WHERE command = 'OLAP DML'
AND status = 'COMPLETED'
ORDER BY slave_number, time;
```

PARTITION	SLAVE_NUMBER	OUTPUT
P10:CY2007	1	<OLAPDMLExpression Expression="TO_CHAR(statlen(units_cube_prt_list))" Value="1"/>
P10:CY2007	1	<OLAPDMLExpression Expression="TO_CHAR(limit(units_cube_prt_list keep al 1))" Value="P10"/>
P10:CY2007	1	<OLAPDMLExpression Expression="TO_CHAR(statlen(time))" Value="17"/>
P10:CY2007	1	<OLAPDMLExpression Expression="TO_CHAR(limit(time keep time_levelrel &ap os;CALENDAR_YEAR&apos;))" Value="CALENDAR_YEAR_CY2007"/>
P9:CY2006	2	<OLAPDMLExpression Expression="TO_CHAR(statlen(units_cube_prt_list))" Value="1"/>
P9:CY2006	2	<OLAPDMLExpression Expression="TO_CHAR(limit(units_cube_prt_list keep al 1))" Value="P9"/>
P9:CY2006	2	<OLAPDMLExpression Expression="TO_CHAR(statlen(time))" Value="17"/>
	.	
	.	
	.	

## CREATE\_EXPORT\_OPTIONS Procedure

This procedure creates an input XML document that describes processing options for the [EXPORT\\_XML Procedure](#) on page 41-45 and the [EXPORT\\_XML\\_TO\\_FILE Procedure](#) on page 41-47.

### Syntax

```
DBMS_CUBE.CREATE_EXPORT_OPTIONS (
    out_options_xml      IN/OUT  CLOB,
    target_version       IN      VARCHAR2  DEFAULT NULL,
    suppress_owner       IN      BOOLEAN   DEFAULT FALSE,
    suppress_namespace   IN      BOOLEAN   DEFAULT FALSE,
    preserve_table_owners IN      BOOLEAN   DEFAULT FALSE,
    metadata_changes     IN      CLOB      DEFAULT NULL);
```

### Parameters

**Table 41–4 CREATE\_EXPORT\_OPTIONS Procedure Parameters**

Parameter	Description
out_options_xml	Contains the generated XML document, which can be passed into the options_xml parameter of the <a href="#">EXPORT_XML Procedure</a> .
target_version	Specifies the version of Oracle Database in which the XML document generated by EXPORT_XML or EXPORT_XML_TO_FILE will be imported. You can specify two to five digits, such as 11.2 or 11.2.0.2.0. This parameter defaults to the current database version, and so can typically be omitted.
suppress_owner	Controls the use of the Owner attribute in XML elements and the owner qualifier in object names. Enter <code>True</code> to drop the owner from the XML, or enter <code>False</code> to retain it. Enter <code>True</code> if you plan to import the exported metadata into a different schema.
suppress_namespace	Controls the use of Namespace attributes in XML elements and the namespace qualifier in object names. Enter <code>True</code> to drop the namespace from the XML, or enter <code>False</code> to retain it (default). Enter <code>True</code> when upgrading to Oracle OLAP 11g metadata.  Namespaces allow objects created in Oracle 10g to coexist with objects created in Oracle 11g. You cannot set or change namespaces.
preserve_table_owners	Controls the use of the owner in qualifying table names in the mapping elements, such as GLOBAL.UNITS_HISTORY_FACT instead of UNITS_HISTORY_FACT. Enter <code>True</code> to retain the table owner, or enter <code>False</code> to default to the current schema for table mappings. If you plan to import the exported metadata to a different schema, you must set this option to <code>True</code> to load data from tables and views in the original schema, unless the destination schema has its own copies of the tables and views.
metadata_changes	Contains an 11g XML description of an object that overwrites the exported object description. The XML document must contain all parent XML elements of the modified element with the attributes needed to uniquely identify them. Use the Name attribute if it exists. See the Examples.

### Examples

The following example generates an XML document of export options:

```

DECLARE
    optionsClob CLOB;

BEGIN
    dbms_lob.createtemporary(optionsClob, false, dbms_lob.CALL);
    dbms_cube.create_export_options(out_options_xml=>optionsClob, suppress_
namespace=>TRUE);
    dbms_output.put_line(optionsClob);
END;
/

```

The DBMS\_OUTPUT.PUT\_LINE procedure displays this XML document (formatted for readability):

```

<?xml version="1.0"?>
<Export TargetVersion="11.2.0.2">
  <ExportOptions>
    <Option Name="SuppressOwner" Value="FALSE"/>
    <Option Name="SuppressNamespace" Value="TRUE"/>
    <Option Name="PreserveTableOwners" Value="FALSE"/>
  </ExportOptions>
</Export>

```

The next example generates an XML document with a metadata change to the mapping of the American long description attribute of the CHANNEL dimension.

```

DECLARE
    importClob      clob;
    exportClob      clob;
    overClob        clob;
    exportOptClob   clob;
    importOptClob   clob;

BEGIN
    dbms_lob.createtemporary(overClob, TRUE);
    dbms_lob.open(overClob, DBMS_LOB.LOB_READWRITE);
    dbms_lob.writeappend(overClob,58, '<Metadata Version="1.3"
MinimumDatabaseVersion="11.2.0.2">');
    dbms_lob.writeappend(overClob,34, '<StandardDimension Name="CHANNEL">');
    dbms_lob.writeappend(overClob,75, '<Description Type="Description"
Language="AMERICAN" Value="Sales Channel"/>');
    dbms_lob.writeappend(overClob,20, '</StandardDimension>');
    dbms_lob.writeappend(overClob,11, '</Metadata>');
    dbms_lob.close(overClob);

    -- Enable 11g clients to access 10g metadata
    dbms_cube.initialize_cube_upgrade;

    -- Create a CLOB containing the export options
    dbms_lob.createtemporary(exportOptClob, TRUE);
    dbms_cube.create_export_options(out_options_xml=>exportOptClob, suppress_
namespace=>TRUE, metadata_changes=>overClob);

    -- Create a CLOB containing the import options
    dbms_lob.createtemporary(importOptClob, TRUE);
    dbms_cube.create_import_options(out_options_xml=>importOptClob, rename_table =>
'MY_OBJECT_MAP');

    -- Create CLOBs for the metadata
    dbms_lob.createtemporary(importClob, TRUE);
    dbms_lob.createtemporary(exportClob, TRUE);

```

```
-- Export metadata from a 10g analytic workspace to a CLOB
dbms_cube.export_xml(object_ids=>'GLOBAL_AW', options_xml=>exportOptClob, out_
xml=>exportClob);

-- Import metadata from the CLOB
dbms_cube.import_xml(in_xml => exportClob, options_xml=>importOptClob, out_
xml=>importClob);

-- Load and aggregate the data
dbms_cube.build(script=>'UNITS_CUBE, PRICE_AND_COST_CUBE');

END;
/
```

The following is the content of `exportClob` (formatting added for readability). The XML document changes the description of Channel to Sales Channel.

```
<Metadata Version="1.3" MinimumDatabaseVersion="11.2.0.2">
  <StandardDimension Name="CHANNEL">
    <Description Type="Description" Language="AMERICAN" Value="Sales Channel"/>
  </StandardDimension>
</Metadata>
```

## CREATE\_IMPORT\_OPTIONS Procedure

This procedure creates an input XML document that describes processing options for the [IMPORT\\_XML Procedure](#) on page 41-49.

### Syntax

```
DBMS_CUBE.CREATE_IMPORT_OPTIONS (
    out_options_xml  IN/OUT  CLOB,
    validate_only    IN      BOOLEAN  DEFAULT FALSE,
    rename_table     IN      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 41–5 CREATE\_IMPORT\_OPTIONS Procedure Parameters**

Parameter	Description
out_options_xml	Contains the generated XML document, which can be passed to the options_xml parameter of the <a href="#">IMPORT_XML Procedure</a> .
validate_only	TRUE causes the IMPORT_XML procedure to validate the metadata described in the input file or the in_xml parameter, without committing the changes to the metadata.
rename_table	The name of a table identifying new names for the imported objects, in the form [schema_name.]table_name. The IMPORT_XML procedure creates objects using the names specified in the table instead of the ones specified in the XML document. See the Usage Notes for the format of the rename table.

### Usage Notes

See "[Rename Table](#)" on page 41-14.

### Examples

This example specifies validation only and a rename table. For an example of the import CLOB being used in an import, see "[IMPORT\\_XML Procedure](#)" on page 41-49.

```
DECLARE
importClob  clob;

BEGIN
    dbms_lob.createtemporary(importClob, TRUE);

    dbms_cube.create_import_options(out_options_xml => importClob, rename_table =>
'MY_OBJECT_MAP', validate_only => TRUE);

    dbms_output.put_line(importClob);
END;
/
```

It generates the following XML document:

```
<?xml version="1.0"?>
<Import>
  <ImportOptions>
    <Option Name="ValidateOnly" Value="TRUE"/>
    <Option Name="RenameTable" Value="MY_OBJECT_MAP"/>
  </ImportOptions>
</Import>
```

</Import>

## CREATE\_MVIEW Function

This function creates a cube materialized view from the definition of a relational materialized view.

### Syntax

```
DBMS_CUBE.CREATE_MVIEW (
    mvowner      IN  VARCHAR2,
    mvname       IN  VARCHAR2,
    sam_parameters IN CLOB  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 41–6 CREATE\_MVIEW Function Parameters**

Parameter	Description
mvowner	Owner of the relational materialized view.
mvname	Name of the relational materialized view. For restrictions, see <a href="#">"Requirements for the Relational Materialized View"</a> on page 41-6.  A single cube materialized view can replace many of the relational materialized views for a table. Choose the materialized view that has the lowest levels of the dimension hierarchies that you want represented in the cube materialized view.
sam_parameters	Parameters in the form ' <i>parameter1=value1, parameter2=value2,...</i> '. See <a href="#">"SQL Aggregation Management Parameters"</a> .

### SQL Aggregation Management Parameters

The CREATE\_MVIEW and DERIVE\_FROM\_MVIEW functions use the SQL aggregation management (SAM) parameters described in [Table 41–7](#). Some parameters support the development of cubes with advanced analytics. Other parameters support the development of Java applications. The default settings are appropriate for cube materialized views that are direct replacements for relational materialized views.

**Table 41–7 SQL Aggregation Management Parameters**

Parameter	Description
ADDTOPS	Adds a top level and a level member to every dimension hierarchy in the cube. If the associated relational dimension has no hierarchy, then a dimension hierarchy is created.  TRUE: Creates levels named <i>ALL_dimension</i> with level members <i>All_dimension</i> . (Default)  FALSE: Creates only the hierarchies and levels identified by the relational dimensions.
ADDUNIQUEKEYPREFIX	Controls the creation of dimension keys.  TRUE: Creates cube dimension keys by concatenating the level name with the relational dimension key. This practice assures that the dimension keys are unique across all levels, such as <i>CITY_NEW_YORK</i> and <i>STATE_NEW_YORK</i> . (Default)  FALSE: Uses the relational dimension keys as cube dimension keys.



**Table 41–7 (Cont.) SQL Aggregation Management Parameters**

Parameter	Description
ATRMAPTTYPE	<p>Specifies whether attributes are mapped by hierarchy levels, dimension levels, or both.</p> <p>HIER_LEVEL: Maps attributes to the levels of a particular dimension hierarchy. (Default)</p> <p>DIM_LEVEL: Maps attributes to the levels of the dimension regardless of hierarchy.</p> <p>BOTH: Maps attributes to both dimension and hierarchy levels.</p> <p>AUTO: Maps attributes to the levels of the dimension for a star schema and to the levels of a particular dimension hierarchy for a snowflake schema.</p>
AWNAME	<p>Provides the name of the analytic workspace that owns the cube. Choose a simple database object name of 1 to 30 bytes. The default name is <i>fact_tablename_AWn</i>.</p>
BUILD	<p>Specifies whether a data refresh will immediately follow creation of the cube materialized view.</p> <p>IMMEDIATE: Refreshes immediately.</p> <p>DEFERRED: Does not perform a data refresh. (Default)</p> <p><b>Note:</b> Only the CREATE_MVIEW function uses this parameter.</p>
CUBEMVOPTION	<p>Controls validation and creation of a cube materialized view. Regardless of this setting, the function creates an analytic workspace containing a cube and its related cube dimensions.</p> <p>COMPLETE_REFRESH: Creates a complete refresh cube materialized view (full update).</p> <p>FAST_REFRESH: Creates a fast refresh materialized view (incremental update).</p> <p>REWRITE_READY: Runs validation checks for a rewrite cube materialized view, but does not create it.</p> <p>REWRITE: Creates a rewrite cube materialized view.</p> <p>REWRITE_WITH_ATTRIBUTES: Creates a rewrite cube materialized view that includes columns with dimension attributes, resulting in faster query response times. (Default)</p> <p><b>Note:</b> <i>The following settings do not create a cube materialized view. Use Analytic Workspace Manager to drop an analytic workspace that does not have a cube materialized view. You can use the DROP_MVIEW procedure to delete an analytic workspace only when it supports a cube materialized view.</i></p> <p>NONE: Does not create a cube materialized view.</p> <p>COMPLETE_REFRESH_READY: Runs validation checks for a complete refresh cube materialized view, but does not create it.</p> <p>FAST_REFRESH_READY: Runs validation checks for fast refresh, but does not create the cube materialized view.</p>
CUBENAME	<p>Provides the name of the cube derived from the relational materialized view. Choose simple database object name of 1 to 30 bytes. The default name is <i>fact_tablename_Cn</i>.</p>

**Table 41–7 (Cont.) SQL Aggregation Management Parameters**

Parameter	Description
DIMJAVABINDVARS	<p>Supports access by Java programs to the XML document.</p> <p>TRUE: Generates an XML template that uses Java bind variable notation for the names of dimensions. No XML validation is performed. You cannot use the <code>IMPORT_XML</code> procedure to create a cube using this template.</p> <p>FALSE: Generates an XML template that does not support Java bind variables. (Default)</p>
DISABLEQRW	<p>Controls disabling of query rewrite on the source relational materialized view.</p> <p>TRUE: Issues an <code>ALTER MATERIALIZED VIEW <i>mview_name</i> DISABLE QUERY REWRITE</code> command.</p> <p>FALSE: No action.</p> <p><b>Note:</b> Only the <code>CREATE_MVIEW</code> function with <code>BUILD=IMMEDIATE</code> uses this parameter.</p>
EXPORTXML	<p>Exports the XML that defines the dimensional objects to a file, which you specify as <code>dir/filename</code>. Both the directory and the file name are case sensitive.</p> <p><i>dir</i>: Name of a database directory.</p> <p><i>filename</i>: The name of the file, typically given an XML filename extension.</p>
FILTERPARTITIONANCESTORLEVELS	<p>Controls the generation of aggregate values above the partitioning level of a partitioned cube.</p> <p>TRUE: Removes levels above the partitioning level from the cube. Requests for summary values above the partitioning level are solved by SQL.</p> <p>FALSE: All levels are retained in the cube. Requests for summary values are solved by OLAP. (Default)</p>
LOGDEST	<p>Directs and stores log messages. By default, the messages are not available.</p> <p>SERVEROUT: Sends messages to server output (typically the screen), which is suitable when working interactively such as in SQL*Plus or SQL Developer.</p> <p>TRACEFILE: Sends messages to the session trace file.</p>
PARTITIONOPTION	<p>Controls partitioning of the cube.</p> <p>NONE: Prevents partitioning.</p> <p>DEFAULT: Allows the Sparsity Advisor to determine whether partitioning is needed and how to partition the cube. (Default)</p> <p>FORCE: Partitions the cube even when the Sparsity Advisor recommends against it. The Sparsity Advisor identifies the best dimension, hierarchy, and level to use for partitioning.</p> <p><i>dimension.hierarchy.level</i>: Partitions the cube using the specified dimension, hierarchy, and level.</p>
POPULATELINEAGE	<p>Controls the appearance of attributes in a cube materialized view.</p> <p>TRUE: Includes all dimension attributes in the cube materialized view. (Default)</p> <p>FALSE: Omits all dimension attributes from the cube materialized view.</p>

**Table 41-7 (Cont.) SQL Aggregation Management Parameters**

Parameter	Description
PRECOMPUTE	<p>Identifies a percentage of the data that is aggregated and stored. The remaining values are calculated as required by queries during the session.</p> <p><i>precompute_percentage[:precompute_top_percentage]</i></p> <p>Specify the top percentage for partitioned cubes. The default value is 35:0, which specifies precomputing 35% of the bottom partition and 0% of the top partition. If the cube is not partitioned, then the second number is ignored.</p>
REMAPCOMPOSITEKEYS	<p>Controls how multicolumn keys are rendered in the cube.</p> <p>TRUE: Creates a unique key attribute whose values are concatenated string expressions with an underscore between the column values. For example, the value BOSTON_MA_USA might be an expression produced from a multicolumn key composed of CITY, STATE, and COUNTRY columns. In addition, an attribute is created for each individual column to store the relational keys. (Default)</p> <p>FALSE: Creates a unique key attribute for each column.</p>
RENDERINGMODE	<p>Controls whether a loss in fidelity between the relational materialized view and the cube materialized view results in a warning message or an exception. See <a href="#">"Requirements for the Relational Materialized View"</a> on page 41-6.</p> <p>LOOSE: Losses are noted in the optional logs generated by the <a href="#">CREATE_MVIEW Function</a> and the <a href="#">DERIVE_FROM_MVIEW Function</a>. No exceptions are raised. (Default)</p> <p>STRICT: Any loss in fidelity raises an exception so that no XML template is created.</p>
SEEFILTERS	<p>Controls whether conditions in the WHERE clause of the relational materialized view's defining query are retained or ignored.</p> <p>TRUE: Renders valid conditions in the XML template. (Default)</p> <p>FALSE: Ignores all conditions.</p>
UNIQUENAMES	<p>Controls whether top level dimensional objects have unique names. Cross namespace conflicts may occur because dimensional objects have different namespaces than relational objects.</p> <p>TRUE: Modifies all relational names when they are rendered in the cube.(Default)</p> <p>FALSE: Duplicates relational names in the cube unless a naming conflict is detected. In that case, a unique name is created.</p>
UNKNOWNKEYASDIM	<p>Controls handling of simple columns with no levels or hierarchies in the GROUP BY clause of the relational materialized view's defining query.</p> <p>TRUE: Renders a simple column without a relational dimension as a cube dimension with no levels or hierarchies.</p> <p>FALSE: Raises an exception when no relational dimension is found for the column. (Default)</p>
VALIDATEXML	<p>Controls whether the generated XML document is validated.</p> <p>TRUE: Validates the template using the VALIDATE_XML procedure. (Default)</p> <p>FALSE: No validation is done.</p>

## Returns

The name of the cube materialized view created by the function.

## Usage Notes

See ["Using SQL Aggregation Management"](#) on page 41-4

## Examples

All examples for the SQL Aggregate Management subprograms use the sample Sales History schema, which is installed in Oracle Database with two relational materialized views: CAL\_MONTH\_SALES\_MV and FWEEK\_PSCAT\_SALES\_MV.

The following script creates a cube materialized view using CAL\_MONTH\_SALES\_MV as the relational materialized view. It uses all default options.

```
SET serverout ON format wrapped
```

```
DECLARE
    salesaw varchar2(30);

BEGIN
    salesaw := dbms_cube.create_mview('SH', 'CAL_MONTH_SALES_MV');
END;
/
```

The next example sets several parameters for creating a cube materialized view from FWEEK\_PSCAT\_SALES\_MV. These parameters change the cube materialized view in the following ways:

- **ADDTOPS:** Adds a top level consisting of a single value to the hierarchies. All of the dimensions in Sales History have a top level already.
- **PRECOMPUTE:** Changes the percentage of materialized aggregates from 35:0 to 40:10.
- **EXPORTXML:** Creates a text file for the XML document.
- **BUILD:** Performs a data refresh.

```
DECLARE
    salescubemv varchar2(30);
    sam_param clob := 'ADDTOPS=FALSE,
                     PRECOMPUTE=40:10,
                     EXPORTXML=WORK_DIR/sales.xml,
                     BUILD=IMMEDIATE';

BEGIN
    salescubemv := dbms_cube.create_mview('SH', 'FWEEK_PSCAT_SALES_MV',
                                         sam_param);
END;
/
```

## DERIVE\_FROM\_MVIEW Function

This function generates an XML template that defines a cube with materialized view capabilities, using the information derived from an existing relational materialized view.

### Syntax

```
DBMS_CUBE.DERIVE_FROM_MVIEW (
    mvowner      IN  VARCHAR2,
    mvname       IN  VARCHAR2,
    sam_parameters IN CLOB  DEFAULT NULL)
RETURN CLOB;
```

### Parameters

**Table 41–8** *DERIVE\_FROM\_MVIEW Function Parameters*

Parameter	Description
mvowner	Owner of the relational materialized view.
mvname	Name of the relational materialized view. For restrictions, see <a href="#">"Requirements for the Relational Materialized View"</a> on page 41-6.  A single cube materialized view can replace many of the relational materialized views for a table. Choose the materialized view that has the lowest levels of the dimension hierarchies that you want represented in the cube materialized view.
sam_parameters	Optional list of parameters in the form ' <i>parameter1=value1, parameter2=value2,...</i> '. See <a href="#">"SQL Aggregation Management Parameters"</a> on page 41-36.

### Returns

An XML template that defines an analytic workspace containing a cube enabled as a materialized view.

### Usage Notes

To create a cube materialized view from an XML template, use the `IMPORT_XML` procedure. Then use the `REFRESH_MVIEW` procedure to refresh the cube materialized view with data.

See ["Using SQL Aggregation Management"](#) on page 41-4.

### Examples

The following example generates an XML template named `sales_cube.xml` from the `CAL_MONTH_SALES_MV` relational materialized view in the `SH` schema.

```
DECLARE
    salescubexml  clob := null;
    sam_param     clob := 'exportXML=WORK_DIR/sales_cube.xml';

BEGIN
    salescubexml := dbms_cube.derive_from_mview('SH', 'CAL_MONTH_SALES_MV',
    sam_param);
END;
```

/

## DROP\_MVIEW Procedure

This procedure drops a cube materialized view and all associated objects from the database. These objects include the dimension materialized views, cubes, cube dimensions, levels, hierarchies, and the analytic workspace.

### Syntax

```
DBMS_CUBE.DROP_MVIEW (
    mvowner      IN  VARCHAR2,
    mvname       IN  VARCHAR2,
    sam_parameters IN CLOB  DEFAULT NULL);
```

### Parameters

**Table 41–9 DROP\_MVIEW Procedure Parameters**

Parameter	Description
mvowner	Owner of the cube materialized view
mvname	Name of the cube materialized view
sam_parameters	EXPORTXML: Exports the XML that drops the dimensional objects to a file, which you specify as <i>dir/filename</i> . Both the directory and the file name are case sensitive.  <i>dir</i> : Name of a database directory.  <i>filename</i> : The name of the file, typically given an XML filename extension.

### Usage Notes

Use this procedure to drop a cube materialized view that you created using the CREATE\_MVIEW and DERIVE\_FROM\_MVIEW functions. If you make modifications to the cubes or dimensions, then DROP\_MVIEW may not be able to drop the cube materialized view.

Some of the CUBEMVOPTION parameters used by the CREATE\_MVIEW and DERIVE\_FROM\_MVIEW functions do not create a materialized view. Use Analytic Workspace Manager to drop the analytic workspace, cubes, and cube dimensions.

If you use the EXPORTXML parameter, then you can use the XML document to drop the cube materialized view, after you re-create it. Use the IMPORT\_XML procedure.

See ["Using SQL Aggregation Management"](#) on page 41-4.

### Examples

The current schema has four materialized views. CB\$CAL\_MONTH\_SALES is a cube materialized view for the SALES table. CB\$TIMES\_DIM\_D1\_CAL\_ROLLUP is a cube dimension materialized view for the TIMES\_DIM dimension on the TIMES dimension table. The others are relational materialized views.

```
SELECT mview_name FROM user_mvviews;
```

```
MVIEW_NAME
-----
CB$CAL_MONTH_SALES
CB$TIMES_DIM_D1_CAL_ROLLUP
CAL_MONTH_SALES_MV
```

FWEEK\_PSCAT\_SALES\_MV

The following command drops both CB\$CAL\_MONTH\_SALES and CB\$TIMES\_DIM\_D1\_CAL\_ROLLUP.

```
EXECUTE dbms_cube.drop_mview('SH', 'CB$CAL_MONTH_SALES');
```

Dropped cube organized materialized view "SH"."CAL\_MONTH\_SALES" including container analytic workspace "SH"."CAL\_MONTH\_SALES\_AW" at 20081110 16:31:40.056.

This query against the data dictionary confirms that the materialized views have been dropped.

```
SELECT mview_name FROM user_mviews;
```

MVIEW\_NAME

-----

CAL\_MONTH\_SALES\_MV

FWEEK\_PSCAT\_SALES\_MV



## EXPORT\_XML Procedure

This procedure writes OLAP metadata to a CLOB.

### Syntax

```
DBMS_CUBE.EXPORT_XML
  (object_ids      IN      VARCHAR2,
   out_xml         IN/OUT CLOB;
```

```
DBMS_CUBE.EXPORT_XML
  (object_ids      IN      VARCHAR2,
   options_xml     IN      CLOB,
   out_xml         IN/OUT CLOB;
```

```
DBMS_CUBE.EXPORT_XML
  (object_ids      IN      VARCHAR2,
   options_dirname IN      VARCHAR2,
   options_filename IN     VARCHAR2,
   out_xml         IN/OUT CLOB;
```

### Parameters

**Table 41–10 EXPORT\_XML Procedure Parameters**

Parameter	Description
object_ids	<p>Any of these identifiers.</p> <ul style="list-style-type: none"> <li>■ The name of a schema, such as GLOBAL.</li> <li>■ The fully qualified name of an analytic workspace in the form <i>owner.aw_name.AW</i>, such as GLOBAL.GLOBAL.AW.</li> <li>■ Cube</li> <li>■ Dimension</li> <li>■ Named build process</li> <li>■ Measure folder</li> </ul> <p>You can specify multiple objects by separating the names with commas.</p> <p><b>Note:</b> When exporting an individual object, be sure to export any objects required to reconstruct it. For example, when exporting a cube, you must also export the dimensions of the cube.</p>
options_dirname	The case-sensitive name of a database directory that contains <i>options_filename</i> .
options_filename	A file containing an XML document of export options.
options_xml	A CLOB variable that contains an XML document of export options. Use the <a href="#">CREATE_EXPORT_OPTIONS Procedure</a> to generate this document.
out_xml	A CLOB variable that will store the XML document of OLAP metadata for the objects listed in <i>object_ids</i> .

### Export Options

The default settings for the export options are appropriate in many cases, so you can omit the *options\_xml* parameter or the *options\_dirname* and *options\_filename* parameters. However, when upgrading Oracle OLAP 10g metadata to

OLAP 11g, you must specify an XML document that changes the default settings. This example changes all of the parameters from False to True; set them appropriately for your schema.

```
<?xml version="1.0"?>
<Export>
  <ExportOptions>
    <Option Name="SuppressNamespace" Value="True"/>
    <Option Name="SuppressOwner" Value="True"/>
    <Option Name="PreserveTableOwners" Value="True"/>
  </ExportOptions>
</Export>
```

You can create this XML document manually or by using the [CREATE\\_EXPORT\\_OPTIONS Procedure](#) on page 41-31.

## Usage Notes

See "[Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g](#)" on page 41-12.

## Example

For an example of using EXPORT\_XML in an upgrade to the same schema, see "[Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g](#)" on page 41-12.

The following PL/SQL script copies an OLAP 11g analytic workspace named GLOBAL11 from the GLOBAL\_AW schema to the GLOBAL schema. No upgrade is performed.

To upgrade into a different schema, change the example as follows:

- Call the INITIALIZE\_CUBE\_UPGRADE procedure.
- Call the CREATE\_EXPORT\_OPTIONS procedure with the additional parameter setting SUPPRESS\_NAMESPACE=>TRUE.

The PL/SQL client must be connected to the database as GLOBAL. The GLOBAL user must have SELECT permissions on GLOBAL\_AW.AW\$GLOBAL and on all relational data sources.

```
BEGIN
  -- Create a CLOB for the export options
  dbms_lob.createtemporary(optionsClob, TRUE);
  dbms_cube.create_export_options(out_options_xml=>optionsClob, suppress_
owner=>TRUE, preserve_table_owners=>TRUE);

  -- Create a CLOB for the XML template
  dbms_lob.createtemporary(exportClob, TRUE);

  -- Export metadata from an analytic workspace to a CLOB
  dbms_cube.export_xml(object_ids=>'GLOBAL_AW.GLOBAL11.AW', options_
xml=>optionsClob, out_xml=>exportClob);

  -- Import metadata from the CLOB
  dbms_cube.import_xml(in_xml=>exportClob);

  -- Load and aggregate the data
  dbms_cube.build(script=>'GLOBAL.UNITS_CUBE, GLOBAL.PRICE_AND_COST_CUBE');

END;
/
```

## EXPORT\_XML\_TO\_FILE Procedure

This procedure exports OLAP metadata to a file. This file can be imported into a new or existing analytic workspace using the `IMPORT_XML` procedure. In this way, you can create a copy of the analytic workspace in another schema or database.

This procedure can also be used as part of the process for upgrading CWM or OLAP standard form (AWXML) metadata contained in an Oracle OLAP 10g analytic workspace to OLAP 11g format.

### Syntax

```
DBMS_CUBE.EXPORT_XML_TO_FILE
    (object_ids      IN      VARCHAR2,
     output_dirname  IN      VARCHAR2,
     output_filename IN      VARCHAR2;

DBMS_CUBE.EXPORT_XML_TO_FILE
    (object_ids      IN      VARCHAR2,
     options_dirname IN      VARCHAR2,
     options_filename IN     VARCHAR2,
     output_dirname  IN      VARCHAR2,
     output_filename IN     VARCHAR2;
```

### Parameters

**Table 41–11 EXPORT\_XML\_TO\_FILE Procedure Parameters**

Parameter	Description
<code>object_ids</code>	<p>Any of these identifiers.</p> <ul style="list-style-type: none"> <li>■ The name of a schema, such as <code>GLOBAL</code>.</li> <li>■ The fully qualified name of an analytic workspace in the form <code>owner.aw_name.AW</code>, such as <code>GLOBAL.GLOBAL.AW</code>.</li> <li>■ Cube</li> <li>■ Dimension</li> <li>■ Named build process</li> <li>■ Measure folder</li> </ul> <p>You can specify multiple objects by separating the names with commas.</p> <p><b>Note:</b> When exporting an individual object, be sure to export any objects required to reconstruct it. For example, when you export a cube, you must also export the dimensions of the cube.</p>
<code>options_dirname</code>	The case-sensitive name of a database directory that contains <code>options_filename</code> . See "Export Options".
<code>options_filename</code>	The name of a file containing an XML document of export options. See "Export Options".
<code>output_dirname</code>	The case-sensitive name of a database directory where <code>output_filename</code> is created.
<code>output_filename</code>	The name of the template file created by the procedure.

### Export Options

The default settings for the export options are appropriate in most cases, and you can omit the `options_dirname` and `options_filename` parameters. However, when

upgrading Oracle OLAP 10g metadata to OLAP 11g, you must specify an XML document that changes the default settings, like the following:

```
<?xml version="1.0"?>
<Export>
  <ExportOptions>
    <Option Name="SuppressNamespace" Value="True"/>
    <Option Name="SuppressOwner" Value="True"/>
    <Option Name="PreserveTableOwners" Value="True"/>
  </ExportOptions>
</Export>
```

You can create this XML document manually or by using the [CREATE\\_EXPORT\\_OPTIONS Procedure](#) on page 41-31.

## Usage Notes

See "[Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g](#)" on page 41-12.

## Examples

The following example generates an XML file named `global.xml` in OLAP 11g format using the default export settings. The metadata is derived from all analytic workspaces and CWM metadata in the `GLOBAL_AW` schema. The output file is generated in the `WORK_DIR` database directory.

```
execute dbms_cube.export_xml_to_file('GLOBAL_AW', 'WORK_DIR', 'global.xml');
```

The next example also generates an XML file named `global.xml` in OLAP 11g format using the export options set in `options.xml`. The metadata is derived from the `GLOBAL` analytic workspace in the `GLOBAL_AW` schema. Both the options file and the output file are in the `WORK_DIR` database directory.

```
execute dbms_cube.export_xml_to_file('GLOBAL_AW.GLOBAL.AW', 'WORK_DIR',
'options.xml', 'WORK_DIR', 'global.xml');
```

## IMPORT\_XML Procedure

This procedure creates, modifies, or drops an analytic workspace by using an XML template.

### Syntax

```
DBMS_CUBE.IMPORT_XML
  (dirname          IN      VARCHAR2,
   filename         IN      VARCHAR2 );
```

```
DBMS_CUBE.IMPORT_XML
  (dirname          IN      VARCHAR2,
   filename         IN      VARCHAR2,
   out_xml          IN/OUT  CLOB );
```

```
DBMS_CUBE.IMPORT_XML
  (input_dirname    IN      VARCHAR2,
   input_filename   IN      VARCHAR2,
   options_dirname  IN      VARCHAR2,
   options_filename IN      VARCHAR2,
   out_xml          IN/OUT  CLOB );
```

```
DBMS_CUBE.IMPORT_XML
  (in_xml           IN      CLOB );
```

```
DBMS_CUBE.IMPORT_XML
  (in_xml           IN      CLOB,
   out_xml          IN/OUT  CLOB );
```

```
DBMS_CUBE.IMPORT_XML
  (in_xml           IN      CLOB,
   options_xml      IN      CLOB,
   out_xml          IN/OUT  CLOB );
```

### Parameters

**Table 41–12** *IMPORT\_XML Procedure Parameters*

Parameter	Description
dirname	The case-sensitive name of a database directory containing the XML document describing an analytic workspace.
filename	A file containing an XML document describing an analytic workspace.
in_xml	A CLOB containing an XML document describing an analytic workspace.
input_dirname	The case-sensitive name of a database directory containing the XML document describing an analytic workspace.
input_filename	A file containing an XML document describing an analytic workspace.
options_dirname	The case-sensitive name of a database directory containing a file of import options.
options_filename	A file of import options.

**Table 41–12 (Cont.) IMPORT\_XML Procedure Parameters**

Parameter	Description
options_xml	An XML document describing the import options. Use the <a href="#">CREATE_IMPORT_OPTIONS Procedure</a> on page 41-34 to generate this document.
out_xml	An XML document that either describes the analytic workspace or, for validation only, describes any errors. It may contain changes that DBMS_CUBE made to the imported XML, such as setting default values or making minor corrections to the XML.

## Usage Notes

The XML can define, modify, or drop an entire analytic workspace, or one or more cubes or dimensions. When defining just cubes or dimensions, you must do so within an existing analytic workspace.

You can also use `IMPORT_XML` to drop an analytic workspace by using the XML document generated by the `DROP_MVIEW` procedure with the `EXPORTXML` parameter.

See "[Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g](#)" on page 41-12.

## Example

This example loads an XML template from a file named `GLOBAL.XML` and located in a database directory named `XML_DIR`.

```
EXECUTE dbms_cube.import_xml('XML_DIR', 'GLOBAL.XML');
```

The next example exports an OLAP 10g template and uses `IMPORT_XML` to validate it before an upgrade to 11g.

```
DECLARE

    exportOptClob clob;
    importOptClob clob;
    importClob    clob;
    exportClob    clob;

BEGIN

    -- Create a CLOB for the export options
    dbms_lob.createtemporary(exportOptClob, TRUE);
    dbms_cube.create_export_options(out_options_xml=>exportOptClob, suppress_
namespace=>TRUE, preserve_table_owners=>TRUE);

    -- Create a CLOB for the XML template
    dbms_lob.createtemporary(exportClob, TRUE);

    -- Create a CLOB for import options
    dbms_lob.createtemporary(importOptClob, TRUE);
    dbms_cube.create_import_options(out_options_xml=>importOptClob, validate_
only=>TRUE);

    -- Create a CLOB for the change log
    dbms_lob.createtemporary(importClob, TRUE);

    -- Enable 11g clients to access 10g metadata
    dbms_cube.initialize_cube_upgrade;

    -- Export metadata from an analytic workspace to a CLOB
```

```
dbms_cube.export_xml(object_ids=>'GLOBAL_AW', options_xml=>exportOptClob, out_
xml=>exportClob);

/* Import metadata from the CLOB. No objects are committed to the database
because the validate_only parameter of CREATE_IMPORT_OPTIONS is set to
TRUE.
*/

dbms_cube.import_xml(in_xml=>exportClob, options_xml=>importOptClob, out_
xml=>importClob);

-- Output the metadata changes
dbms_output.put_line('This is the validation log:');
dbms_output.put_line(importClob);

END;
/
```

The contents of `importClob` show that the XML is valid. Otherwise, error messages appear in the `<RootCommitResult>` element.

```
This is the validation log:
<?xml version="1.0" encoding="UTF-16"?>
<RootCommitResult>

</RootCommitResult>
```

For an example of `IMPORT_XML` within the context of an upgrade from 10g to 11g metadata, see ["Custom Upgrade"](#) on page 41-16.

## INITIALIZE\_CUBE\_UPGRADE Procedure

This procedure processes analytic workspaces created in Oracle OLAP 10g so they can be used by Oracle OLAP 11g clients. It processes all analytic workspaces in the current schema. Run this procedure once for each schema in which there are 10g analytic workspaces.

Without this processing step, 11g clients cannot connect to a database containing a 10g analytic workspace with subobjects of a dimension or cube having the same name. Additionally, some DBMS\_CUBE procedures and functions, such as EXPORT\_XML and EXPORT\_XML\_TO\_FILE, do not work on the 10g metadata.

After processing, OLAP 11g clients can connect and use the alternate names provided by INITIALIZE\_CUBE\_UPGRADE for the conflicting subobjects. OLAP 10g clients continue to use the original names.

INITIALIZE\_CUBE\_UPGRADE does not upgrade any OLAP 10g objects to OLAP 11g format.

See ["Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g"](#) on page 41-12.

### Syntax

```
DBMS_CUBE.INITIALIZE_CUBE_UPGRADE;
```

### Usage Notes

This procedure creates and populates a table named CUBE\_UPGRADE\_INFO. If it already exists, the table is truncated and repopulated.

While the 10g namespace allowed subobjects with the same name in the same dimension or cube, the 11g namespace does not. When INITIALIZE\_CUBE\_UPGRADE detects a name conflict among subobjects such as levels, hierarchies, and dimension attributes, it creates a row in CUBE\_UPGRADE\_INFO providing a new, unique name for each one. Rows may also be created for objects that do not require renaming; these rows are distinguished by a value of 0 or null in the CONFLICT column. Top-level objects, such as dimensions and cubes, are not listed.

You can edit the table using SQL INSERT and UPDATE if you want to customize the names of OLAP 10g objects on OLAP 11g clients.

The UPGRADE\_AW, EXPORT\_XML and EXPORT\_XML\_TO\_FILE procedures use the names specified in the NEW\_NAME column of the table to identify objects in CWM or OLAP standard form (AWXML) analytic workspaces, rather than the original names.

The following table describes the columns of CUBE\_UPGRADE\_INFO.

Column	Datatype	NULL	Description
OWNER	VARCHAR2	NOT NULL	Owner of the analytic workspace.
AW	VARCHAR2	NOT NULL	Name of the analytic workspace.
AWXML_ID	VARCHAR2	NOT NULL	Full logical name of the object requiring modification, in the form <i>simple_name</i> . [ <i>subtype_name</i> ]. <i>object_type</i> . For example, TIME.DIMENSION and PRODUCT.COLOR.ATTRIBUTE.
NEW_NAME	VARCHAR2	NOT NULL	The name the object will have in Oracle 11g after the upgrade.



Column	Datatype	NULL	Description
OBJECT_CLASS	VARCHAR2	--	DerivedMeasure for calculated measures, or empty for all other object types.
CONFLICT	NUMBER	--	Indicates the reason that the row was added to CUBE_UPGRADE_INFO: <ul style="list-style-type: none"> <li>■ 0: The object does not have a naming conflict but appears in the table for other reasons.</li> <li>■ 1: Two objects have the same name and would create a conflict in the OLAP 11g namespace. The object type (such as level or hierarchy) will be added to the names.</li> </ul>

## Examples

The following command creates and populates the CUBE\_UPGRADE\_INFO table:

```
EXECUTE dbms_cube.initialize_cube_upgrade;
```

The table shows that the OLAP 10g analytic workspace has a hierarchy and a level named MARKET\_SEGMENT, which will be renamed. The table also contains rows for calculated measures, but these objects do not require renaming: The value of CONFLICT is 0.

```
SELECT awxml_id, new_name, conflict FROM cube_upgrade_info;
```

AWXML_ID	NEW_NAME	CONFLICT
CUSTOMER.MARKET_SEGMENT.HIERARCHY	MARKET_SEGMENT_HIERARCHY	1
CUSTOMER.MARKET_SEGMENT.LEVEL	MARKET_SEGMENT_LEVEL	1
UNITS_CUBE.EXTENDED_COST.MEASURE	EXTENDED_COST	0
UNITS_CUBE.EXTENDED_MARGIN.MEASURE	EXTENDED_MARGIN	0
UNITS_CUBE.CHG_SALES_PP.MEASURE	CHG_SALES_PP	0
UNITS_CUBE.CHG_SALES_PY.MEASURE	CHG_SALES_PY	0
UNITS_CUBE.PCTCHG_SALES_PP.MEASURE	PCTCHG_SALES_PP	0
UNITS_CUBE.PCTCHG_SALES_PY.MEASURE	PCTCHG_SALES_PY	0
UNITS_CUBE.PRODUCT_SHARE.MEASURE	PRODUCT_SHARE	0
UNITS_CUBE.CHANNEL_SHARE.MEASURE	CHANNEL_SHARE	0
UNITS_CUBE.MARKET_SHARE.MEASURE	MARKET_SHARE	0
UNITS_CUBE.CHG_EXTMRGN_PP.MEASURE	CHG_EXTMRGN_PP	0
UNITS_CUBE.CHG_EXTMRGN_PY.MEASURE	CHG_EXTMRGN_PY	0
UNITS_CUBE.PCTCHG_EXTMRGN_PP.MEASURE	PCTCHG_EXTMRGN_PP	0
UNITS_CUBE.PCTCHG_EXTMRGN_PY.MEASURE	PCTCHG_EXTMRGN_PY	0
UNITS_CUBE.CHG_UNITS_PP.MEASURE	CHG_UNITS_PP	0
UNITS_CUBE.EXTMRGN_PER_UNIT.MEASURE	EXTMRGN_PER_UNIT	0
UNITS_CUBE.SALES_YTD.MEASURE	SALES_YTD	0
UNITS_CUBE.SALES_YTD_PY.MEASURE	SALES_YTD_PY	0
UNITS_CUBE.PCTCHG_SALES_YTD_PY.MEASURE	PCTCHG_SALES_YTD_PY	0
UNITS_CUBE.SALES_QTD.MEASURE	SALES_QTD	0
UNITS_CUBE.CHG_UNITS_PY.MEASURE	CHG_UNITS_PY	0

## REFRESH\_MVIEW Procedure

This procedure refreshes the data in a cube materialized view.

### Syntax

```
DBMS_CUBE.REFRESH_MVIEW (
    mvowner          IN  VARCHAR2,
    mvname           IN  VARCHAR2,
    method           IN  VARCHAR2          DEFAULT NULL,
    refresh_after_errors IN  BOOLEAN        DEFAULT FALSE,
    parallelism      IN  BINARY_INTEGER    DEFAULT 0,
    atomic_refresh   IN  BOOLEAN          DEFAULT FALSE,
    scheduler_job    IN  VARCHAR2          DEFAULT NULL,
    sam_parameters   IN  CLOB              DEFAULT NULL,
    nested           IN  BOOLEAN          DEFAULT FALSE );
```

### Parameters

**Table 41–13 REFRESH\_MVIEW Procedure Parameters**

Parameter	Description
mvowner	Owner of the cube materialized view.
mvname	Name of the cube materialized view.
method	<p>A full or a fast (partial) refresh. In a fast refresh, only changed rows are inserted in the cube and the affected areas of the cube are re-aggregated.</p> <p>You can specify a method for each cube in sequential order, or a single method to apply to all cubes. If you list more cubes than methods, then the last method applies to the additional cubes.</p> <ul style="list-style-type: none"> <li>■ C: Complete refresh clears all dimension values before loading. (Default)</li> <li>■ F: Fast refresh of a cube materialized view, which performs an incremental refresh and re-aggregation of only changed rows in the source table.</li> <li>■ ?: Fast refresh if possible, and otherwise a complete refresh.</li> <li>■ P: Recomputes rows in a cube materialized view that are affected by changed partitions in the detail tables.</li> <li>■ S: Fast solve of a compressed cube. A fast solve reloads all the detail data and re-aggregates only the changed values.</li> </ul> <p>See the "Usage Notes" for the BUILD procedure on page 41-26 for additional details.</p>
refresh_after_errors	<p>TRUE to roll back just the cube or dimension with errors, and then continue building the other objects.</p> <p>FALSE to roll back all objects in the build.</p>
parallelism	<p>Number of parallel processes to allocate to this job.</p> <p>See the "Usage Notes" for the BUILD procedure for additional details.</p>

**Table 41–13 (Cont.) REFRESH\_MVIEW Procedure Parameters**

Parameter	Description
atomic_refresh	TRUE prevents users from accessing intermediate results during a build. It freezes the current state of an analytic workspace at the beginning of the build to provide current sessions with consistent data. This option thaws the analytic workspace at the end of the build to give new sessions access to the refreshed data. If an error occurs during the build, then all objects are rolled back to the frozen state.  FALSE enables users to access intermediate results during an build.
scheduler_job	Any text identifier for the job, which will appear in the log table. The string does not need to be unique.
sam_parameters	None.
nested	TRUE performs nested refresh operations for the specified set of cube materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.  All objects must reside in a single analytic workspace.

## Usage Notes

REFRESH\_MVIEW changes *mvname* to the name of the cube, then passes the cube name and all parameters to the BUILD procedure. Thus, you can use the BUILD procedure to refresh a cube materialized view. See the ["BUILD Procedure"](#) on page 41-19 for additional information about the parameters.

## Examples

The following example uses the default settings to refresh a cube materialized view named CB\$F WEEK\_PSCAT\_SALES.

```
SET serverout ON format wrapped
```

```
EXECUTE dbms_cube.refresh_mview('SH', 'CB$F WEEK_PSCAT_SALES');
```

The next example changes the refresh method to use fast refresh if possible, continue refreshing after an error, and use two parallel processes.

```
EXECUTE dbms_cube.refresh_mview('SH', 'CB$F WEEK_PSCAT_SALES', '?', TRUE, 2);
```

After successfully refreshing the cube materialized view, REFRESH\_MVIEW returns a message like the following:

```
Completed refresh of cube mview "SH"."CB$F WEEK_PSCAT_SALES" at 20081114
15:04:46.370.
```

## UPGRADE\_AW Procedure

This procedure creates an Oracle OLAP 11g analytic workspace from a copy of the metadata contained in an OLAP 10g analytic workspace. The original OLAP 10g analytic workspace is not affected and can exist at the same time and in the same schema as the OLAP 11g analytic workspace.

UPGRADE\_AW automatically runs INITIALIZE\_CUBE\_UPGRADE if the CUBE\_UPGRADE\_INFO table does not exist. If it does exist, then UPGRADE\_AW does not overwrite it, thus preserving any changes you made to the table.

See ["Upgrading Analytic Workspaces From OLAP 10g to OLAP 11g"](#) on page 41-12.

### Syntax

```
DBMS_CUBE.UPGRADE_AW
  (sourceaw          IN VARCHAR2,
   destaw            IN VARCHAR2,
   upgoptions        IN CLOB DEFAULT NULL);
```

### Parameters

**Table 41-14 UPGRADE\_AW Procedure Parameters**

Parameter	Description
sourceaw	The name of a 10g analytic workspace.
destaw	A new name for the generated 11g analytic workspace. It cannot be the same as <i>sourceaw</i> .
upgoptions	<p>One or more of these upgrade options, as a string in the form 'OPTION=VALUE'. Separate multiple options with commas.</p> <ul style="list-style-type: none"> <li> <p>■ PRESERVE_TABLE_OWNERS:</p> <p>YES preserves the original source table mappings. Use this option when creating an OLAP 11g analytic workspace in a different schema from the 10g analytic workspace, and you want the new objects mapped to tables in the original schema. (Default)</p> <p>NO removes the schema owner from the source table mappings. Use this option when creating an OLAP 11g analytic workspace in a different schema from the 10g analytic workspace, and you want the new objects mapped to tables in the destination schema.</p> </li> <li> <p>■ RENAME_TABLE: The name of a table that specifies new names for objects as they are created in OLAP 11g format. These changes are in addition to those specified by the INITIALIZE_CUBE_UPGRADE procedure. See <a href="#">"CREATE_IMPORT_OPTIONS Procedure"</a> on page 41-34 for information about creating a rename table.</p> </li> </ul>

### Examples

This example upgrades an OLAP 10g analytic workspace named GLOBAL10 to an OLAP 11g analytic workspace named GLOBAL11, using a rename table named MY\_OBJECT\_MAP:

```
BEGIN
  -- Upgrade the analytic workspace
  dbms_cube.upgrade_aw(sourceaw =>'GLOBAL10', destaw => 'GLOBAL11', upgoptions =>
```

```
'RENAME_TABLE=MY_OBJECT_MAP');  
  
-- Load and aggregate the data  
dbms_cube.build(script=>'UNITS_CUBE, PRICE_AND_COST_CUBE');  
  
END;  
/
```

## VALIDATE\_XML Procedure

This procedure checks the XML to assure that it is valid without committing the results to the database. It does not create an analytic workspace.

### Syntax

```
DBMS_CUBE.VALIDATE_XML
  (dirname          IN  VARCHAR2,
   filename         IN  VARCHAR2 );

DBMS_CUBE.VALIDATE_XML
  (in_xml          IN  CLOB );
```

### Parameters

**Table 41–15** VALIDATE\_XML Procedure Parameters

Parameter	Description
dirname	The case-sensitive name of a database directory.
filename	The name of a file containing an XML template.
IN_XML	The name of a CLOB containing an XML template.

### Usage Notes

You should always load a template into the same version and release of Oracle Database as the one used to generate the template. The XML may not be valid if it was generated by a different release of the software.

### Example

This example reports a problem in the schema:

```
EXECUTE dbms_cube.validate_xml('UPGRADE_DIR', 'MYGLOBAL.XML');
BEGIN dbms_cube.validate_xml('UPGRADE_DIR', 'MYGLOBAL.XML'); END;

*
ERROR at line 1:
ORA-37162: OLAP error
'GLOBAL.PRICE_CUBE.$AW_ORGANIZATION': XOQ-01950: The AWCubeOrganization for
cube "GLOBAL.PRICE_CUBE" contains multiple BuildSpecifications with the same
name.
'GLOBAL.UNITS_CUBE.$AW_ORGANIZATION': XOQ-01950: The AWCubeOrganization for
cube "GLOBAL.UNITS_CUBE" contains multiple BuildSpecifications with the same
name.
XOQ-01400: invalid metadata objects
ORA-06512: at "SYS.DBMS_CUBE", line 411
ORA-06512: at "SYS.DBMS_CUBE", line 441
ORA-06512: at "SYS.DBMS_CUBE", line 501
ORA-06512: at "SYS.DBMS_CUBE", line 520
ORA-06512: at line 1
```

After the problems are corrected, the procedure reports no errors:

```
EXECUTE dbms_cube.validate_xml('UPGRADE_DIR', 'MYGLOBAL.XML');

PL/SQL procedure successfully completed.
```

This example loads an XML template into a temporary CLOB, then validates it. The script is named GLOBAL.XML, and it is located in a database directory named XML\_DIR.

```
DEFINE xml_file = 'GLOBAL.XML';

SET ECHO ON;
SET SERVEROUT ON;

DECLARE
    xml_file      BFILE := bfilename('XML_DIR', '&xml_file');
    in_xml        CLOB;
    out_xml       CLOB := NULL;
    dest_offset   INTEGER := 1;
    src_offset    INTEGER := 1;
    lang_context  INTEGER := 0;
    warning       INTEGER;
BEGIN
    -- Setup the clob from a file
    DBMS_LOB.CREATETEMPORARY(in_xml, TRUE);
    DBMS_LOB.OPEN(in_xml, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(xml_file, DBMS_LOB.FILE_READONLY);
    DBMS_LOB.LOADCLOBFROMFILE(in_xml, xml_file, DBMS_LOB.LOBMAXSIZE,
        dest_offset, src_offset, 0, lang_context, warning);

    -- Validate the xml
    DBMS_CUBE.VALIDATE_XML(in_xml);
END;
/
```





---

---

## DBMS\_CUBE\_ADVISE

DBMS\_CUBE\_ADVISE contains subprograms for evaluating cube materialized views to support log-based fast refresh and query rewrite.

This chapter contains the following topics:

- [Using DBMS\\_CUBE\\_ADVISE](#)
- [Summary of DBMS\\_CUBE\\_ADVISE Subprograms](#)

**See Also:** *Oracle OLAP User's Guide* for information about cube materialized views

---

## Using DBMS\_CUBE\_ADVISE

- [Security Model](#)

## Security Model

The `MV_CUBE_ADVICE` function requires the `ADVISOR` privilege.

## Summary of DBMS\_CUBE\_ADVISE Subprograms

**Table 42-1 Summary of DBMS\_CUBE\_ADVISE Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">MV_CUBE_ADVICE Function</a> on page 42-5	Evaluates the metadata of a cube materialized view and generates recommendations for constraints, SQL dimension objects, and materialized view logs to support a broad range of query rewrite and fast refresh opportunities.
<a href="#">SET_CNS_EXCEPTION_LOG Procedure</a> on page 42-8	Identifies the name of an exception log used in validated constraints generated by MV_CUBE_ADVICE.
<a href="#">TRACE Procedure</a> on page 42-9	Displays or suppresses diagnostic messages for MV_CUBE_ADVICE.

## MV\_CUBE\_ADVICE Function

This table function evaluates the metadata for a specified cube materialized view. It generates recommendations and returns them as a SQL result set. These SQL statements can be used to create constraints, SQL dimension objects, and materialized view logs that allow the broadest range of query rewrite transformations and log-based fast refresh of the cube materialized view.

### Syntax

```
DBMS_CUBE_ADVICE.MV_CUBE_ADVICE (
    owner          IN  VARCHAR2  DEFAULT USER,
    mvname         IN  VARCHAR2,
    reqtype        IN  VARCHAR2  DEFAULT '0',
    validate       IN  NUMBER    DEFAULT 0)
RETURN COAD_ADVICE_T PIPELINED;
```

### Parameters

**Table 42–2 MV\_CUBE\_ADVICE Function Parameters**

Parameter	Description
owner	Owner of the cube materialized view
mvname	Name of the cube, such as UNITS_CUBE, or the cube materialized view, such as CB\$UNITS_CUBE
reqtype	Type of advice to generate: <ul style="list-style-type: none"> <li>■ 0: All applicable advice types</li> <li>■ 1: Column NOT NULL constraints</li> <li>■ 2: Primary key constraints</li> <li>■ 3: Foreign key constraints</li> <li>■ 4: Relational dimension objects</li> <li>■ 5: Cube materialized view logs with primary key</li> </ul>
validate	Validation option: <ul style="list-style-type: none"> <li>■ 0: Validate the constraints</li> <li>■ 1: Do not validate the constraints</li> </ul>

### Returns

A table of type COAD\_ADVICE\_T, consisting of a set of rows of type COAD\_ADVICE\_REC. [Table 42–3](#) describes the columns.

**Table 42–3 MV\_CUBE\_ADVICE Return Values**

Column	Data Type	Description
OWNER	VARCHAR2 (30)	Owner of the dimensional object identified in APIOBJECT.
APIOBJECT	VARCHAR2 (30)	Name of a cube enhanced with materialized view capabilities, such as UNITS_CUBE.
SQLOBJOWN	VARCHAR2 (30)	Owner of the relational object identified in SQLOBJECT.
SQLOBJECT	VARCHAR2 (65)	Name of the master table, such as UNITS_FACT, or the cube materialized view, such as CB\$UNITS_CUBE.

**Table 42–3 (Cont.) MV\_CUBE\_ADVICE Return Values**

Column	Data Type	Description
ADVICETYPE	NUMBER(38,0)	Type of recommendation: <ul style="list-style-type: none"> <li>■ 1: Create NOT NULL constraints on the foreign key columns</li> <li>■ 2: Create primary key constraints on the master table</li> <li>■ 3: Create primary key constraints on the master view</li> <li>■ 4: Create foreign key constraints on the master table</li> <li>■ 5: Create foreign key constraints on the master view</li> <li>■ 6: Create relational dimensions on the master dimension tables</li> <li>■ 7: Create a materialized view log</li> <li>■ 8: Compile the materialized view</li> </ul>
DISPOSITION	CLOB	Pre-existing conditions that conflict with the recommendations and should be resolved before SQLTEXT can be executed.
SQLTEXT	CLOB	SQL statement that implements the recommendation.
DROPTTEXT	CLOB	SQL statement that reverses SQLTEXT. Pre-existing conditions may prevent these statements from restoring the schema to its previous state.

## Usage Notes

This function is available in Analytic Workspace Manager as the Materialized View Advisor, which will generate a SQL script with the recommendations.

You can query the returned rows the same as any other table, as shown in the example.

MV\_CUBE\_ADVICE generates unique object names each time it is called. You should execute the function once, capture the results, and work with those SQL statements.

Take care when dropping database objects. If a table already has a materialized view log, it will have the same name used in the SQL DROP MATERIALIZED VIEW LOG statement in the DROPTTEXT column. You should avoid inadvertently dropping materialized view logs, especially when they may be used for remote data replication.

## Examples

The following query displays the SQL statements recommended by MV\_CUBE\_ADVICE. UNITS\_FACT is the master table for UNITS\_CUBE, and MV\_CUBE\_ADVICE generates an ALTER TABLE command to add primary key constraints.

It also generates an ALTER MATERIALIZED VIEW command to compile the CB\$UNITS\_CUBE cube materialized view.

```
SQL> SELECT apiobject, sqlobject, sqltext
       FROM TABLE(dbms_cube_advise.mv_cube_advise('GLOBAL', 'CB$UNITS_CUBE'));
```

```
APIOBJECT      SQLOBJECT      SQLTEXT
-----
UNITS_CUBE    UNITS_FACT     alter table "GLOBAL"."UNITS_FACT" add constr
int "COAD_PK000208" PRIMARY KEY ("CHANNEL_ID"
```

```
        , "ITEM_ID", "SHIP_TO_ID", "MONTH_ID") rely d  
        isable novalidate  
  
UNITS_CUBE    CB$UNITS_CUBE    alter materialized view "GLOBAL"."CB$UNITS_CU  
              BE" compile
```

## SET\_CNS\_EXCEPTION\_LOG Procedure

This procedure identifies the name of an exception log used in validated constraints generated by MV\_CUBE\_ADVICE.

### Syntax

```
DBMS_CUBE_ADVICE.SET_CNS_EXCEPTION_LOG (
    exceptlogtab    IN    VARCHAR2    DEFAULT user.EXCEPTIONS);
```

### Parameters

**Table 42–4 SET\_CNS\_EXCEPTION\_LOG Procedure Parameters**

Parameter	Description
exceptlogtab	The name of an existing exception log.

### Usage Notes

To create an exception log, use the `utlexcpt.sql` or the `utlexpt1.sql` script before executing `SET_CNS_EXCEPTION_LOG`.

The `validate` parameter of `MV_CUBE_ADVICE` must be set to 1.

### Examples

The `utlexcpt.sql` script creates a table named `EXCEPTIONS`, and the `SET_CNS_EXCEPTION_LOG` procedure identifies it as the exception log for `MV_CUBE_ADVICE`. The `ALTER TABLE` statement now includes the clause `VALIDATE EXCEPTIONS INTO "GLOBAL"."EXCEPTIONS"`.

```
SQL> @utlexcpt
Table created.
```

```
SQL> EXECUTE dbms_cube_advice.set_cns_exception_log;
PL/SQL procedure successfully completed.
```

```
SQL> SELECT apiobject, sqlobject, advicetype type, sqltext
       FROM TABLE(
           dbms_cube_advice.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE', '2', 1));
```

APIOBJECT	SQLOBJECT	TYPE	SQLTEXT
UNITS_CUBE	UNITS_FACT	2	alter table "GLOBAL"."UNITS_FACT" add constraint "COAD_PK000219" PRIMARY KEY ("CHANNEL_ID", "ITEM_ID", "SHIP_TO_ID", "MONTH_ID") norely enable validate exceptions into "GLOBAL"."EXCEPTIONS"
UNITS_CUBE	CB\$UNITS_CUBE	8	alter materialized view "GLOBAL"."CB\$UNITS_CUBE" compile



## TRACE Procedure

This procedure turns on and off diagnostic messages to server output for the MV\_CUBE\_ADVISE function.

### Syntax

```
DBMS_CUBE_ADVISE.TRACE (
    diaglevel          IN BINARY_INTEGER DEFAULT 0);
```

### Parameters

**Table 42–5 TRACE Procedure Parameters**

Parameter	Description
diaglevel	0 to turn tracing off, or 1 to turn tracing on.

### Examples

The following example directs the diagnostic messages to server output. The SQL\*Plus SERVEROUTPUT setting displays the messages.

```
SQL> SET SERVEROUT ON FORMAT WRAPPED
SQL> EXECUTE dbms_cube_advise.trace(1);
DBMS_COAD_DIAG: Changing diagLevel from [0] to [1]
```

PL/SQL procedure successfully completed.

```
SQL> SELECT sqlobject, sqltext, droptext
FROM TABLE(
    dbms_cube_advise.mv_cube_advice('GLOBAL', 'CB$UNITS_CUBE'))
WHERE apiobject='UNITS_CUBE';
```

SQLOBJECT	SQLTEXT	DROPTEXT
UNITS_FACT	alter table "GLOBAL"."UNITS_FACT" add constraint "COAD_PK000222" PRIMARY KEY ("CHANNEL_ID", "ITEM_ID", "SHIP_TO_ID", "MONTH_ID")	alter table "GLOBAL"."UNITS_FACT" drop constraint "COAD_PK000222" cascade
CB\$UNITS_CUBE	alter materialized view "GLOBAL"."CB\$UNITS_CUBE" compile	alter materialized view "GLOBAL"."CB\$UNITS_CUBE" compile

```
20070706 07:25:27.462780000 DBMS_COAD_DIAG NOTE: Parameter mvOwner : GLOBAL
20070706 07:25:27.462922000 DBMS_COAD_DIAG NOTE: Parameter mvName : CB$UNITS_CUBE
20070706 07:25:27.462967000 DBMS_COAD_DIAG NOTE: Parameter factTab : .
20070706 07:25:27.463011000 DBMS_COAD_DIAG NOTE: Parameter cubeName : UNITS_CUBE
20070706 07:25:27.463053000 DBMS_COAD_DIAG NOTE: Parameter cnsState : rely disable novalidate
20070706 07:25:27.463094000 DBMS_COAD_DIAG NOTE: Parameter NNState : disable novalidate
20070706 07:25:27.462368000 DBMS_COAD_DIAG NOTE: Begin NN:
20070706 07:25:27.833530000 DBMS_COAD_DIAG NOTE: End NN:
20070706 07:25:27.833620000 DBMS_COAD_DIAG NOTE: Begin PK:
20070706 07:25:28.853418000 DBMS_COAD_DIAG NOTE: End PK:
20070706 07:25:28.853550000 DBMS_COAD_DIAG NOTE: Begin FK:
20070706 07:25:28.853282000 DBMS_COAD_DIAG NOTE: End FK:
20070706 07:25:28.853359000 DBMS_COAD_DIAG NOTE: Begin RD:
20070706 07:25:29.660471000 DBMS_COAD_DIAG NOTE: End RD:
```

## TRACE Procedure

---

```
20070706 07:25:29.661363000 DBMS_COAD_DIAG NOTE: Begin CM:  
20070706 07:25:29.665106000 DBMS_COAD_DIAG NOTE: End   CM:
```

```
SQL> EXECUTE dbms_cube_advise.trace(0);  
DBMS_COAD_DIAG: Changing diagLevel from [1] to [0]
```

```
PL/SQL procedure successfully completed.
```

---

---

## DBMS\_CUBE\_LOG

DBMS\_CUBE\_LOG contains subprograms for creating and managing logs for cubes and cube dimensions.

**See Also:** *Oracle OLAP User's Guide* regarding use of the OLAP option to support business intelligence and analytical applications

This chapter contains the following topics:

- [Using DBMS\\_CUBE\\_LOG](#)
- [Summary of DBMS\\_CUBE\\_LOG Subprograms](#)

## Using DBMS\_CUBE\_LOG

DBMS\_CUBE\_LOG manages several logs. These logs enable you to track the progress of long running processes, then use the results to profile performance characteristics. They provide information to help you diagnose and remedy problems that may occur during development and maintenance of a cube: Hierarchies that are improperly structured in the relational source tables, records that fail to load, or data refreshes that take too long to complete. They also help diagnose performance problems in querying cubes.

Analytic Workspace Manager creates the logs automatically using the default names and types. It also disables the logs when Analytic Workspace Manager is closed. To use the same logs outside of Analytic Workspace Manager, you must first enable them. Alternatively, you can create and manage different logs for use outside of Analytic Workspace Manager.

This section contains the following topics:

- [Logging Types](#)
- [Logging Targets](#)
- [Verbosity Levels](#)
- [Security Model](#)
- [Creating Cube Logs](#)
- [Cube Build Log](#)
- [Cube Dimension Compile Log](#)
- [Cube Operations Log](#)
- [Cube Rejected Records Log](#)

## Logging Types

Several logs are available, each one dedicated to storing messages of a particular type. You may use all of them or only those that you find particularly valuable. The logs and their contents are described later in this topic.

- [Cube Build Log](#)
- [Cube Dimension Compile Log](#)
- [Cube Operations Log](#)
- [Cube Rejected Records Log](#)

DBMS\_CUBE\_LOG provides functions that return the binary integer for each log type. You can produce more readable code by using these functions instead of integers for the argument values of other DBMS\_CUBE\_LOG procedures and functions. Refer to these descriptions:

- [TYPE\\_BUILD Function](#)
- [TYPE\\_DIMENSION\\_COMPILE Function](#)
- [TYPE\\_OPERATIONS Function](#)
- [TYPE\\_REJECTED\\_RECORDS Function](#)

## Logging Targets

The `TABLE_CREATE` procedure creates database tables for storing the logs. Using the `ENABLE` procedure, you can create additional targets with changes in the destination or logging level. For example, you might target the Cube Operations log to both a table and a disk file. These are the available targets:

- Disk file
- LOB
- Database table
- Trace file

See "[ENABLE Procedure](#)" on page 43-17 for more information about creating multiple targets.

`DBMS_CUBE_LOG` provides functions that return the binary integer for each target type. You can produce more readable code by using these functions instead of integers for the argument values of other `DBMS_CUBE_LOG` procedures and functions. Refer to these descriptions:

- [TARGET\\_FILE Function](#)
- [TARGET\\_LOB Function](#)
- [TARGET\\_TABLE Function](#)
- [TARGET\\_TRACE Function](#)

## Verbosity Levels

You can decide how much information is recorded in a log. You may want fewer details when leaving a job to run overnight than when you are monitoring the success of a new build. You can choose from these verbosity levels. Each level adds to the preceding level.

- **LOWEST:** Logs the status of each command used to build the cube dimensions and cubes, the use of slave processes, and summary records. This is the basic logging level.
- **LOW:** Logs messages from the OLAP engine, such as start and finish records for SQL Import, Aggregate, and Update.
- **MEDIUM:** Logs messages at the level used by Analytic Workspace Manager.
- **HIGH:** Logs messages that provide tuning information, such as composite lengths, partitioning details, object sizes, and aggregation work lists. This level is intended for use by Oracle Field Services.
- **HIGHEST:** Logs debugging messages and other information typically sent to a trace file. This level is intended for use by Oracle Support Services.

DBMS\_CUBE\_LOG provides functions that return the binary integer for each verbosity level. You can produce more readable code by using these functions instead of integers for the argument values of other DBMS\_CUBE\_LOG procedures and functions. Refer to these descriptions:

- [LEVEL\\_LOWEST Function](#)
- [LEVEL\\_LOW Function](#)
- [LEVEL\\_MEDIUM Function](#)
- [LEVEL\\_HIGH Function](#)
- [LEVEL\\_HIGHEST Function](#)

## Security Model

The `TABLE_CREATE` procedure requires the `CREATE TABLE` privilege.



## Creating Cube Logs

To store logging information in a database table, you must create that table using the TABLE\_CREATE procedure. Cube Build logs are always stored in tables. The ENABLE procedure creates the other target types for the other logs.

### To create a Cube Build log:

- Execute the TABLE\_CREATE procedure.

The following command creates a Cube Build log with the default name of CUBE\_BUILD\_LOG:

```
EXECUTE dbms_cube_log.table_create(dbms_cube_log.type_build);
```

### To create a Cube Dimension Compile log, Cube Operations log, or Cube Rejected Records log with a database table target:

1. Execute the TABLE\_CREATE procedure to create the table.
2. Execute the ENABLE procedure to begin logging.

These commands create and enable a Cube Operations table with the default name of CUBE\_OPERATIONS\_LOG and the default verbosity level:

```
EXECUTE dbms_cube_log.table_create(dbms_cube_log.type_operations);  
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations);
```

### To create a Cube Dimension Compile log, Cube Operations log, or Cube Rejected Records log with a trace file, disk file, or LOB target:

- Execute the ENABLE procedure.

This command enables the Cube Rejected Records log, sets verbosity to the lowest level, and directs the output to a disk file named rejects.log in the WORK\_DIR database directory:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_rejected_records, -  
    dbms_cube_log.target_file, dbms_cube_log.level_lowest, -  
    'WORK_DIR/rejects.log');
```

## Cube Build Log

The Cube Build log provides information about what happened during a build. Use this log to determine whether the build produced the results you were expecting, and if not, why not.

The contents of the Cube Build log is refreshed continuously during a build. You can query the log at any time to evaluate the progress of the build and to estimate the time to completion.

The default name of the Cube Build log is CUBE\_BUILD\_LOG. The following table describes its contents.

---



---

**Note:** To manage a Cube Build log, use only the TABLE\_CREATE and VERSION procedures.

---



---

Column	Datatype	NULL	Description
BUILD_ID	NUMBER	--	A unique sequence number for the build. The same number is used for slave processes in a parallel build.
SLAVE_NUMBER	NUMBER	--	A counter for slave processes in a parallel build: 0 is the master process, 1 is the first slave, 2 is the second slave, and so forth.
STATUS	VARCHAR2 (10)	--	The current status of the command: STARTED, COMPLETED, FAILED, or WORKING.
COMMAND	VARCHAR2 (25)	--	The name of the command being executed, such as BUILD, LOAD, and SOLVE.
BUILD_OBJECT	VARCHAR2 (500)	--	The name of the cube or cube dimension being processed.
BUILD_OBJECT_TYPE	VARCHAR2 (10)	--	The type of object: CUBE, DIMENSION, or BUILD.
OUTPUT	CLOB	--	Information structured like an XML document about the command, or NULL when there is no additional information, such as for a STARTED row.
AW	VARCHAR2 (30)	--	The name of the analytic workspace that contains the objects of the build.
OWNER	VARCHAR2 (30)		The owner of the analytic workspace and all the objects of the build.
PARTITION	VARCHAR2 (10)	--	The name of the partition being processed, or NULL when the current operation does not correspond to a partition.
SCHEDULER_JOB	VARCHAR2 (100)	--	A user-specified string to identify the build.
TIME	TIMESTAMP (6)	--	The time the row is added to the table.
BUILD_SCRIPT	CLOB	--	The cube build script. Populated only in rows where COMMAND is BUILD.
BUILD_TYPE	VARCHAR2 (22)	--	The origin of the build: DBMS_CUBE, DBMS_MVIEW, JAVA, or SLAVE.
COMMAND_DEPTH	NUMBER (2)	--	The nesting depth of the command. For example, COMPILE HIERARCHIES is a component step of COMPILE, so if COMPILE has a depth of 1, then COMPILE HIERARCHIES has a depth of 2.

---

<b>Column</b>	<b>Datatype</b>	<b>NULL</b>	<b>Description</b>
BUILD_SUB_OBJECT	VARCHAR2 (30)	--	The name of a subobject being processed, such as a measure that does not inherit the aggregation rules of the cube.
REFRESH_METHOD	VARCHAR2 (1)	--	The refresh method, such as C or F, that is associated with the current command. The refresh method is important only for the CLEAR step.
SEQ_NUMBER	NUMBER	--	Not currently used.
COMMAND_NUMBER	NUMBER	--	The sequence number of the command in the current process, which can be used to distinguish the same command on different objects. For example, a LOAD on PRODUCT and a LOAD on TIME.
IN_BRANCH	NUMBER (1)	--	Not currently used.
COMMAND_STATUS_NUMBER	NUMBER	--	Identifies the sequence number of all rows for a particular command. For example, a particular command might be represented by four rows: The first row has a status of STARTED and the last row has a status of COMPLETED. This column is used for sorting.

---

## Cube Dimension Compile Log

When solving a cube, OLAP checks the dimension hierarchies to make sure they are valid. Errors that occur during this validation are written to the Cube Dimension Compile log. The checks include:

- **Circularity:** Hierarchies are defined by parent-child relations among dimension members. Circularity occurs when a dimension member is specified as its own ancestor or descendant.
- **Hierarchy type:** Hierarchies can be level based or value based. You can define a cube so that only level-based hierarchies are valid, such as a cube materialized view.
- **Level options:** Level-based hierarchies can be regular, ragged, or skip level. You can define a dimension so that only regular hierarchies are valid, such as a Time dimension.

The default name of the Cube Dimension Compile log is CUBE\_DIMENSION\_COMPILE. The following table describes its contents.

Column	Datatype	NULL	Description
ID	NUMBER	--	Current operation identifier
SEQ_NUMBER	NUMBER	--	Sequence number in the Cube Build log
ERROR#	NUMBER (8)	NOT NULL	Number of the error being reported
ERROR_MESSAGE	VARCHAR2 (2000)	--	Error message associated with the error
DIMENSION	VARCHAR2 (100)	--	Name of the dimension being compiled
DIMENSION_MEMBER	VARCHAR2 (100)	--	Faulty dimension member
MEMBER_ANCESTOR	VARCHAR2 (100)	--	Parent of DIMENSION_MEMBER
HIERARCHY1	VARCHAR2 (100)	--	First hierarchy involved in the error
HIERARCHY2	VARCHAR2 (100)	--	Second hierarchy involved in the error
ERROR_CONTEXT	CLOB	--	Additional information about the error

## Cube Operations Log

The Cube Operations log contains messages and debugging information for all OLAP engine events. You can track current operations at a very detailed level. Using the `SQL_ID` column, you can join the Cube Operations log to dynamic performance views such as `V$SQL`, `V$SESSION`, and `V$SESSION_LONGOPS` to see cube operations in the context of other database operations such as I/O Wait and CPU.

The default name of the Cube Operations log is `CUBE_OPERATIONS_LOG`. The following table describes its contents.

Column	Datatype	NULL	Description
<code>INST_ID</code>	NUMBER	NOT NULL	Instance identifier
<code>SID</code>	NUMBER	NOT NULL	Session identifier
<code>SERIAL#</code>	NUMBER	NOT NULL	Session serial number
<code>USER#</code>	NUMBER	NOT NULL	User identifier
<code>SQL_ID</code>	VARCHAR2 (13)	--	Executing SQL statement identifier
<code>JOB</code>	NUMBER	--	Job identifier
<code>ID</code>	NUMBER	--	Current operation identifier
<code>PARENT_ID</code>	NUMBER	--	Parent operation identifier
<code>SEQ_NUMBER</code>	NUMBER	--	Sequence number in the Cube Build log
<code>TIME</code>	TIMESTAMP (6) WITH TIME ZONE	NOT NULL	Time the record was added to the Cube Operations log
<code>LOG_LEVEL</code>	NUMBER (4)	NOT NULL	Verbosity level of the record, as specified by the <code>DBMS_CUBE_LOG.ENABLE</code> procedure.
<code>DEPTH</code>	NUMBER (4)	--	Nesting depth of the record. For example, a level of 0 indicates that the operation and suboperation are not nested within other operations and suboperations.
<code>OPERATION</code>	VARCHAR2 (15)	NOT NULL	Current operation, such as <code>AGGREGATE</code> , <code>ROWSOURCE</code> , or <code>SQLIMPORT</code> .
<code>SUBOPERATION</code>	VARCHAR2 (20)	--	Current suboperation, such as <code>Loading</code> or <code>Import</code>
<code>STATUS</code>	VARCHAR2 (10)	NOT NULL	Current status of the operation, such as <code>START</code> , <code>TRACE</code> , <code>COMPLETED</code> , or <code>Failed</code> .
<code>NAME</code>	VARCHAR2 (20)	NOT NULL	Name of the record, such as <code>ROWS LOADED</code> , <code>AVE_ROW_LEN</code> , and <code>PAGEPOOLSIZ</code>
<code>VALUE</code>	VARCHAR2 (4000)	--	Value of <code>NAME</code>
<code>DETAILS</code>	CLOB	--	Additional information about <code>NAME</code> .

## Cube Rejected Records Log

The Cube Rejected Records log contains a summary of the loader job and any records that were rejected because they did not meet the expected format.

A single row in the source table may have errors in more than one field. Each field will generate an error in log, resulting in multiple rows with the same rowid in the SOURCE\_ROW column.

The default name of the Cube Rejected Records log is CUBE\_REJECTED\_RECORDS. The following table describes its contents.

Column	Datatype	NULL	Description
ID	NUMBER	--	Current operation identifier
SEQ_NUMBER	NUMBER	--	Sequence number in the Cube Build log
ERROR#	NUMBER (8)	NOT NULL	Number of the error triggered by the record
ERROR_MESSAGE	VARCHAR2	--	Error message associated with the error
RECORD#	NUMBER (38)	--	Input record number
SOURCE_ROW	ROWID	--	Rowid of the row in the source table; null when the source is a view or a query

## Summary of DBMS\_CUBE\_LOG Subprograms

**Table 43-1 DBMS\_CUBE\_LOG Subprograms**

Subprogram	Description
<a href="#">DEFAULT_NAME Function</a> on page 43-15	Returns the default table names of the various log types.
<a href="#">DISABLE Procedure</a> on page 43-16	Turns logging off for the duration of a session.
<a href="#">ENABLE Procedure</a> on page 43-17	Turns on logging for the duration of a session, redirects logging to additional output types, and changes the verbosity level in the logs.
<a href="#">FLUSH Procedure</a> on page 43-19	Forces all buffered messages to be written to the logs.
<a href="#">GET_LOG Procedure</a> on page 43-20	Returns the current settings for the level and location of a particular log.
<a href="#">GET_LOG_SPEC Function</a> on page 43-22	Retrieves a description of all active logs.
<a href="#">GET_PARAMETER Function</a> on page 43-23	Returns the current values of the options that control various aspects of logging.
<a href="#">LEVEL_HIGH Function</a> on page 43-24	Returns the integer value of the high verbosity level.
<a href="#">LEVEL_HIGHEST Function</a> on page 43-25	Returns the integer value of the highest verbosity level.
<a href="#">LEVEL_LOW Function</a> on page 43-26	Returns the integer value of the low verbosity level.
<a href="#">LEVEL_LOWEST Function</a> on page 43-27	Returns the integer value of the lowest verbosity level.
<a href="#">LEVEL_MEDIUM Function</a> on page 43-28	Returns the integer value of the medium verbosity level.
<a href="#">SET_LOG_SPEC Procedure</a> on page 43-29	Sets all logging to the values specified in the input string.
<a href="#">SET_PARAMETER Procedure</a> on page 43-30	Sets options that control various aspects of logging.
<a href="#">TABLE_CREATE Procedure</a> on page 43-32	Creates the table targets for the OLAP logs.
<a href="#">TARGET_FILE Function</a> on page 43-33	Returns the integer value of a disk file target.
<a href="#">TARGET_LOB Function</a> on page 43-34	Returns the integer value of a LOB target.
<a href="#">TARGET_TABLE Function</a> on page 43-35	Returns the integer value of a database table target.
<a href="#">TARGET_TRACE Function</a> on page 43-36	Returns the integer value of a trace file target.
<a href="#">TYPE_BUILD Function</a> on page 43-37	Returns the integer value of the Cube Build log.
<a href="#">TYPE_DIMENSION_COMPILE Function</a> on page 43-38	Returns the integer value of the Cube Dimension Compile log.

**Table 43-1 (Cont.) DBMS\_CUBE\_LOG Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">TYPE_OPERATIONS Function</a> on page 43-39	Returns the integer value of the Cube Operations log.
<a href="#">TYPE_REJECTED_RECORDS Function</a> on page 43-40	Returns the integer value of the Cube Rejected Records log.
<a href="#">VERSION Function</a> on page 43-41	Returns the version number of a specific log table or the current version number of a specific log type.



## DEFAULT\_NAME Function

This function returns the default table names of the various log types.

### Syntax

```
DBMS_CUBE_LOG.DEFAULT_NAME (
    LOG_TYPE          IN  BINARY_INTEGER  DEFAULT TYPE_OPERATIONS)
RETURN VARCHAR2;
```

### Parameters

**Table 43–2** *DEFAULT\_NAME Function Parameters*

Parameter	Description
log_type	One of the following log types: <ul style="list-style-type: none"> <li>■ 1: TYPE_OPERATIONS</li> <li>■ 2: TYPE_REJECTED_RECORDS</li> <li>■ 3: TYPE_DIMENSION_COMPILE</li> <li>■ 4: TYPE_BUILD</li> </ul> See <a href="#">"Logging Types"</a> on page 43-3.

### Returns

The default table name of the specified log type.

### Examples

This example returns the default name of the Cube Operations log:

```
SELECT dbms_cube_log.default_name FROM dual;
```

```
DEFAULT_NAME
-----
CUBE_OPERATIONS_LOG
```

The next example returns the default name of the Cube Rejected Records log:

```
select dbms_cube_log.default_name(dbms_cube_log.type_rejected_records) -
       "Default Name" from dual;
```

```
Default Name
-----
CUBE_REJECTED_RECORDS
```

## DISABLE Procedure

This procedure turns logging off for the duration of a session, unless logging is explicitly turned on again with the `ENABLE` procedure.

### Syntax

```
DBMS_CUBE_LOG.DISABLE (
    LOG_TYPE      IN  BINARY_INTEGER  DEFAULT,
    LOG_TARGET    IN  BINARY_INTEGER  DEFAULT);
```

### Parameters

**Table 43–3** *DISABLE Procedure Parameters*

Parameter	Description
log_type	<p>One of the following log types:</p> <ul style="list-style-type: none"> <li>■ 1: TYPE_OPERATIONS</li> <li>■ 2: TYPE_REJECTED_RECORDS</li> <li>■ 3: TYPE_DIMENSION_COMPILE</li> </ul> <p><b>Note:</b> You cannot disable the Cube Build log with this procedure.</p> <p>See "<a href="#">Logging Types</a>" on page 43-3.</p>
log_target	<p>One of the following destinations for the logging records. The logs are sent to a table unless you previously specified a different target using the <code>ENABLE</code> procedure.</p> <ul style="list-style-type: none"> <li>■ 1: TARGET_TABLE</li> <li>■ 2: TARGET_TRACE</li> <li>■ 3: TARGET_FILE</li> <li>■ 4: TARGET_LOB</li> </ul> <p>See "<a href="#">Logging Targets</a>" on page 43-4</p>

### Example

This command disables the dimension compilation error log table:

```
EXECUTE dbms_cube_log.disable(dbms_cube_log.type_dimension_compile);
```

## ENABLE Procedure

This procedure turns on logging for the duration of a session or until it is turned off using the `DISABLE` procedure.

The `ENABLE` procedure also allows you to direct logging to additional output types and to change the amount of detail in the logs. You can enable a log type to each of the log targets. For example, you can enable the Cube Operations log to the trace file, a table, and a file at different verbosity levels, but you cannot enable the Cube Operations log to two files at the same time.

This procedure also checks the format of the logs and updates them if necessary.

### Syntax

```
DBMS_CUBE_LOG.ENABLE (
    LOG_TYPE      IN      BINARY_INTEGER DEFAULT NULL,
    LOG_TARGET    IN      BINARY_INTEGER DEFAULT NULL,
    LOG_LEVEL     IN      BINARY_INTEGER DEFAULT NULL);
```

```
DBMS_CUBE_LOG.ENABLE (
    LOG_TYPE      IN      BINARY_INTEGER DEFAULT NULL,
    LOG_TARGET    IN      BINARY_INTEGER DEFAULT NULL,
    LOG_LEVEL     IN      BINARY_INTEGER DEFAULT NULL,
    LOG_LOCATION  IN      VARCHAR2      DEFAULT NULL);
```

```
DBMS_CUBE_LOG.ENABLE (
    LOG_TYPE      IN      BINARY_INTEGER DEFAULT NULL,
    LOG_TARGET    IN      BINARY_INTEGER DEFAULT NULL,
    LOG_LEVEL     IN      BINARY_INTEGER DEFAULT NULL,
    LOG_LOCATION  IN/OUT CLOB );
```

### Parameters

**Table 43–4** *ENABLE Procedure Parameters*

Parameter	Description
<code>log_type</code>	<p>One of the following log types:</p> <ul style="list-style-type: none"> <li>■ 1: <code>TYPE_OPERATIONS</code></li> <li>■ 2: <code>TYPE_REJECTED_RECORDS</code></li> <li>■ 3: <code>TYPE_DIMENSION_COMPILE</code></li> </ul> <p><b>Note:</b> You cannot disable the Cube Build log with this procedure.</p> <p>See "<a href="#">Logging Types</a>" on page 43-3.</p>
<code>log_target</code>	<p>One of the following destinations for the logging records. The logs are sent to a table unless you previously specified a different target using the <code>ENABLE</code> procedure.</p> <ul style="list-style-type: none"> <li>■ 1: <code>TARGET_TABLE</code></li> <li>■ 2: <code>TARGET_TRACE</code></li> <li>■ 3: <code>TARGET_FILE</code></li> <li>■ 4: <code>TARGET_LOB</code></li> </ul> <p>See "<a href="#">Logging Targets</a>" on page 43-4</p>

**Table 43–4 (Cont.) ENABLE Procedure Parameters**

Parameter	Description
log_level	<p>One of the following log verbosity levels. Each level adds new types of messages to the previous level.</p> <ul style="list-style-type: none"> <li>▪ 1: LEVEL_LOWEST</li> <li>▪ 2: LEVEL_LOW</li> <li>▪ 3: LEVEL_MEDIUM</li> <li>▪ 4: LEVEL_HIGH</li> <li>▪ 5: LEVEL_HIGHEST</li> </ul> <p>See "<a href="#">Verbosity Levels</a>" on page 43-5.</p>
log_location	The full identity of the log, such as <i>owner.table_name</i> when <i>log_target</i> is a table.

## Examples

The following command enables all cube logs:

```
EXECUTE dbms_cube_log.enable;
```

The following PL/SQL procedure sets the log level to LEVEL\_LOWEST:

```
BEGIN
  dbms_cube_log.disable(dbms_cube_log.type_rejected_records);
  dbms_cube_log.enable(dbms_cube_log.type_rejected_records,
    dbms_cube_log.target_table, dbms_cube_log.level_lowest);
END;
/
```

## FLUSH Procedure

This procedure forces all buffered messages to be written to the logs. The buffers are flushed automatically throughout a session, but manually flushing them before viewing the logs assures that you can view all of the messages.

### Syntax

```
DBMS_CUBE_LOG.FLUSH ( );
```

### Example

The following example flushes the buffers for all of the logs:

```
EXECUTE dbms_cube_log.flush;
```

## GET\_LOG Procedure

This procedure returns the current settings for the level and location of a particular log.

### Syntax

```
DBMS_CUBE_LOG.GET_LOG (
    LOG_TYPE      IN  BINARY_INTEGER  DEFAULT TYPE_OPERATIONS,
    LOG_TARGET    IN  BINARY_INTEGER  DEFAULT TARGET_TABLE,
    LOG_LEVEL     OUT BINARY_INTEGER,
    LOG_LOCATION  OUT  VARCHAR2 );
```

### Parameters

**Table 43–5** GET\_LOG Procedure Parameters

Parameter	Description
log_type	<p>One of the following log types:</p> <ul style="list-style-type: none"> <li>■ 1: TYPE_OPERATIONS</li> <li>■ 2: TYPE_REJECTED_RECORDS</li> <li>■ 3: TYPE_DIMENSION_COMPILE</li> </ul> <p>See <a href="#">"Logging Types"</a> on page 43-3.</p>
log_target	<p>One of the following destinations for the logging records. The logs are sent to a table unless you previously specified a different target using the ENABLE procedure.</p> <ul style="list-style-type: none"> <li>■ 1: TARGET_TABLE</li> <li>■ 2: TARGET_TRACE</li> <li>■ 3: TARGET_FILE</li> <li>■ 4: TARGET_LOB</li> </ul> <p>See <a href="#">"Logging Targets"</a> on page 43-4</p>
log_level	<p>One of the following log verbosity levels. Each level adds new types of messages to the previous level.</p> <ul style="list-style-type: none"> <li>■ 1: LEVEL_LOWEST</li> <li>■ 2: LEVEL_LOW</li> <li>■ 3: LEVEL_MEDIUM</li> <li>■ 4: LEVEL_HIGH</li> <li>■ 5: LEVEL_HIGHEST</li> </ul> <p>See <a href="#">"Verbosity Levels"</a> on page 43-5.</p>
log_location	<p>The full identity of the log, such as <i>owner.table_name</i> when log_target is a table.</p>

### Usage Notes

If log\_type is not active, then log\_level and log\_location are null. Use DBMS\_CUBE\_LOG.ENABLE to activate a log.

### Examples

This PL/SQL procedure provides information about the Cube Rejected Records log:

```
SET serverout ON format wrapped
```

```
DECLARE
    myloglevel  binary_integer;
    mylogtarget varchar2(128);

BEGIN
    dbms_cube_log.get_log(dbms_cube_log.type_rejected_records,
        dbms_cube_log.target_table, myloglevel, mylogtarget);

    dbms_output.put_line('Log Level: ' || myloglevel);
    dbms_output.put_line('Log Target: ' || mylogtarget);
END;
/
```

The procedure generates results like the following:

Log Level: 5

Log Target: GLOBAL.CUBE\_REJECTED\_RECORDS

## GET\_LOG\_SPEC Function

This function retrieves a description of all active Cube Operations logs, Cube Rejected Records logs, and Cube Dimension Compile logs.

### Syntax

```
DBMS_CUBE_LOG.GET_LOG_SPEC ( )  
    RETURN VARCHAR2;
```

### Returns

The type and target of all active logs.

### Usage Notes

You can use the output from this function as the input to SET\_LOG\_SPEC.

### Examples

The following example shows that the Cube Operations log, Cube Rejected Records log, and Cube Dimension Compile log are active. The Cube Operations log is stored in the session trace file and the other logs are stored in tables.

```
SELECT dbms_cube_log.get_log_spec FROM dual;
```

```
GET_LOG_SPEC
```

```
-----  
OPERATIONS(TABLE, TRACE) REJECTED_RECORDS(TABLE[DEBUG])
```



## GET\_PARAMETER Function

This function returns the current values of the options that control various aspects of logging. To set these options, use the `SET_PARAMETER` function.

### Syntax

```
DBMS_CUBE_LOG.GET_PARAMETER (
    LOG_TYPE      IN  BINARY_INTEGER,
    LOG_PARAMETER IN  BINARY_INTEGER )
RETURN BINARY_INTEGER;
```

### Parameters

**Table 43–6** *GET\_PARAMETER Function Parameters*

Parameter	Description
<code>log_type</code>	One of the following log types: <ul style="list-style-type: none"> <li>■ 1: TYPE_OPERATIONS</li> <li>■ 2: TYPE_REJECTED_RECORDS</li> <li>■ 3: TYPE_DIMENSION_COMPILE</li> </ul> See " <a href="#">Logging Types</a> " on page 43-3.
<code>log_parameter</code>	One of the following options: <ul style="list-style-type: none"> <li>■ 1: MAX_ERRORS</li> <li>■ 2: FLUSH_INTERVAL</li> <li>■ 3: LOG_FULL_RECORD</li> <li>■ 4: LOG_EVERY_N</li> <li>■ 5: ALLOW_ERRORS</li> </ul> See " <a href="#">SET_PARAMETER Procedure</a> " on page 43-30.

### Returns

The value of the specified `log_parameter`.

### Examples

This example shows the current maximum number of errors in the Cube Rejected Records log before logging stops. This parameter was previously set with the `SET_PARAMETER` procedure.

```
SELECT dbms_cube_log.get_parameter(dbms_cube_log.type_rejected_records, 1) -
       "Maximum Records" FROM dual;
```

```
Maximum Records
-----
                100
```

## LEVEL\_HIGH Function

This function returns the integer value of the high verbosity level.

### Syntax

```
DBMS_CUBE_LOG.LEVEL_HIGH ()  
    RETURN BINARY_INTEGER;
```

### Returns

4

### Usage Notes

Use this function instead of its binary integer equivalent for the *LOG\_LEVEL* parameter in DBMS\_CUBE\_LOG subprograms. See "[Verbosity Levels](#)" on page 43-5.

### Example

This command sets the verbosity level of the cube operations table to high:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -  
    dbms_cube_log.target_table, dbms_cube_log.level_high);
```

## LEVEL\_HIGHEST Function

This function returns the integer value of the highest verbosity level.

### Syntax

```
DBMS_CUBE_LOG.LEVEL_HIGHEST ()  
    RETURN BINARY_INTEGER;
```

### Returns

5

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_LEVEL parameter in DBMS\_CUBE\_LOG subprograms. See "[Verbosity Levels](#)" on page 43-5.

### Example

This command sets the verbosity level of the cube operations table to highest:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -  
    dbms_cube_log.target_table, dbms_cube_log.level_highest);
```

## LEVEL\_LOW Function

This function returns the integer value of the low verbosity level.

### Syntax

```
DBMS_CUBE_LOG.LEVEL_LOW ()  
    RETURN BINARY_INTEGER;
```

### Returns

2

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_LEVEL parameter in DBMS\_CUBE\_LOG subprograms. See "[Verbosity Levels](#)" on page 43-5.

### Example

This command sets the verbosity level of the cube operations table to low:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -  
    dbms_cube_log.target_table, dbms_cube_log.level_low);
```

## LEVEL\_LOWEST Function

This function returns the integer value of the lowest verbosity level. This level logs the status of each command used to build the cube dimensions and cubes, the use of slave processes, and summary records.

### Syntax

```
DBMS_CUBE_LOG.LEVEL_LOWEST ()  
    RETURN BINARY_INTEGER;
```

### Returns

1

### Usage Notes

Use this function instead of its binary integer equivalent for the `LOG_LEVEL` parameter in `DBMS_CUBE_LOG` subprograms. See "[Verbosity Levels](#)" on page 43-5.

### Example

This command sets the verbosity level of the cube operations table to lowest:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -  
    dbms_cube_log.target_table, dbms_cube_log.level_lowest);
```

## LEVEL\_MEDIUM Function

This function returns the integer value of the medium verbosity level.

### Syntax

```
DBMS_CUBE_LOG.LEVEL_MEDIUM (  
    RETURN BINARY_INTEGER;
```

### Returns

3

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_LEVEL parameter in DBMS\_CUBE\_LOG subprograms. See "[Verbosity Levels](#)" on page 43-5.

### Example

This command sets the verbosity level of the cube operations table to medium:

```
EXECUTE dbms_cube_log.enable(dbms_cube_log.type_operations, -  
    dbms_cube_log.target_table, dbms_cube_log.level_medium);
```

## SET\_LOG\_SPEC Procedure

This procedure sets all logging to the values specified in the input string.

### Syntax

```
DBMS_CUBE_LOG.SET_LOG_SPEC (
    LOG_SPEC      IN  VARCHAR2 );
```

### Parameters

**Table 43–7 SET\_LOG\_SPEC Procedure Parameters**

Parameter	Description
log_spec	<p>A string consisting of <i>type(target)</i> pairs.</p> <p><i>Type</i> can be:</p> <ul style="list-style-type: none"> <li>■ OPERATIONS</li> <li>■ REJECTED_RECORDS</li> <li>■ DIMENSION_COMPILE</li> </ul> <p><i>Target</i> can be:</p> <ul style="list-style-type: none"> <li>■ TABLE</li> <li>■ TRACE</li> <li>■ FILE</li> <li>■ LOB</li> </ul>

### Usage Notes

The GET\_LOG\_SPEC function returns a properly formatted string for SET\_LOG\_SPEC.

### Examples

This PL/SQL procedure disables all logs, verifies that they are disabled, then activates the Cube Operations log and the Cube Rejected Records log.

```
BEGIN
    dbms_cube_log.disable;
    dbms_output.put_line('Cube Logs: ' || dbms_cube_log.get_log_spec);

    dbms_cube_log.set_log_spec('OPERATIONS(TRACE) REJECTED_RECORDS(TABLE)');
    dbms_output.put_line('Cube Logs: ' || dbms_cube_log.get_log_spec);
END;
/
```

The output from the procedure verifies that the DISABLE function de-activated all logs, and the SET\_LOG\_SPEC function activated two logs:

Cube Logs:

```
Cube Logs: OPERATIONS(TRACE) REJECTED_RECORDS(TABLE)
```

## SET\_PARAMETER Procedure

This procedure sets options that control various aspects of logging. To obtain the current value of these options, use the GET\_PARAMETER function.

### Syntax

```
DBMS_CUBE_LOG.SET_PARAMETER (
    LOG_TYPE      IN  BINARY_INTEGER,
    LOG_PARAMETER IN  BINARY_INTEGER,
    VALUE         IN  BINARY_INTEGER );
```

### Parameters

**Table 43–8 SET\_PARAMETER Procedure Parameters**

Parameter	Description
log_type	<p>One of the following log types:</p> <ul style="list-style-type: none"> <li>■ 1: TYPE_OPERATIONS</li> <li>■ 2: TYPE_REJECTED_RECORDS</li> <li>■ 3: TYPE_DIMENSION_COMPILE</li> <li>■ 4: TYPE_BUILD</li> </ul> <p>See "<a href="#">Logging Types</a>" on page 43-3.</p>
log_parameter	<p>One of the following parameters:</p> <ul style="list-style-type: none"> <li>■ 1: MAX_ERRORS <ul style="list-style-type: none"> <li>Maximum number of records before signalling an end to logging, such as the number of rejected records in the Cube Rejected Records log or the number of compilation errors in the dimension compilation error log.</li> </ul> </li> <li>■ 2: FLUSH_INTERVAL <ul style="list-style-type: none"> <li>The number of seconds to buffer the records before writing them to a log. When this parameter is 0, the records are written directly to the logs without buffering.</li> </ul> </li> <li>■ 3: LOG_FULL_RECORD <ul style="list-style-type: none"> <li>Controls logging of rejected records. Set this parameter to one of the following constants: <ul style="list-style-type: none"> <li>0: FULL_RECORD_AUTO: Log the full record when no row ID is available.</li> <li>1: FULL_RECORD_ALWAYS: Always log the full record.</li> <li>2: FULL_RECORD_NEVER: Never log the full record.</li> </ul> </li> </ul> </li> <li>■ 4: LOG_EVERY_N <ul style="list-style-type: none"> <li>Enters a progress message every <i>n</i> rows during data maintenance.</li> </ul> </li> <li>■ 5: ALLOW_ERRORS: Displays logging errors, which are initially turned off to allow processing to proceed.</li> </ul>
value	The new value of <i>log_parameter</i> .

### Examples

This PL/SQL procedure sets the two parameters, then uses the GET\_PARAMETER function to show the settings:



```
BEGIN
  dbms_cube_log.set_parameter(dbms_cube_log.type_rejected_records, 1, 150);
  dbms_cube_log.set_parameter(dbms_cube_log.type_rejected_records, 2, 5);

  dbms_output.put_line('Max rejected records: ' ||
    dbms_cube_log.get_parameter(dbms_cube_log.type_rejected_records, 1));

  dbms_output.put_line('Buffer time: ' ||
    dbms_cube_log.get_parameter(dbms_cube_log.type_rejected_records, 2) ||
    ' seconds');
END;
/
```

The procedure displays this information:

Max rejected records: 150

Buffer time: 5 seconds

## TABLE\_CREATE Procedure

This procedure creates the table targets for the OLAP logs. You must have the CREATE TABLE privilege to use this procedure.

TABLE\_CREATE also upgrades existing log tables to the current version while preserving the data.

### Syntax

```
DBMS_CUBE_LOG.TABLE_CREATE (
    LOG_TYPE      IN   BINARY_INTEGER  DEFAULT,
    TBLNAME       IN   VARCHAR2        DEFAULT );
```

### Parameters

**Table 43–9** TABLE\_CREATE Procedure Parameters

Parameter	Description
log_type	One of the following log types: <ul style="list-style-type: none"> <li>▪ 1: TYPE_OPERATIONS</li> <li>▪ 2: TYPE_REJECTED_RECORDS</li> <li>▪ 3: TYPE_DIMENSION_COMPILE</li> <li>▪ 4: TYPE_BUILD</li> </ul> See " <a href="#">Logging Types</a> " on page 43-3.
tblname	A table name for the log. These are the default names: <ul style="list-style-type: none"> <li>▪ CUBE_OPERATIONS_LOG</li> <li>▪ CUBE_REJECTED_RECORDS</li> <li>▪ CUBE_DIMENSION_COMPILE</li> <li>▪ CUBE_BUILD_LOG</li> </ul>

### Examples

The

This example creates a Cube Operations log table named CUBE\_OPERATIONS\_LOG:

```
EXECUTE dbms_cube_log.table_create;
```

This example creates a Cube Rejected Records log table named CUBE\_REJECTED\_RECORDS:

```
EXECUTE dbms_cube_log.table_create(dbms_cube_log.type_rejected_records);
```

The next example creates a Cube Build log table named MY\_BUILD\_LOG:

```
EXECUTE dbms_cube_log.table_create -
    (dbms_cube_log.type_build, 'MY_BUILD_LOG');
```

## TARGET\_FILE Function

This function returns the integer value of a file target in DBMS\_CUBE\_LOG subprograms.

### Syntax

```
DBMS_CUBE_LOG.TARGET_FILE ()  
    RETURN BINARY_INTEGER;
```

### Returns

3

### Usage Notes

Use this function instead of its binary integer equivalent for the *LOG\_LEVEL* parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Targets](#)" on page 43-4.

### Example

This command disables the Cube Operations log file:

```
EXECUTE dbms_cube_log.disable -  
    (dbms_cube_log.type_operations, dbms_cube_log.target_file);
```

## TARGET\_LOB Function

This function returns the integer value of a LOB target.

### Syntax

```
DBMS_CUBE_LOG.TARGET_LOB ()  
    RETURN BINARY_INTEGER;
```

### Returns

4

### Usage Notes

Use this function instead of its binary integer equivalent for the *LOG\_LEVEL* parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Targets](#)" on page 43-4.

### Example

This command disables the Cube Operations log LOB:

```
EXECUTE dbms_cube_log.disable -  
    (dbms_cube_log.type_operations, dbms_cube_log.target_lob);
```

## TARGET\_TABLE Function

This function returns the integer value of a table target.

### Syntax

```
DBMS_CUBE_LOG.TARGET_TABLE (  
    RETURN BINARY_INTEGER;
```

### Returns

1

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_TARGET parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Targets](#)" on page 43-4.

### Example

This command disables the Cube Operations log table:

```
EXECUTE dbms_cube_log.disable -  
    (dbms_cube_log.type_operations, dbms_cube_log.target_table);
```

## TARGET\_TRACE Function

This function returns the integer value of a trace file target.

### Syntax

```
DBMS_CUBE_LOG.TARGET_TRACE ()  
    RETURN BINARY_INTEGER;
```

### Returns

2

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_TARGET parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Targets](#)" on page 43-4.

### Example

This command disables the Cube Operations log trace file:

```
EXECUTE dbms_cube_log.disable -  
    (dbms_cube_log.type_operations, dbms_cube_log.target_trace);
```

## TYPE\_BUILD Function

This function returns the integer value of the Cube Build log.

### Syntax

```
DBMS_CUBE_LOG.TYPE_BUILD ()  
    RETURN BINARY_INTEGER;
```

### Returns

4

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_TYPE parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Types](#)" on page 43-3.

### Example

This query returns the default name of a Cube Build log:

```
SELECT dbms_cube_log.default_name(dbms_cube_log.type_build) "Log Name" -  
       FROM dual;
```

```
Log Name
```

```
-----  
CUBE_BUILD_LOG
```

## TYPE\_DIMENSION\_COMPILE Function

This function returns the integer value of the Cube Dimension Compile log.

### Syntax

```
DBMS_CUBE_LOG.TYPE_DIMENSION_COMPILE ()  
    RETURN BINARY_INTEGER;
```

### Returns

3

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_TYPE parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Types](#)" on page 43-3.

### Example

This query returns the default name of a Cube Dimension Compile log:

```
SELECT dbms_cube_log.default_name(dbms_cube_log.type_dimension_compile) -  
       "Log Name" FROM dual;
```

```
Log Name  
-----  
CUBE_DIMENSION_COMPILE
```



## TYPE\_OPERATIONS Function

This function returns the integer value of the Cube Operations log.

### Syntax

```
DBMS_CUBE_LOG.TYPE_OPERATIONS (  
    RETURN BINARY_INTEGER;
```

### Returns

1

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_TYPE parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Types](#)" on page 43-3.

### Example

This query returns the default name of a Cube Dimension Compile log:

```
SELECT dbms_cube_log.default_name(dbms_cube_log.type_operations) "Log Name" -  
       FROM dual;
```

```
Log Name  
-----  
CUBE_OPERATIONS_LOG
```

## TYPE\_REJECTED\_RECORDS Function

This function returns the integer value of the cube Cube Rejected Records log.

### Syntax

```
DBMS_CUBE_LOG.TYPE_REJECTED_RECORDS (  
    RETURN BINARY_INTEGER;
```

### Returns

2

### Usage Notes

Use this function instead of its binary integer equivalent for the LOG\_TYPE parameter in DBMS\_CUBE\_LOG subprograms. See "[Logging Types](#)" on page 43-3.

### Example

This query returns the default name of a Cube Rejected Records log:

```
SELECT dbms_cube_log.default_name(dbms_cube_log.type_rejected_records) -  
       "Log Name" FROM dual;
```

```
Log Name  
-----  
CUBE_REJECTED_RECORDS
```

## VERSION Function

This function returns the version number of a specific log table or the current version number of a specific log type.

### Syntax

```
DBMS_CUBE_LOG.VERSION (
    LOG_TYPE      IN   BINARY_INTEGER  DEFAULT 1,
    TBLNAME       IN   VARCHAR2        DEFAULT NULL)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 43–10** *VERSION Function Parameters*

Parameter	Description
log_type	One of the following log types: <ul style="list-style-type: none"> <li>■ 1: TYPE_OPERATIONS</li> <li>■ 2: TYPE_REJECTED_RECORDS</li> <li>■ 3: TYPE_DIMENSION_COMPILE</li> <li>■ 4: TYPE_BUILD</li> </ul> See " <a href="#">Logging Types</a> " on page 43-3.
tblname	The name of the log table whose version is returned.

### Returns

A version number

### Examples

This example returns the current version of the Cube Operations log:

```
SELECT dbms_cube_log.version FROM dual;
```

```
VERSION
-----
      2
```

This example returns the version number of an existing Cube Rejected Records log named CUBE\_REJECTED\_RECORDS.

```
SELECT dbms_cube_log.version(dbms_cube_log.type_rejected_records, -
    'CUBE_REJECTED_RECORDS') version FROM dual;
```

```
VERSION
-----
      2
```



---

---

## DBMS\_DATA\_MINING

Oracle Data Mining is an analytical technology that derives actionable information from data in an Oracle Database. You can use Oracle Data Mining to evaluate the probability of future events and discover unsuspected associations and groupings within your data.

The `DBMS_DATA_MINING` package is the programmatic interface for creating and managing data mining models (mining model schema objects). Oracle Data Mining also supports a family of SQL functions for deploying data mining models.

Oracle Data Miner, a graphical interface to Oracle Data Mining, is available for download from the Oracle Technology Network at:

<http://www.oracle.com/technetwork/database/options/odm/>

### See Also:

- [Chapter 45, "DBMS\\_DATA\\_MINING\\_TRANSFORM"](#). This package supports data pre-processing for data mining models.
- [Chapter 104, "DBMS\\_PREDICTIVE\\_ANALYTICS"](#). This package supports several routines that perform automated data mining.
- *Oracle Database SQL Language Reference* for information about the SQL Data Mining scoring functions.
- *Oracle Data Mining Concepts* for an introduction to Oracle Data Mining.
- *Oracle Data Mining Concepts* for new features in Oracle Data Mining.

This chapter contains the following topics:

- [Using DBMS\\_DATA\\_MINING](#)
  - Overview
  - Mining Model Objects
  - Security Model
  - Deprecated Subprograms
  - Mining Functions
  - Model Settings
  - Data Types
- [Summary of DBMS\\_DATA\\_MINING Subprograms](#)

---

## Using DBMS\_DATA\_MINING

This section contains topics that relate to using the DBMS\_DATA\_MINING package.

- [Overview](#)
- [Mining Model Objects](#)
- [Security Model](#)
- [Deprecated Subprograms](#)
- [Mining Functions](#)
- [Model Settings](#)
- [Data Types](#)

## Overview

Oracle Data Mining supports both **supervised** and **unsupervised** data mining. Supervised data mining predicts a target value based on historical data. Unsupervised data mining discovers natural groupings and does not use a target.

**See Also:** *Oracle Data Mining Concepts* for more information

A data mining **function** refers to the methods for solving a given class of data mining problems. The mining function must be specified when a model is created. See "[Mining Functions](#)" on page 44-9.

---

**Note on Terminology:** In data mining terminology, a *function* is a general type of problem to be solved by a given approach to data mining. In SQL language terminology, a *function* is an operator that returns a value.

In Oracle Data Mining documentation, the term *function*, or *mining function* refers to a data mining function; the term *SQL function* or *SQL Data Mining function* refers to a SQL function for scoring (deploying) data mining models. The SQL Data Mining functions are documented in *Oracle Database SQL Language Reference*.

---

Supervised data mining functions include:

- Classification
- Regression
- Attribute Importance

Unsupervised data mining functions include:

- Clustering
- Association
- Feature Extraction
- Anomaly Detection (one-class classification)

The steps you use to build and apply a mining model depend on the data mining function and the algorithm being used. The algorithms supported by Oracle Data Mining are listed in [Table 44-1](#).

**Table 44-1 Oracle Data Mining Algorithms**

Algorithm	Abbreviation	Function
Adaptive Bayes Network (deprecated)	ABN	Classification
Apriori	AP	Association
Decision Tree	DT	Classification
Generalized Linear Model	GLM	Classification and Regression
<i>k</i> -Means (default clustering algorithm)	KM	Clustering
Minimal Descriptor Length	MDL	Attribute Importance

**Table 44–1 (Cont.) Oracle Data Mining Algorithms**

<b>Algorithm</b>	<b>Abbreviation</b>	<b>Function</b>
Naive Bayes (default classification algorithm)	NB	Classification
Non-Negative Matrix Factorization	NMF	Feature Extraction
Orthogonal Partitioning Clustering	O-Cluster	Clustering
Support Vector Machine (default regression algorithm)	SVM	Classification and regression (and anomaly detection through one-class classification)



## Mining Model Objects

Mining models are Oracle Database schema objects. They support the standard security features of Oracle Database. Mining models are also supported by SQL COMMENT and SQL AUDIT.

### See Also:

- *Oracle Data Mining Administrator's Guide* for information about mining model objects, SQL COMMENT, and SQL AUDIT
- *Oracle Data Mining Administrator's Guide* for information about mining model security
- *Oracle Data Mining Administrator's Guide* for information about the sample Data Mining programs

## ALL\_MINING\_MODELS

You can query the data dictionary view ALL\_MINING\_MODELS to obtain a list of accessible mining models.

### Example 44-1 ALL\_MINING\_MODELS

```
SQL> describe all_mining_models
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2 (30)
MODEL_NAME	NOT NULL	VARCHAR2 (30)
MINING_FUNCTION		VARCHAR2 (30)
ALGORITHM		VARCHAR2 (30)
CREATION_DATE	NOT NULL	DATE
BUILD_DURATION		NUMBER
MODEL_SIZE		NUMBER
COMMENTS		VARCHAR2 (4000)

**See Also:** *Oracle Data Mining Application Developer's Guide* for more information about ALL\_MINING\_MODELS and related views

## Mining Model Naming Restrictions

The naming rules for models are more restrictive than the naming rules for most database schema objects. A model name must satisfy the following additional requirements:

- It must be 25 or fewer characters long.
- It must be a nonquoted identifier. Oracle requires that nonquoted identifiers contain only alphanumeric characters, the underscore (\_), dollar sign (\$), and pound sign (#); the initial character must be alphabetic. Oracle strongly discourages the use of the dollar sign and pound sign in nonquoted literals.

Naming requirements for schema objects are fully documented in *Oracle Database SQL Language Reference*.

## ALL\_MINING\_MODEL\_ATTRIBUTES

You can query the data dictionary view ALL\_MINING\_MODEL\_ATTRIBUTES to obtain a list of the **data attributes** for each accessible mining model. Data attributes are the

columns of data used by an algorithm to build a model. Some or all of these columns must be present in the data to which the model is applied.

Data attributes are referred to as the **model signature**. The `ALL_MINING_MODEL_ATTRIBUTES` view lists the data attributes in the model signature, including the target if the model is supervised.

An algorithm builds an internal representation of the data attributes and uses them as either **categoricals** (data that classifies or categorizes) or as **numericals** (continuous data). These internal **model attributes** can be viewed using the `GET_MODEL_DETAILS` functions.

#### **Example 44–2 ALL\_MINING\_MODEL\_ATTRIBUTES**

```
SQL> describe all_mining_model_attributes
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2 (30)
MODEL_NAME	NOT NULL	VARCHAR2 (30)
ATTRIBUTE_NAME	NOT NULL	VARCHAR2 (30)
ATTRIBUTE_TYPE		VARCHAR2 (11)
DATA_TYPE		VARCHAR2 (12)
DATA_LENGTH		NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
USAGE_TYPE		VARCHAR2 (8)
TARGET		VARCHAR2 (3)

**See Also:** *Oracle Data Mining Application Developer's Guide* for more information about attributes and `ALL_MINING_MODEL_ATTRIBUTES`

## **ALL\_MINING\_MODEL\_SETTINGS**

The view `ALL_MINING_MODEL_SETTINGS` returns the **settings** for each accessible mining model. Settings control various characteristics of mining models.

All settings have default values. The values of some settings are generated by the algorithm by default. You can override the default value of a setting by specifying its value in a settings table for the model. All settings, both default and user-specified, are listed in `ALL_MINING_MODEL_SETTINGS`.

#### **Example 44–3 ALL\_MINING\_MODEL\_SETTINGS**

```
SQL> describe all_mining_model_settings
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2 (30)
MODEL_NAME	NOT NULL	VARCHAR2 (30)
SETTING_NAME	NOT NULL	VARCHAR2 (30)
SETTING_VALUE		VARCHAR2 (4000)
SETTING_TYPE		VARCHAR2 (7)

**See Also:**

- [Model Settings](#) on page 44-10
- *Oracle Data Mining Application Developer's Guide* for more information about `ALL_MINING_MODEL_SETTINGS`

## Security Model

The DBMS\_DATA\_MINING package is owned by user SYS and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The DBMS\_DATA\_MINING package exposes APIs that are leveraged by the Oracle Data Mining option. Users who wish to create mining models in their own schema require the CREATE MINING MODEL system privilege (as well as the CREATE TABLE and CREATE VIEW system privilege). Users who wish to create mining models in other schemas require the CREATE ANY MINING MODEL system privilege (as well as the corresponding table and view creation privileges).

Users have full control over managing models that exist within their own schema. Additional system privileges necessary for managing data mining models in other schemas include ALTER ANY MINING MODEL, DROP ANY MINING MODEL, SELECT ANY MINING MODEL, COMMENT ANY MINING MODEL, and AUDIT ANY.

Individual object privileges on mining models, ALTER MINING MODEL and SELECT MINING MODEL, can be used to selectively grant privileges on a model to a different user.

**See Also:** *Oracle Data Mining Administrator's Guide* for more information about the security features of Oracle Data Mining

## Deprecated Subprograms

The following subprograms were deprecated in Oracle Data Mining 11g Release 1 (11.1).

- GET\_DEFAULT\_SETTINGS

Replaced with data dictionary views:  
USER/ALL/DBA\_MINING\_MODEL\_SETTINGS

- GET\_MODEL\_SETTINGS

Replaced with data dictionary views:  
USER/ALL/DBA\_MINING\_MODEL\_SETTINGS

- GET\_MODEL\_SIGNATURE

Replaced with data dictionary views:  
USER/ALL/DBA\_MINING\_MODEL\_ATTRIBUTES

The following view was deprecated in Oracle Data Mining 11g Release 1 (11.1).

- DM\_USER\_MODELS

Replaced with data dictionary views: USER/ALL/DBA\_MINING\_MODELS

The Adaptive Bayes Network algorithm was deprecated in Oracle Data Mining 11g Release 1 (11.1).

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

---

Since 11g Release 1 (11.1), the DMSYS schema is no longer present in the database. Oracle Data Mining metadata now resides in SYS.

## Mining Functions

The constants that specify the mining function of a model are listed in [Table 44-2](#). The concept of a "mining function" is introduced in ["Overview"](#) on page 44-3.

All models are created with a mining function. The mining function is a required argument to the [CREATE\\_MODEL Procedure](#).

**Table 44-2 Mining Functions**

Value	Description
ASSOCIATION	<p>Association is a descriptive mining function. An association model identifies relationships and the probability of their occurrence within a data set.</p> <p>Association models use the Apriori algorithm.</p>
ATTRIBUTE_IMPORTANCE	<p>Attribute Importance is a predictive mining function. An attribute importance model identifies the relative importance of an attribute in predicting a given outcome.</p> <p>Attribute Importance models use the Minimal Descriptor Length algorithm.</p>
CLASSIFICATION	<p>Classification is a predictive mining function. A classification model uses historical data to predict a categorical target.</p> <p>Classification models can use: Naive Bayes, Adaptive Bayes Network (deprecated), Decision Tree, Logistic Regression, or Support Vector Machine algorithms. The default is Naive Bayes.</p> <p>The classification function can also be used for <b>anomaly detection</b>. In this case, the SVM algorithm with a null target is used (One-Class SVM).</p>
CLUSTERING	<p>Clustering is a descriptive mining function. A clustering model identifies natural groupings within a data set.</p> <p>Clustering models can use: <i>k</i>-Means or O-Cluster algorithms. The default is <i>k</i>-Means.</p>
FEATURE_EXTRACTION	<p>Feature Extraction is a descriptive mining function. A feature extraction model creates an optimized data set on which to base a model.</p> <p>Feature extraction models use the Non-Negative Matrix Factorization algorithm.</p>
REGRESSION	<p>Regression is a predictive mining function. A regression model uses historical data to predict a numerical target.</p> <p>Regression models can use Support Vector Machine or Linear Regression. The default is Support Vector Machine.</p>

## Model Settings

Oracle Data Mining uses settings to specify the algorithm and other characteristics of a model. Some settings are general, some are specific to a mining function, and some are specific to an algorithm.

All settings have default values. If you want to override one or more of the settings for a model, you must create a settings table. The settings table must have the column names and data types shown in [Table 44-3](#).

**Table 44-3 Required Columns in the Model Settings Table**

Column Name	Data Type
SETTING_NAME	VARCHAR2 (30)
SETTING_VALUE	VARCHAR2 (4000)

The information you provide in the settings table is used by the model at build time. The name of the settings table is an optional argument to the [CREATE\\_MODEL Procedure](#).

You can find the settings used by a model by querying the data dictionary view `ALL_MINING_MODEL_SETTINGS`. This view lists the model settings used by the mining models to which you have access. All the setting values are included in the view, whether default or user-specified. See "[ALL\\_MINING\\_MODEL\\_SETTINGS](#)" on page 44-6.

## Algorithm Names

The `ALGO_NAME` setting specifies the model algorithm. The values for the `ALGO_NAME` setting are listed in [Table 44-4](#).

**Table 44-4 Algorithm Names**

ALGO_NAME Value	Description	Mining Function
ALGO_ADAPTIVE_BAYES_NETWORK	Adaptive Bayes Network (deprecated)	Classification
ALGO_DECISION_TREE	Decision Tree	Classification
ALGO_NAIVE_BAYES	Naive Bayes	Classification
ALGO_GENERALIZED_LINEAR_MODEL	Generalized Linear Model	Classification and Regression
ALGO_SUPPORT_VECTOR_MACHINES	Support Vector Machine	Classification and Regression
ALGO_KMEANS	Enhanced <i>k</i> -Means	Clustering
ALGO_O_CLUSTER	O-Cluster	Clustering
ALGO_AI_MDL	Minimum Description Length	Attribute Importance
ALGO_APRIORI_ASSOCIATION_RULES	Apriori	Association Rules
ALGO_NONNEGATIVE_MATRIX_FACTOR	Non-Negative Matrix Factorization	Feature Extraction

Oracle Data Mining supports more than one algorithm for the classification, regression, and clustering mining functions. Each of these mining functions has a default algorithm, as shown in [Table 44-5](#).

**Table 44–5 Default Algorithms**

Mining Function	Default Algorithm
Classification	Naive Bayes
Regression	Support Vector Machine
Clustering	<i>k</i> -Means

## Automatic Data Preparation

The **PREP\_AUTO** setting indicates whether or not the model will use Automatic Data Preparation (ADP). By default ADP is disabled.

When you enable ADP, the model uses heuristics to transform the build data according to the requirements of the algorithm. The transformation instructions are stored with the model and reused whenever the model is applied. You can view the transformation instructions in the model details.

You can choose to supplement automatic data preparations by specifying additional transformations in the `xform_list` parameter when you build the model. (See "[CREATE\\_MODEL Procedure](#)" on page 44-50.)

If you do not use ADP (default) *and* do not specify transformations in the `xform_list` parameter to `CREATE_MODEL` (also the default), you will continue to operate in 10.2 mode. This means that you must implement your own transformations separately in the build, test, and scoring data; you must take special care to implement the exact same transformations in each data set.

If you do not use ADP, but you *do* specify transformations in the `xform_list` parameter to `CREATE_MODEL`, Oracle Data Mining embeds the transformation definitions in the model and prepares the test and scoring data to match the build data. Because of automatic and embedded data preparation, mining models are known as **supermodels**.

The values for the `PREP_AUTO` setting are described in [Table 44–6](#).

**Table 44–6 PREP\_AUTO Setting**

PREP_AUTO Value	Description
PREP_AUTO_OFF	Disable Automatic Data Preparation (default).
PREP_AUTO_ON	Enable Automatic Data Preparation.

**See Also:** *Oracle Data Mining Concepts* for information about data preparation

## Mining Function Settings

The settings described in [Table 44–7](#) apply to a mining function.

**Table 44–7 Mining Function Settings**

Mining Function	Setting Name	Setting Value	Description
Association	ASSO_MAX_RULE_LENGTH	TO_CHAR (2<= numeric_exp r <=20)	Maximum rule length for association rules. Default is 4.
Association	ASSO_MIN_CONFIDENCE	TO_CHAR (0<= numeric_exp r <=1)	Minimum confidence for association rules. Default is 0.1.
Association	ASSO_MIN_SUPPORT	TO_CHAR (0<= numeric_exp r <=1)	Minimum support for association rules. Default is 0.1.
Classification	CLAS_COST_TABLE_NAME	table_name	(Decision Tree only) Name of a table that stores a cost matrix to be used by the algorithm in building the model. The cost matrix specifies the costs associated with misclassifications.  Only Decision Tree models can use a cost matrix at build time. All classification algorithms can use a cost matrix at apply time.  The cost matrix table is user-created. See <a href="#">"ADD_COST_MATRIX Procedure"</a> on page 44-25 for the column requirements.  See <i>Oracle Data Mining Concepts</i> for information about costs.
Classification	CLAS_PRIORS_TABLE_NAME	table_name	(Naive Bayes) Name of a table that stores prior probabilities to offset differences in distribution between the build data and the scoring data.  The priors table is user-created. See <i>Oracle Data Mining Application Developer's Guide</i> for the column requirements. See <i>Oracle Data Mining Concepts</i> for additional information about priors.
Classification	CLAS_WEIGHTS_TABLE_NAME	table_name	(GLM and SVM only) Name of a table that stores weighting information for individual target values in SVM classification and GLM logistic regression models. The weights are used by the algorithm to bias the model in favor of higher weighted classes.  The class weights table is user-created. See <i>Oracle Data Mining Application Developer's Guide</i> for the column requirements. See <i>Oracle Data Mining Concepts</i> for additional information about class weights.
Clustering	CLUS_NUM_CLUSTERS	TO_CHAR ( numeric_exp r >=1)	Maximum number of leaf clusters generated by a clustering algorithm. (Oracle Data Mining clustering algorithms are hierarchical, as described in <i>Oracle Data Mining Concepts</i> .)  Enhanced <i>k</i> -Means usually produces the exact number of clusters specified by CLUS_NUM_CLUSTERS, unless there are fewer distinct data points.  O-Cluster may produce fewer clusters than the number specified by CLUS_NUM_CLUSTERS, depending on the data.  Default is 10.
Feature Extraction	FEAT_NUM_FEATURES	TO_CHAR ( numeric_exp r >=1)	Number of features to be extracted by a feature extraction model.  The default is estimated from the data by the algorithm.

**See Also:** *Oracle Data Mining Concepts* for information about mining functions



## Global Settings

The settings in [Table 44–8](#) are applicable to any type of model, but are currently only implemented for specific algorithms.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–8 Global Settings**

Setting Name	Setting Value	Description
ODMS_ITEM_ID_COLUMN_NAME	<i>column_name</i>	<p>(Association Rules only) Name of a column that contains the items in a transaction. When this setting is specified, the algorithm expects the data to be presented in native transactional format, consisting of two columns:</p> <ul style="list-style-type: none"> <li>▪ Case ID, either categorical or numerical</li> <li>▪ Item ID, either categorical or numerical, specified by <code>ODMS_ITEM_ID_COLUMN_NAME</code></li> </ul> <p>A typical example of transactional data is market basket data, wherein a case represents a basket that may contain many items. Each item is stored in a separate row, and many rows may be needed to represent a case. The case ID values do not uniquely identify each row. Transactional data is also called multi-record case data.</p> <p>Association Rules is normally used with transactional data, but it can also be applied to single-record case data (similar to other algorithms).</p> <p>For more information about single-record and multi-record case data, see <i>Oracle Data Mining Application Developer's Guide</i>.</p>

**Table 44–8 (Cont.) Global Settings**

Setting Name	Setting Value	Description
ODMS_ITEM_VALUE_COLUMN_NAME	<i>column_name</i>	<p>(Association Rules only) Name of a column that contains a value associated with each item in a transaction. This setting is only used when a value has been specified for ODMS_ITEM_ID_COLUMN_NAME indicating that the data is presented in native transactional format.</p> <p>When ODMS_ITEM_VALUE_COLUMN_NAME is specified, the algorithm expects the build data to consist of three columns:</p> <ul style="list-style-type: none"> <li>▪ Case ID, either categorical or numerical</li> <li>▪ Item ID, either categorical or numerical, specified by ODMS_ITEM_ID_COLUMN_NAME</li> <li>▪ Item value, either categorical or numerical, specified by ODMS_ITEM_VALUE_COLUMN_NAME</li> </ul> <p>The item value column may specify information such as the number of items (for example, three apples) or the type of the item (for example, macintosh apples).</p>
ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_MEAN_MODE ODMS_MISSING_VALUE_DELETE_ROW	<p>(GLM only) How to treat missing values in the training data. This setting does not affect the scoring data.</p> <p>Oracle Data Mining replaces missing values with the mean (numeric attributes) or the mode (categorical attributes) both at build time and apply time. You can set ODMS_MISSING_VALUE_TREATMENT to ODMS_MISSING_VALUE_DELETE_ROW to override this behavior in the training data. When ODMS_MISSING_VALUE_TREATMENT is set to ODMS_MISSING_VALUE_DELETE_ROW, the rows in the training data that contain missing values are deleted. However, if you want to replicate this missing value treatment in the scoring data, you must perform the transformation explicitly. For instructions, see <i>Oracle Data Mining Concepts</i>.</p> <p>The value ODMS_MISSING_VALUE_DELETE_ROW is only valid for tables without nested columns. If this value is used with nested data, an exception is raised.</p>
ODMS_ROW_WEIGHT_COLUMN_NAME	<i>column_name</i>	<p>(GLM only) Name of a column in the training data that contains a weighting factor for the rows.</p> <p>Row weights can be used as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. Row weights can also be used to emphasize certain rows during model construction. For example, to bias the model towards rows that are more recent and away from potentially obsolete data.</p>

**See Also:**

*Oracle Data Mining Concepts* for information about GLM

*Oracle Data Mining Concepts* for information about Association Rules

**Algorithm Settings: Adaptive Bayes Network (deprecated)**

These settings affect the behavior of the Adaptive Bayes Network algorithm.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–9 ABN Settings**

Setting	Value	Description
ABNS_MAX_BUILD_MINUTES	TO_CHAR( numeric_expr >=0)	Maximum time to complete an ABN model build. Default is 0, which implies no time limit.
ABNS_MAX_NB_PREDICTORS	TO_CHAR( numeric_expr >0)	Maximum number of predictors, measured by their MDL ranking, to be considered for building an ABN model of type <code>abns_naive_bayes</code> . Default is 10.
ABNS_MAX_PREDICTORS	TO_CHAR( numeric_expr >0)	Maximum number of predictors, measured by their MDL ranking, to be considered for building an ABN model of type <code>abns_single_feature</code> or <code>abns_multi_feature</code> . Default is 25.
ABNS_MODEL_TYPE	ABNS_MULTI_FEATURE ABNS_NAIVE_BAYES ABNS_SINGLE_FEATURE	Type of ABN model. The default is <code>multi_feature</code> .

### Algorithm Settings: Decision Tree

These settings affect the behavior of the Decision Tree algorithm.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the `ALL`, `USER`, or `DBA` prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–10 Decision Tree Settings**

Setting	Value	Description
TREE_IMPURITY_METRIC	TREE_IMPURITY_ENTROPY TREE_IMPURITY_GINI	Tree impurity metric for Decision Tree. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with a metric. Decision trees can use either gini ( <code>TREE_IMPURITY_GINI</code> ) or entropy ( <code>TREE_IMPURITY_ENTROPY</code> ) as the purity metric. By default, the algorithm uses gini.
TREE_TERM_MAX_DEPTH	TO_CHAR( 2<= numeric_expr <=20)	Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). Default is 7.
TREE_TERM_MINPCT_MODE	TO_CHAR( 0<= numeric_expr <=10)	No child shall have fewer records than this number, which is expressed as a percentage of the training rows. Default is 0.05, indicating 0.05%.
TREE_TERM_MINPCT_SPLIT	TO_CHAR( 0 <= numeric_expr <=20)	Criteria for splits: minimum number of records in a parent node expressed as a percent of the total number of records used to train the model. No split is attempted if number of records is below this value. Default is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	TO_CHAR( numeric_expr >=0)	No child shall have fewer records than this number. Default is 10.
TREE_TERM_MINREC_SPLIT	TO_CHAR( numeric_expr >=0)	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if number of records is below this value. Default is 20.

## Algorithm Settings: Generalized Linear Models

These settings affect the behavior of GLM models. GLM can be used for classification (logistic regression) or regression (linear regression).

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the `ALL`, `USER`, or `DBA` prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–11 GLM Settings**

Setting Name	Setting Value	Description
GLMS_CONF_LEVEL	TO_CHAR (0 < numeric_expr < 1)	The confidence level for coefficient confidence intervals. The default confidence level is 0.95.
GLMS_DIAGNOSTICS_TABLE_NAME	table_name	The name of a table to contain row-level diagnostic information for a GLM model. The table is created during model build.  If you want to create a diagnostics table, you must specify a case ID when you build the model. (See the <a href="#">CREATE_MODEL Procedure</a> .) If you specify a diagnostics table but do not provide a case ID, an exception is raised.  For information on GLM diagnostics, see <i>Oracle Data Mining Concepts</i> .
GLMS_REFERENCE_CLASS_NAME	target_value	The target value to be used as the reference value in a logistic regression model. Probabilities will be produced for the other (non-reference) class.  By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.
GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE GLMS_RIDGE_REG_DISABLE	Whether or not ridge regression will be enabled.  By default, the algorithm determines whether or not to use ridge. You can explicitly enable ridge by setting <code>GLMS_RIDGE_REGRESSION</code> to <code>GLMS_RIDGE_REG_ENABLE</code> .  Ridge applies to both regression and classification mining functions.  When ridge is enabled, no prediction bounds are produced by the <code>PREDICTION_BOUNDS</code> SQL operator.
GLMS_RIDGE_VALUE	TO_CHAR (0 < numeric_expr)	The value for the ridge parameter used by the algorithm. This setting is only used when you explicitly enable ridge regression by setting <code>GLMS_RIDGE_REGRESSION</code> to <code>GLMS_RIDGE_REG_ENABLE</code> .  If ridge regression is enabled internally by the algorithm, the ridge parameter is determined by the algorithm.
GLMS_VIF_FOR_RIDGE	GLMS_VIF_RIDGE_ENABLE GLMS_VIF_RIDGE_DISABLE	(Linear regression only) Whether or not to produce Variance Inflation Factor (VIF) statistics when ridge is being used.  By default, VIF is not produced when ridge is enabled.  When you explicitly enable ridge regression by setting <code>GLMS_RIDGE_REGRESSION</code> to <code>GLMS_RIDGE_REG_ENABLE</code> , you can request VIF statistics by setting <code>GLMS_VIF_FOR_RIDGE</code> to <code>GLMS_VIF_RIDGE_ENABLE</code> ; the algorithm will produce VIF if enough system resources are available.

**See Also:** *Oracle Data Mining Concepts* for information about GLM

## Algorithm Settings: k-Means

These settings affect the behavior of the *k*-Means algorithm.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–12 k-Means Settings**

Setting Name	Setting Value	Description
KMNS_BLOCK_GROWTH	TO_CHAR (1< numeric_expr <=5)	Growth factor for memory allocated to hold cluster data Default value is 2
KMNS_CONV_TOLERANCE	TO_CHAR (0< numeric_expr <=0.5)	Convergence tolerance for <i>k</i> -Means algorithm Default is 0.01
KMNS_DISTANCE	KMNS_COSINE KMNS_EUCLIDEAN KMNS_FAST_COSINE	Distance Function for <i>k</i> -Means Clustering. The default is euclidean.
KMNS_ITERATIONS	TO_CHAR (0< numeric_expr <=20)	Number of iterations for <i>k</i> -Means algorithm Default is 3
KMNS_MIN_PCT_ATTR_SUPPORT	TO_CHAR (0<= numeric_expr <=1)	The fraction of attribute values that must be non-null in order for the attribute to be included in the rule description for the cluster.  Setting the parameter value too high in data with missing values can result in very short or even empty rules. Default is 0.1.
KMNS_NUM_BINS	TO_CHAR (numeric_expr >0)	Number of histogram bins. Specifies the number of bins in the attribute histogram produced by <i>k</i> -Means. The bin boundaries for each attribute are computed globally on the entire training data set. The binning method is equi-width. All attributes have the same number of bins with the exception of attributes with a single value that have only one bin.  Default is 10.
KMNS_SPLIT_CRITERION	KMNS_SIZE KMNS_VARIANCE	Split criterion for <i>k</i> -Means Clustering. The default criterion is the variance.

**See Also:** *Oracle Data Mining Concepts* for information about *k*-Means.

### Algorithm Settings: Naive Bayes

These settings affect the behavior of the Naive Bayes Algorithm.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–13 Naive Bayes Settings**

Setting Name	Setting Value	Description
NABS_PAIRWISE_THRESHOLD	TO_CHAR (0<= numeric_expr <=1)	Value of pairwise threshold for NB algorithm Default is 0.01.
NABS_SINGLETON_THRESHOLD	TO_CHAR (0<= numeric_expr <=1)	Value of singleton threshold for NB algorithm Default value is 0.01

**See Also:** *Oracle Data Mining Concepts* for information about Naive Bayes

## Algorithm Settings: Non-Negative Matrix Factorization

These settings affect the behavior of the Non-Negative Matrix Factorization algorithm.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–14 NMF Settings**

Setting Name	Setting Value	Description
NMFS_CONV_TOLERANCE	TO_CHAR(0 < numeric_expr <= 0.5)	Convergence tolerance for NMF algorithm Default is 0.05
NMFS_NONNEGATIVE_SCORING	NMFS_NONNEG_SCORING_ENABLE NMFS_NONNEG_SCORING_DISABLE	Whether negative numbers should be allowed in scoring results. When set to NMFS_NONNEG_SCORING_ENABLE, negative feature values will be replaced with zeros. When set to NMFS_NONNEG_SCORING_DISABLE, negative feature values will be allowed. Default is NMFS_NONNEG_SCORING_ENABLE
NMFS_NUM_ITERATIONS	TO_CHAR(1 <= numeric_expr <= 500)	Number of iterations for NMF algorithm Default is 50
NMFS_RANDOM_SEED	TO_CHAR(numeric_expr)	Random seed for NMF algorithm. Default is -1.

**See Also:** *Oracle Data Mining Concepts* for information about NMF

## Algorithm Settings: O-Cluster

These settings affect the behavior of the O-Cluster algorithm.

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_ALL_MINING_MODEL_SETTINGS`.

**Table 44–15 O-Cluster Settings**

Setting Name	Setting Value	Description
OCLT_MAX_BUFFER	TO_CHAR(numeric_expr > 0)	Buffer size for O-Cluster. Default is 50,000.
OCLT_SENSITIVITY	TO_CHAR(0 <= numeric_expr <= 1)	A fraction that specifies the peak density required for separating a new cluster. The fraction is related to the global uniform density. Default is 0.5.

**See Also:** *Oracle Data Mining Concepts* for information about O-Cluster

## Algorithm Settings: Support Vector Machine

These settings affect the behavior of the Support Vector Machine algorithm. SVM can be used for classification or regression, or for anomaly detection (classification with a null target).

You can query the data dictionary view `*_MINING_MODEL_SETTINGS` (using the ALL, USER, or DBA prefix) to find the setting values for a model. See *Oracle Data Mining Application Developer's Guide* for information about `*_MINING_MODEL_SETTINGS`.

**Table 44–16 SVM Settings**

Setting Name	Setting Value	Description
SVMS_ACTIVE_LEARNING	SVMS_AL_DISABLE SVMS_AL_ENABLE	Whether active learning is enabled or disabled. By default, active learning is enabled.  When active learning is enabled, the SVM algorithm uses active learning to build a reduced size model. When active learning is disabled, the SVM algorithm builds a standard model.
SVMS_COMPLEXITY_FACTOR	TO_CHAR( <i>numeric_exp</i> <i>r</i> >0)	Value of complexity factor for SVM algorithm (both classification and regression).  Default value estimated from the data by the algorithm.
SVMS_CONV_TOLERANCE	TO_CHAR( <i>numeric_exp</i> <i>r</i> >0)	Convergence tolerance for SVM algorithm.  Default is 0.001.
SVMS_EPSILON	TO_CHAR( <i>numeric_exp</i> <i>r</i> >0)	Value of epsilon factor for SVM regression.  Default value estimated from the data by the algorithm.
SVMS_KERNEL_CACHE_SIZE	TO_CHAR( <i>numeric_exp</i> <i>r</i> >0)	Value of kernel cache size for SVM algorithm. Applies to Gaussian kernel only.  Default is 50000000 bytes.
SVMS_KERNEL_FUNCTION	svm_gaussian svms_linear	Kernel for Support Vector Machine. The default is determined by the algorithm based on the number of attributes in the training data. When there are many attributes, the algorithm uses a linear kernel, otherwise it uses a nonlinear (Gaussian) kernel.  The number of attributes does not correspond to the number of columns in the training data. The algorithm explodes categorical attributes to binary, numeric attributes. In addition, Oracle Data Mining handles each row in a nested column as a separate attribute. SVM takes these factors into account when choosing the kernel function.
SVMS_OUTLIER_RATE	TO_CHAR( <i>0</i> < <i>numeric_expr</i> <1)	The desired rate of outliers in the training data. Valid for One-Class SVM models only (anomaly detection).  Default is .1.
SVMS_STD_DEV	TO_CHAR( <i>numeric_exp</i> <i>r</i> >0)	Value of standard deviation for SVM algorithm.  This is applicable only for Gaussian kernel.  Default value estimated from the data by the algorithm.

**See Also:** *Oracle Data Mining Concepts* for information about SVM

## Data Types

The `DBMS_DATA_MINING` package uses object data types to store information about model attributes. Most of these types are returned by the table functions `GET_n`, where *n* identifies the type of information to return. These functions take a model name as input and return the requested information as a collection of rows.

For a list of the `GET` functions, see "[Summary of DBMS\\_DATA\\_MINING Subprograms](#)" on page 44-23.

Oracle Data Mining also uses object data types for handling transactional data. These types, `DM_NESTED_NUMERICALS` and `DM_NESTED_CATEGORICALS` specify nested tables that can be used for storing a set of mining attributes in a single column. For more information on nested tables, see the *Oracle Data Mining Application Developer's Guide*.

All the table functions use **pipelining**, which causes each row of output to be materialized as it is read from model storage, without waiting for the generation of the complete table object. For more information on pipelined, parallel table functions, consult the *Oracle Database PL/SQL Language Reference*.

The Data Mining object data types are described in [Table 44-17](#).

**Table 44-17 DBMS\_DATA\_MINING Summary of Data Types**

Data Type	Description
<code>DM_ABN_DETAIL</code>	Information about an attribute in an Adaptive Bayes Network model.
<code>DM_ABN_DETAILS</code>	A collection of <code>DM_ABN_DETAIL</code> . Returned by <a href="#">GET_MODEL_DETAILS_ABN Function</a> .
<code>DM_CENTROID</code>	The centroid of a cluster.
<code>DM_CENTROIDS</code>	A collection of <code>DM_CENTROID</code> . A member of <code>DM_CLUSTER</code> .
<code>DM_CHILD</code>	A child node of a cluster.
<code>DM_CHILDREN</code>	A collection of <code>DM_CHILD</code> . A member of <code>DM_CLUSTER</code> .
<code>DM_CLUSTER</code>	A cluster. A cluster includes <code>DM_PREDICATES</code> , <code>DM_CHILDREN</code> , <code>DM_CENTROIDS</code> , and <code>DM_HISTOGRAMS</code> . It also includes a <code>DM_RULE</code> .
<code>DM_CLUSTERS</code>	A collection of <code>DM_CLUSTER</code> . Returned by <a href="#">GET_MODEL_DETAILS_KM Function</a> and <a href="#">GET_MODEL_DETAILS_OC Function</a> .
<code>DM_CONDITIONAL</code>	The conditional probability of an attribute in a Naive Bayes model.
<code>DM_CONDITIONALS</code>	A collection of <code>DM_CONDITIONAL</code> . Returned by <a href="#">GET_MODEL_DETAILS_NB Function</a> .
<code>DM_COST_ELEMENT</code>	The actual and predicted values in a cost matrix.
<code>DM_COST_MATRIX</code>	A collection of <code>DM_COST_ELEMENT</code> . Returned by <a href="#">GET_MODEL_COST_MATRIX Function</a> .
<code>DM_GLM_COEFF</code>	The coefficient and associated statistics of an attribute in a Generalized Linear Model.
<code>DM_GLM_COEFF_SET</code>	A collection of <code>DM_GLM_COEFF</code> . Returned by <a href="#">GET_MODEL_DETAILS_GLM Function</a> .
<code>DM_HISTOGRAM_BIN</code>	A histogram associated with a cluster.



**Table 44–17 (Cont.) DBMS\_DATA\_MINING Summary of Data Types**

<b>Data Type</b>	<b>Description</b>
DM_HISTOGRAMS	A collection of DM_HISTOGRAM_BIN. A member of DM_CLUSTER.
DM_ITEM	An item in an association rule.
DM_ITEMS	A collection of DM_ITEM.
DM_ITEMSET	A collection of DM_ITEMS.
DM_ITEMSETS	A collection of DM_ITEMSET. Returned by <a href="#">GET_FREQUENT_ITEMSETS Function</a> .
DM_MODEL_GLOBAL_DETAIL	High-level statistics about a model.
DM_MODEL_GLOBAL_DETAILS	A collection of DM_MODEL_GLOBAL_DETAIL. Returned by <a href="#">GET_MODEL_DETAILS_GLOBAL Function</a> .
DM_MODEL_SETTING	A model setting.
DM_MODEL_SETTINGS	A collection of DM_MODEL_SETTING. Returned by <a href="#">GET_MODEL_SETTINGS Function</a> and <a href="#">GET_DEFAULT_SETTINGS Function</a> .
DM_MODEL_SIGNATURE_ATTRIBUTE	An attribute in the model signature.
DM_MODEL_SIGNATURE	A collection of DM_MODEL_SIGNATURE. Returned by <a href="#">GET_MODEL_SIGNATURE Function</a> .
DM_NB_DETAIL	Information about an attribute in a Naive Bayes model.
DM_NB_DETAILS	A collection of DM_DB_DETAIL. Returned by <a href="#">GET_MODEL_DETAILS_NB Function</a> .
DM_NESTED_CATEGORICAL	The name and value of a categorical attribute.
DM_NESTED_CATEGORICALS	A collection of DM_NESTED_CATEGORICAL. A collection of attributes defined as a single model attribute. Transactional data must be defined as nested attributes for Data Mining.
DM_NESTED_NUMERICAL	The name and value of a numerical attribute.
DM_NESTED_NUMERICALS	A collection of DM_NESTED_NUMERICAL. A collection of attributes defined as a single model attribute. Transactional data must be defined as nested attributes for Data Mining.
DM_NMF_ATTRIBUTE	An attribute in a feature of a Non-Negative Matrix Factorization model.
DM_NMF_ATTRIBUTE_SET	A collection of DM_NMF_ATTRIBUTE. A member of DM_NMF_FEATURE.
DM_NMF_FEATURE	A feature in a Non-Negative Matrix Factorization model.
DM_NMF_FEATURE_SET	A collection of DM_NMF_FEATURE. Returned by <a href="#">GET_MODEL_DETAILS_NMF Function</a> .
DM_PREDICATE	Antecedent and consequent attributes.
DM_PREDICATES	A collection of DM_PREDICATE. A member of DM_RULE, DM_CLUSTER, and DM_ABN_DETAIL.
DM_RANKED_ATTRIBUTE	An attribute ranked by its importance in an Attribute Importance model.
DM_RANKED_ATTRIBUTES	A collection of DM_RANKED_ATTRIBUTE. Returned by <a href="#">GET_MODEL_DETAILS_AI Function</a> .

**Table 44–17 (Cont.) DBMS\_DATA\_MINING Summary of Data Types**

Data Type	Description
DM_RULE	<p>A rule that defines a conditional relationship.</p> <p>The rule can be one of the association rules returned by <a href="#">GET_ASSOCIATION_RULES Function</a>, or it can be a rule associated with a cluster in the collection of clusters returned by <a href="#">GET_MODEL_DETAILS_KM Function</a> and <a href="#">GET_MODEL_DETAILS_OC Function</a>.</p>
DM_RULES	<p>A collection of DM_RULE. Returned by <a href="#">GET_ASSOCIATION_RULES Function</a>.</p>
DM_SVM_ATTRIBUTE	<p>The name, value, and coefficient of an attribute in a Support Vector Machine model.</p>
DM_SVM_ATTRIBUTE_SET	<p>A collection of DM_SVM_ATTRIBUTE. Returned by <a href="#">GET_MODEL_DETAILS_SVM Function</a>. Also a member of DM_SVM_LINEAR_COEFF.</p>
DM_SVM_LINEAR_COEFF	<p>The linear coefficient of each attribute in a Support Vector Machine model.</p>
DM_SVM_LINEAR_COEFF_SET	<p>A collection of DM_SVM_LINEAR_COEFF. Returned by <a href="#">GET_MODEL_DETAILS_SVM Function</a> for an SVM model built using the linear kernel.</p>
DM_TRANSFORM	<p>The transformation and reverse transformation expressions for an attribute.</p>
DM_TRANSFORMS	<p>A collection of DM_TRANSFORM. Returned by <a href="#">GET_MODEL_TRANSFORMATIONS Function</a>.</p>
TRANSFORM_LIST	<p>A list of user-specified transformations for a model. Accepted as a parameter by the <a href="#">CREATE_MODEL Procedure</a>.</p> <p>This collection type is defined in the <a href="#">DBMS_DATA_MINING_TRANSFORM</a> package.</p>

## Summary of DBMS\_DATA\_MINING Subprograms

Table 44–18 summarizes the subprograms included in the DBMS\_DATA\_MINING package.

**Table 44–18 DBMS\_DATA\_MINING Package Subprograms**

Data Type	Purpose
<a href="#">ADD_COST_MATRIX Procedure</a> on page 44-25	Adds a cost matrix to a classification model
<a href="#">ALTER_REVERSE_EXPRESSION Procedure</a> on page 44-27	Changes the reverse transformation expression to an expression that you specify
<a href="#">APPLY Procedure</a> on page 44-30	Applies a model to a data set (scores the data)
<a href="#">COMPUTE_CONFUSION_MATRIX Procedure</a> on page 44-34	Computes the confusion matrix from the <code>APPLY</code> results on test data for a classification model; also provides the accuracy of the model
<a href="#">COMPUTE_LIFT Procedure</a> on page 44-40	Computes lift for a given positive target value from the <code>APPLY</code> results on test data for a classification model
<a href="#">COMPUTE_ROC Procedure</a> on page 44-45	Computes Receiver Operating Characteristic (ROC) for a classification model
<a href="#">CREATE_MODEL Procedure</a> on page 44-50	Creates (builds) a model
<a href="#">DROP_MODEL Procedure</a> on page 44-54	Drops a model
<a href="#">EXPORT_MODEL Procedure</a> on page 44-55	Exports a model to a dump file
<a href="#">GET_ASSOCIATION_RULES Function</a> on page 44-58	Returns the rules from an association model
<a href="#">GET_DEFAULT_SETTINGS Function</a> on page 44-62	Returns all the default settings for all mining functions and algorithms
<a href="#">GET_FREQUENT_ITEMSETS Function</a> on page 44-63	Returns the frequent itemsets for an association model
<a href="#">GET_MODEL_COST_MATRIX Function</a> on page 44-65	Returns the cost matrix for a model
<a href="#">GET_MODEL_DETAILS_ABN Function</a> on page 44-67	Returns the details of an Adaptive Bayes Network model
<a href="#">GET_MODEL_DETAILS_AI Function</a> on page 44-69	Returns the details of an Attribute Importance model
<a href="#">GET_MODEL_DETAILS_GLM Function</a> on page 44-71	Returns the details of a Generalized Linear Model
<a href="#">GET_MODEL_DETAILS_GLOBAL Function</a> on page 44-74	Returns high-level statistics about a model
<a href="#">GET_MODEL_DETAILS_KM Function</a> on page 44-77	Returns the details of a <i>k</i> -Means model
<a href="#">GET_MODEL_DETAILS_NB Function</a> on page 44-80	Returns the details of a Naive Bayes model
<a href="#">GET_MODEL_DETAILS_NMF Function</a> on page 44-82	Returns the details of an NMF model

**Table 44–18 (Cont.) DBMS\_DATA\_MINING Package Subprograms**

<b>Data Type</b>	<b>Purpose</b>
<a href="#">GET_MODEL_DETAILS_OC Function</a> on page 44-84	Returns the details of an O-Cluster model
<a href="#">GET_MODEL_DETAILS_SVM Function</a> on page 44-88	Returns the details of an SVM model with a linear kernel
<a href="#">GET_MODEL_DETAILS_XML Function</a> on page 44-90	Returns the details of a Decision Tree model
<a href="#">GET_MODEL_SETTINGS Function</a> on page 44-92	Returns the settings used to build a model
<a href="#">GET_MODEL_SIGNATURE Function</a> on page 44-93	Returns the signature of a model
<a href="#">GET_MODEL_TRANSFORMATIONS Function</a> on page 44-95	Returns the user-specified transformation definitions embedded in the model, as well as many of the ADP transformations
<a href="#">GET_TRANSFORM_LIST Procedure</a> on page 44-97	Converts between two different transformation specification formats
<a href="#">IMPORT_MODEL Procedure</a> on page 44-100	Imports a model into a user schema
<a href="#">RANK_APPLY Procedure</a> on page 44-104	Ranks the predictions from the APPLY results for a classification model
<a href="#">REMOVE_COST_MATRIX Procedure</a> on page 44-107	Removes a cost matrix from a model
<a href="#">RENAME_MODEL Procedure</a> on page 44-108	Renames a model

## ADD\_COST\_MATRIX Procedure

This procedure associates a cost matrix table with a classification model. The cost matrix biases the model by assigning costs or benefits to specific model outcomes.

The cost matrix is stored with the model and taken into account when the model is scored. The stored cost matrix is the default scoring matrix for the model.

You can also specify a cost matrix inline when you invoke a Data Mining SQL function for scoring. When an inline cost matrix is specified, it is used instead of the default, stored cost matrix (if one exists).

To obtain the default scoring matrix for a model, use the `GET_MODEL_COST_MATRIX` function. To remove the default scoring matrix from a model, use the `REMOVE_COST_MATRIX` procedure. See "[GET\\_MODEL\\_COST\\_MATRIX Function](#)" on page 44-65 and "[REMOVE\\_COST\\_MATRIX Procedure](#)" on page 44-107.

### See Also:

- "Biasing a Classification Model" in *Oracle Data Mining Concepts* for more information about costs
- *Oracle Database SQL Language Reference* for syntax of inline cost matrix

## Syntax

```
DBMS_DATA_MINING.ADD_COST_MATRIX (
    model_name           IN VARCHAR2,
    cost_matrix_table_name IN VARCHAR2,
    cost_matrix_schema_name IN VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 44–19 ADD\_COST\_MATRIX Procedure Parameters**

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is assumed.
<code>cost_matrix_table_name</code>	Name of the cost matrix table (described in <a href="#">Table 44–20</a> ).
<code>cost_matrix_schema_name</code>	Schema of the cost matrix table. If no schema is specified, the current schema is used.

## Usage Notes

1. If the model is not in your schema, then `ADD_COST_MATRIX` requires the `ALTER ANY MINING MODEL` system privilege or the `ALTER` object privilege for the mining model.
2. The cost matrix table must have the columns shown in [Table 44–20](#). Note that the actual and predicted target values must have the same data type.

**Table 44–20 Required Columns in a Cost Matrix Table**

Column Name	Data Type
<code>ACTUAL_TARGET_VALUE</code>	<code>VARCHAR2 (4000)</code> for categorical targets <code>NUMBER</code> for numeric targets

**Table 44–20 (Cont.) Required Columns in a Cost Matrix Table**

Column Name	Data Type
PREDICTED_TARGET_VALUE	VARCHAR2 (4000) for categorical targets NUMBER for numeric targets
COST	NUMBER

- Since a benefit can be viewed as a negative cost, you can specify a benefit for a given outcome by providing a negative number in the `costs` column of the cost matrix table.
- All classification algorithms can use a cost matrix for scoring. The Decision Tree algorithm can also use a cost matrix at build time. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the `CLAS_COST_TABLE_NAME` setting in the settings table for the model. See [Table 44–7, "Mining Function Settings"](#).

The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, use the `REMOVE_COST_MATRIX` procedure to remove the cost matrix and the `ADD_COST_MATRIX` procedure to add a new one.

## Example

This example creates a cost matrix table called `COSTS_NB` and adds it to a Naive Bayes model called `NB_SH_CLAS_SAMPLE`. The model has a binary target: 1 means that the customer responds to a promotion; 0 means that the customer does not respond. The cost matrix assigns a cost of .25 to misclassifications of customers who do not respond and a cost of .75 to misclassifications of customers who do respond. This means that it is three times more costly to misclassify responders than it is to misclassify non-responders.

```
CREATE TABLE costs_nb (
  actual_target_value      NUMBER,
  predicted_target_value   NUMBER,
  cost                     NUMBER);
INSERT INTO costs_nb values (0, 0, 0);
INSERT INTO costs_nb values (0, 1, .25);
INSERT INTO costs_nb values (1, 0, .75);
INSERT INTO costs_nb values (1, 1, 0);
COMMIT;

EXEC dbms_data_mining.add_cost_matrix('nb_sh_clas_sample', 'costs_nb');

SELECT cust_gender, COUNT(*) AS cnt, ROUND(AVG(age)) AS avg_age
FROM mining_data_apply_v
WHERE PREDICTION(nb_sh_clas_sample COST MODEL
  USING cust_marital_status, education, household_size) = 1
GROUP BY cust_gender
ORDER BY cust_gender;
```

C	CNT	AVG_AGE
F	72	39
M	555	44

## ALTER\_REVERSE\_EXPRESSION Procedure

This procedure replaces a reverse transformation expression with an expression that you specify. If the attribute does not have a reverse expression, the procedure creates one from the specified expression.

You can also use this procedure to customize the output of clustering, feature extraction, and anomaly detection models.

### Syntax

```
DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION (
    model_name          VARCHAR2,
    expression          CLOB,
    attribute_name      VARCHAR2 DEFAULT NULL,
    attribute_subname   VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 44–21 ALTER\_REVERSE\_EXPRESSION Procedure Parameters**

Parameter	Description
model_name	Name of the model in the form [ <i>schema_name</i> .] <i>model_name</i> . If you do not specify a schema, your own schema is used.
expression	A SQL expression
attribute_name	Name of the attribute. Specify NULL if you wish to apply <i>expression</i> to a cluster, feature, or One-Class SVM prediction.
attribute_subname	Name of the nested attribute if <i>attribute_name</i> is a nested column, otherwise NULL.

### Usage Notes

1. For purposes of model transparency, Oracle Data Mining provides reverse transformations for transformations that are embedded in a model. Reverse transformations are used in model details and in the results of scoring.

**See Also:** [About Transformation Lists](#) in [Chapter 45](#)

---

**Note:** Use caution when altering the reverse expression for the target of a model that has a cost matrix. If you specify a reverse expression that is inconsistent with the target values in the cost matrix table, you will not be able to score the model.

See "[ADD\\_COST\\_MATRIX Procedure](#)" on page 44-25 and *Oracle Data Mining Concepts* for information about cost matrixes.

---

2. To prevent reverse transformation of an attribute, you can specify NULL for *expression*.
3. You can use ALTER\_REVERSE\_EXPRESSION to label clusters produced by clustering models and features produced by feature extraction.

You can use ALTER\_REVERSE\_EXPRESSION to replace the zeros and ones returned by anomaly-detection models. By default, anomaly-detection models label anomalous records with 0 and all other records with 1.

**See Also:** *Oracle Data Mining Concepts* for information about anomaly detection

## Examples

1. In this example, the target (`affinity_card`) of the model `CLASS_MODEL` is manipulated internally as `yes` or `no` instead of `1` or `0` but returned as `1s` and `0s` when scored. The `ALTER_REVERSE_EXPRESSION` procedure causes the target values to be returned as `TRUE` or `FALSE`.

The data sets `MINING_DATA_BUILD` and `MINING_DATA_TEST` are included with the Oracle Data Mining sample programs. See *Oracle Data Mining Administrator's Guide* for information about the sample programs.

```

DECLARE
    v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.SET_TRANSFORM(v_xlst,
        'affinity_card', NULL,
        'decode(affinity_card, 1, 'yes', 'no')',
        'decode(affinity_card, 'yes', 1, 0)');
    dbms_data_mining.CREATE_MODEL(
        model_name          => 'CLASS_MODEL',
        mining_function     => dbms_data_mining.classification,
        data_table_name    => 'mining_data_build',
        case_id_column_name => 'cust_id',
        target_column_name  => 'affinity_card',
        settings_table_name => NULL,
        data_schema_name   => 'dmuser',
        settings_schema_name => NULL,
        xform_list         => v_xlst );
END;
/
SELECT cust_income_level, occupation,
       PREDICTION(CLASS_MODEL USING *) predict_response
FROM   mining_data_test WHERE age = 60 AND cust_gender IN 'M'
ORDER BY cust_income_level;

```

CUST_INCOME_LEVEL	OCCUPATION	PREDICT_RESPONSE
A: Below 30,000	Transp.	1
E: 90,000 - 109,999	Transp.	1
E: 90,000 - 109,999	Sales	1
G: 130,000 - 149,999	Handler	0
G: 130,000 - 149,999	Crafts	0
H: 150,000 - 169,999	Prof.	1
J: 190,000 - 249,999	Prof.	1
J: 190,000 - 249,999	Sales	1

```

BEGIN
    dbms_data_mining.ALTER_REVERSE_EXPRESSION (
        model_name          => 'CLASS_MODEL',
        expression         => 'decode(affinity_card, 'yes', 'TRUE', 'FALSE')',
        attribute_name     => 'affinity_card');
END;
/
column predict_response on
column predict_response format a20
SELECT cust_income_level, occupation,
       PREDICTION(CLASS_MODEL USING *) predict_response

```



```
FROM mining_data_test WHERE age = 60 AND cust_gender IN 'M'
ORDER BY cust_income_level;
```

CUST_INCOME_LEVEL	OCCUPATION	PREDICT_RESPONSE
A: Below 30,000	Transp.	TRUE
E: 90,000 - 109,999	Transp.	TRUE
E: 90,000 - 109,999	Sales	TRUE
G: 130,000 - 149,999	Handler	FALSE
G: 130,000 - 149,999	Crafts	FALSE
H: 150,000 - 169,999	Prof.	TRUE
J: 190,000 - 249,999	Prof.	TRUE
J: 190,000 - 249,999	Sales	TRUE

2. This example specifies labels for the clusters that result from the `sh_clus` model. The labels consist of the word "Cluster" and the internal numeric identifier for the cluster.

```
BEGIN
  dbms_data_mining.ALTER_REVERSE_EXPRESSION('sh_clus', ''Cluster ''||value');
END;
/
```

```
SELECT cust_id, cluster_id(sh_clus using *) cluster_id
FROM sh_aprep_num
WHERE cust_id < 100011
ORDER by cust_id;
```

CUST_ID	CLUSTER_ID
100001	Cluster 18
100002	Cluster 14
100003	Cluster 14
100004	Cluster 18
100005	Cluster 19
100006	Cluster 7
100007	Cluster 18
100008	Cluster 14
100009	Cluster 8
100010	Cluster 8

## APPLY Procedure

This procedure applies a mining model to the data of interest, and generates the results in a table. The apply process is also referred to as **scoring**.

For predictive mining functions, the apply process generates predictions in a target column. For descriptive mining functions such as clustering, the apply process assigns each case to a cluster with a probability.

In Oracle Data Mining, the apply operation is not applicable to association models and attribute importance models.

---



---

**Note:** Scoring can also be performed directly in SQL using the Data Mining functions. See

- "Data Mining Functions" in *Oracle Database SQL Language Reference*
  - "Scoring and Deployment" in *Oracle Data Mining Application Developer's Guide* and
- 
- 

### Syntax

```
DBMS_DATA_MINING.APPLY (
    model_name          IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    case_id_column_name IN VARCHAR2,
    result_table_name   IN VARCHAR2,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 44–22** *APPLY Procedure Parameters*

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.
data_table_name	Name of table or view representing data to be scored
case_id_column_name	Name of the case identifier column
result_table_name	Name of the table to store apply results
data_schema_name	Name of the schema containing the data to be scored

### Usage Notes

1. The data provided for APPLY must undergo the same preprocessing as the data used to create and test the model. When you use Automatic Data Preparation, the preprocessing required by the algorithm is handled for you by the model — both at build time and apply time. (See "[Automatic Data Preparation](#)" on page 44-11.)
2. APPLY creates a table in the user's schema to hold the results. The columns are algorithm-specific.

The columns in the results table are listed in [Table 44–23](#) through [Table 44–27](#). The case ID column name in the results table will match the case ID column name provided by you. The type of the incoming case ID column is also preserved in APPLY output.

---

**Note:** Make sure that the case ID column does not have the same name as one of the columns that will be created by APPLY. For example, when applying a classification model, the case ID in the scoring data must not be 'PREDICTION' or 'PROBABILITY' (See [Table 44-23](#)).

---

- The data type for the 'PREDICTION', 'CLUSTER\_ID', and 'FEATURE\_ID' output columns is influenced by any reverse expression that is embedded in the model by the user. If the user does not provide a reverse expression that alters the scored value type, then the types will conform to the descriptions in the following tables. See "[ALTER\\_REVERSE\\_EXPRESSION Procedure](#)" on page 44-27.

## Classification

The results table for classification has the columns described in [Table 44-23](#). If the target of the model is categorical, the PREDICTION column will have a VARCHAR2 data type. If the target is numerical, the PREDICTION column will have a NUMBER data type.

**Table 44-23** APPLY Results Table for Classification

Column Name	Data Type
<i>Case ID column name</i>	VARCHAR2 or NUMBER
PREDICTION	VARCHAR2 or NUMBER
PROBABILITY	NUMBER

## One-Class SVM (Anomaly Detection)

The results table for anomaly detection has the columns described in [Table 44-24](#).

**Table 44-24** APPLY Results Table for Anomaly Detection

Column Name	Data Type
<i>Case ID column name</i>	VARCHAR2 or NUMBER
PREDICTION	NUMBER
PROBABILITY	NUMBER

Values in the PREDICTION column can be either 0 or 1. When the prediction is 1, the case is a typical example. When the prediction is 0, the case is an outlier.

## Regression using SVM or GLM

The results table for regression has the columns described in [Table 44-25](#).

**Table 44-25** APPLY Results Table for Regression

Column Name	Data Type
<i>Case ID column name</i>	VARCHAR2 or NUMBER
PREDICTION	NUMBER

## Clustering using *k*-Means or O-Cluster

Clustering is an unsupervised mining function, and hence there are no targets. The results of an `APPLY` operation will contain simply the cluster identifier corresponding to a case, and the associated probability. The results table has the columns described in [Table 44–26](#).

**Table 44–26** *APPLY Results Table for Clustering*

Column Name	Data Type
<i>Case ID column name</i>	VARCHAR2 or NUMBER
CLUSTER_ID	NUMBER
PROBABILITY	NUMBER

## Feature Extraction using NMF

Feature extraction is also an unsupervised mining function, and hence there are no targets. The results of an `APPLY` operation will contain simply the feature identifier corresponding to a case, and the associated match quality. The results table has the columns described in [Table 44–27](#).

**Table 44–27** *APPLY Results Table for Feature Extraction*

Column Name	Data Type
<i>Case ID column name</i>	VARCHAR2 or NUMBER
FEATURE_ID	NUMBER
MATCH_QUALITY	NUMBER

## Examples

This example applies the GLM regression model `GLMR_SH_REGR_SAMPLE` to the data in the `MINING_DATA_APPLY_V` view. The apply results are output to the table `REGRESSION_APPLY_RESULT`.

```
SQL> BEGIN
      DBMS_DATA_MINING.APPLY (
        model_name      => 'glmr_sh_regr_sample',
        data_table_name => 'mining_data_apply_v',
        case_id_column_name => 'cust_id',
        result_table_name => 'regression_apply_result');
      END;
      /

SQL> SELECT * FROM regression_apply_result WHERE cust_id > 101485;

CUST_ID PREDICTION
-----
101486 22.8048824
101487 25.0261101
101488 48.6146619
101489 51.82595
101490 22.6220714
101491 61.3856816
101492 24.1400748
101493 58.034631
101494 45.7253149
101495 26.9763318
```

101496 48.1433425  
101497 32.0573434  
101498 49.8965531  
101499 56.270656  
101500 21.1153047

## COMPUTE\_CONFUSION\_MATRIX Procedure

This procedure computes a confusion matrix, stores it in a table in the user's schema, and returns the model accuracy.

A confusion matrix is a test metric for classification models. It compares the predictions generated by the model with the actual target values in a set of test data. The matrix is  $n$ -by- $n$ , where  $n$  is the number of classes. The confusion matrix lists the number of times each class was correctly predicted and the number of times it was predicted to be one of the other classes.

COMPUTE\_CONFUSION\_MATRIX accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

### See Also:

*Oracle Data Mining Concepts* for more details about confusion matrixes and other test metrics for classification

["COMPUTE\\_LIFT Procedure"](#) on page 44-40

["COMPUTE\\_ROC Procedure"](#) on page 44-45

## Syntax

```
DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
    accuracy                OUT NUMBER,
    apply_result_table_name IN  VARCHAR2,
    target_table_name       IN  VARCHAR2,
    case_id_column_name     IN  VARCHAR2,
    target_column_name      IN  VARCHAR2,
    confusion_matrix_table_name IN VARCHAR2,
    score_column_name       IN  VARCHAR2 DEFAULT 'PREDICTION',
    score_criterion_column_name IN VARCHAR2 DEFAULT 'PROBABILITY',
    cost_matrix_table_name  IN  VARCHAR2 DEFAULT NULL,
    apply_result_schema_name IN  VARCHAR2 DEFAULT NULL,
    target_schema_name     IN  VARCHAR2 DEFAULT NULL,
    cost_matrix_schema_name IN  VARCHAR2 DEFAULT NULL,
    score_criterion_type    IN  VARCHAR2 DEFAULT 'PROBABILITY');
```

## Parameters

**Table 44–28 COMPUTE\_CONFUSION\_MATRIX Procedure Parameters**

Parameter	Description
accuracy	Output parameter containing the overall percentage accuracy of the predictions.
apply_result_table_name	Table containing the predictions.
target_table_name	Table containing the known target values from the test data.
case_id_column_name	Case ID column in the apply results table. Must match the case identifier in the targets table.
target_column_name	Target column in the targets table. Contains the known target values from the test data.
confusion_matrix_table_name	Table containing the confusion matrix. The table will be created by the procedure in the user's schema. The columns in the confusion matrix table are described in the Usage Notes.
score_column_name	Column containing the predictions in the apply results table. The default column name is PREDICTION, which is the default name created by the APPLY procedure (See "APPLY Procedure" on page 44-30).
score_criterion_column_name	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions. By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted. The score_criterion_type parameter indicates whether probabilities or costs will be used for scoring. The default column name is 'PROBABILITY', which is the default name created by the APPLY procedure (See "APPLY Procedure" on page 44-30). See the Usage Notes for additional information.
cost_matrix_table_name	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to 'COSTS', the costs in this table will be used as the scoring criteria. The columns in a cost matrix table are described in the Usage Notes.
apply_result_schema_name	Schema of the apply results table. If null, the user's schema is assumed.
target_schema_name	Schema of the table containing the known targets. If null, the user's schema is assumed.
cost_matrix_schema_name	Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.

**Table 44–28 (Cont.) COMPUTE\_CONFUSION\_MATRIX Procedure Parameters**

Parameter	Description
score_criterion_type	<p>Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter.</p> <p>The default value of score_criterion_type is 'PROBABILITY'. To use costs as the scoring criterion, specify 'COST'.</p> <p>If score_criterion_type is set to 'COST' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring.</p> <p>See the Usage Notes and the Examples.</p>

**Usage Notes**

- The predictive information you pass to COMPUTE\_CONFUSION\_MATRIX may be generated using SQL PREDICTION functions, the DBMS\_DATA\_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the confusion matrix.
- Instead of passing a cost matrix to COMPUTE\_CONFUSION\_MATRIX, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL PREDICTION\_COST function to populate the score criterion column.
- The predictions that you pass to COMPUTE\_CONFUSION\_MATRIX are in a table or view specified in apply\_result\_table\_name.

```
CREATE TABLE apply_result_table_name AS (
    case_id_column_name          VARCHAR2,
    score_column_name           VARCHAR2,
    score_criterion_column_name VARCHAR2);
```

- A cost matrix must have the columns described in [Table 44–29](#).

**Table 44–29 Columns in a Cost Matrix**

Column Name	Data Type
actual_target_value	NUMBER or VARCHAR2
predicted_target_value	NUMBER or VARCHAR2
cost	NUMBER

**See Also:** *Oracle Data Mining Concepts* for more information about cost matrixes

- The confusion matrix created by COMPUTE\_CONFUSION\_MATRIX has the columns described in [Table 44–30](#).

**Table 44–30 Columns in a Confusion Matrix**

Column Name	Data Type
actual_target_value	NUMBER or VARCHAR2



**Table 44–30 (Cont.) Columns in a Confusion Matrix**

Column Name	Data Type
predicted_target_value	NUMBER or VARCHAR2
value	NUMBER

**See Also:** *Oracle Data Mining Concepts* for more information about confusion matrixes

## Examples

These examples use the Naive Bayes model `nb_sh_clas_sample`, which is created by one of the Oracle Data Mining sample programs.

### Compute a Confusion Matrix Based on Probabilities

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
  SELECT cust_id,
         PREDICTION(nb_sh_clas_sample USING *) prediction,
         PREDICTION_PROBABILITY(nb_sh_clas_sample USING *) probability
  FROM mining_data_test_v;
```

Using probabilities as the scoring criterion, you can compute the confusion matrix as follows.

```
DECLARE
  v_accuracy NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
    accuracy => v_accuracy,
    apply_result_table_name => 'nb_apply_results',
    target_table_name => 'mining_data_test_v',
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card',
    confusion_matrix_table_name => 'nb_confusion_matrix',
    score_column_name => 'PREDICTION',
    score_criterion_column_name => 'PROBABILITY'
    cost_matrix_table_name => null,
    apply_result_schema_name => null,
    target_schema_name => null,
    cost_matrix_schema_name => null,
    score_criterion_type => 'PROBABILITY');
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/
```

The confusion matrix and model accuracy are shown as follows.

```
**** MODEL ACCURACY ****: .7847
```

```
SQL>SELECT * from nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
-----
                1                0                60
                0                0               891
                1                1               286
                0                1               263
```

### Compute a Confusion Matrix Based on a Cost Matrix Table

The confusion matrix in the previous example shows a high rate of false positives. For 263 cases, the model predicted 1 when the actual value was 0. You could use a cost matrix to minimize this type of error.

The cost matrix table `nb_cost_matrix` specifies that a false positive is 3 times more costly than a false negative.

```
SQL> SELECT * from nb_cost_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      COST
-----
                0                0          0
                0                1          .75
                1                0          .25
                1                1           0
```

This statement shows how to generate the predictions using `APPLY`.

```
BEGIN
  DBMS_DATA_MINING.APPLY(
    model_name      => 'nb_sh_clas_sample',
    data_table_name => 'mining_data_test_v',
    case_id_column_name => 'cust_id',
    result_table_name => 'nb_apply_results');
END;
/
```

This statement computes the confusion matrix using the cost matrix table. The score criterion column is named 'PROBABILITY', which is the name generated by `APPLY`.

```
DECLARE
  v_accuracy NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
    accuracy              => v_accuracy,
    apply_result_table_name => 'nb_apply_results',
    target_table_name     => 'mining_data_test_v',
    case_id_column_name   => 'cust_id',
    target_column_name    => 'affinity_card',
    confusion_matrix_table_name => 'nb_confusion_matrix',
    score_column_name     => 'PREDICTION',
    score_criterion_column_name => 'PROBABILITY',
    cost_matrix_table_name => 'nb_cost_matrix',
    apply_result_schema_name => null,
    target_schema_name     => null,
    cost_matrix_schema_name => null,
    score_criterion_type  => 'COST');
  DBMS_OUTPUT.PUT_LINE('**** MODEL ACCURACY ****: ' || ROUND(v_accuracy,4));
END;
/
```

The resulting confusion matrix shows a decrease in false positives (212 instead of 263).

```
**** MODEL ACCURACY ****: .798

SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
-----
                1                0          91
```

0	0	942
1	1	255
0	1	212

### Compute a Confusion Matrix Based on Embedded Costs

You can use the `ADD_COST_MATRIX` procedure to embed a cost matrix in a model. The embedded costs can be used instead of probabilities for scoring. This statement adds the previously-defined cost matrix to the model.

```
BEGIN
  DBMS_DATA_MINING.ADD_COST_MATRIX ('nb_sh_clas_sample', 'nb_cost_matrix');
END;
/
```

The following statement applies the model to the test data using the embedded costs and stores the results in a table.

```
CREATE TABLE nb_apply_results AS
  SELECT cust_id,
         PREDICTION(nb_sh_clas_sample COST MODEL USING *) prediction,
         PREDICTION_COST(nb_sh_clas_sample COST MODEL USING *) cost
  FROM mining_data_test_v;
```

You can compute the confusion matrix using the embedded costs.

```
DECLARE
  v_accuracy          NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_CONFUSION_MATRIX (
    accuracy           => v_accuracy,
    apply_result_table_name => 'nb_apply_results',
    target_table_name  => 'mining_data_test_v',
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card',
    confusion_matrix_table_name => 'nb_confusion_matrix',
    score_column_name  => 'PREDICTION',
    score_criterion_column_name => 'COST',
    cost_matrix_table_name => null,
    apply_result_schema_name => null,
    target_schema_name  => null,
    cost_matrix_schema_name => null,
    score_criterion_type => 'COST');
END;
/
```

The results are:

```
**** MODEL ACCURACY ****: .798
```

```
SQL> SELECT * FROM nb_confusion_matrix;
ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE      VALUE
-----
                1                0            91
                0                0            942
                1                1            255
                0                1            212
```

## COMPUTE\_LIFT Procedure

This procedure computes lift and stores the results in a table in the user's schema.

Lift is a test metric for binary classification models. To compute lift, one of the target values must be designated as the positive class. `COMPUTE_LIFT` compares the predictions generated by the model with the actual target values in a set of test data. Lift measures the degree to which the model's predictions of the positive class are an improvement over random chance.

Lift is computed on scoring results that have been ranked by probability (or cost) and divided into quantiles. Each quantile includes the scores for the same number of cases.

`COMPUTE_LIFT` calculates quantile-based and cumulative statistics. The number of quantiles and the positive class are user-specified. Additionally, `COMPUTE_LIFT` accepts three input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing either probabilities or costs associated with the predictions
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values
- (Optional) A cost matrix table with predefined columns. See the Usage Notes for the column requirements.

### See Also:

*Oracle Data Mining Concepts* for more details about lift and test metrics for classification

["COMPUTE\\_CONFUSION\\_MATRIX Procedure"](#) on page 44-34

["COMPUTE\\_ROC Procedure"](#) on page 44-45

## Syntax

```
DBMS_DATA_MINING.COMPUTE_LIFT (
  apply_result_table_name      IN VARCHAR2,
  target_table_name           IN VARCHAR2,
  case_id_column_name         IN VARCHAR2,
  target_column_name          IN VARCHAR2,
  lift_table_name             IN VARCHAR2,
  positive_target_value        IN VARCHAR2,
  score_column_name           IN VARCHAR2 DEFAULT 'PREDICTION',
  score_criterion_column_name IN VARCHAR2 DEFAULT 'PROBABILITY',
  num_quantiles                IN NUMBER DEFAULT 10,
  cost_matrix_table_name       IN VARCHAR2 DEFAULT NULL,
  apply_result_schema_name    IN VARCHAR2 DEFAULT NULL,
  target_schema_name          IN VARCHAR2 DEFAULT NULL,
  cost_matrix_schema_name     IN VARCHAR2 DEFAULT NULL,
  score_criterion_type        IN VARCHAR2 DEFAULT 'PROBABILITY');
```

## Parameters

**Table 44–31 COMPUTE\_LIFT Procedure Parameters**

Parameter	Description
apply_result_table_name	Table containing the predictions.
target_table_name	Table containing the known target values from the test data.
case_id_column_name	Case ID column in the apply results table. Must match the case identifier in the targets table.
target_column_name	Target column in the targets table. Contains the known target values from the test data.
lift_table_name	Table containing the lift statistics. The table will be created by the procedure in the user's schema.  The columns in the lift table are described in the Usage Notes.
positive_target_value	The positive class. This should be the class of interest, for which you want to calculate lift.  If the target column is a NUMBER, you can use the TO_CHAR ( ) operator to provide the value as a string.
score_column_name	Column containing the predictions in the apply results table.  The default column name is 'PREDICTION', which is the default name created by the APPLY procedure (See "APPLY Procedure" on page 44-30).
score_criterion_column_name	Column containing the scoring criterion in the apply results table. Contains either the probabilities or the costs that determine the predictions.  By default, scoring is based on probability; the class with the highest probability is predicted for each case. If scoring is based on cost, the class with the lowest cost is predicted.  The score_criterion_type parameter indicates whether probabilities or costs will be used for scoring.  The default column name is 'PROBABILITY', which is the default name created by the APPLY procedure (See "APPLY Procedure" on page 44-30).  See the Usage Notes for additional information.
num_quantiles	Number of quantiles to be used in calculating lift. The default is 10.
cost_matrix_table_name	(Optional) Table that defines the costs associated with misclassifications. If a cost matrix table is provided and the score_criterion_type parameter is set to 'COST', the costs will be used as the scoring criteria.  The columns in a cost matrix table are described in the Usage Notes.
apply_result_schema_name	Schema of the apply results table.  If null, the user's schema is assumed.
target_schema_name	Schema of the table containing the known targets.  If null, the user's schema is assumed.

**Table 44–31 (Cont.) COMPUTE\_LIFT Procedure Parameters**

Parameter	Description
cost_matrix_schema_name	Schema of the cost matrix table, if one is provided. If null, the user's schema is assumed.
score_criterion_type	Whether to use probabilities or costs as the scoring criterion. Probabilities or costs are passed in the column identified in the score_criterion_column_name parameter.  The default value of score_criterion_type is 'PROBABILITY'. To use costs as the scoring criterion, specify 'COST'.  If score_criterion_type is set to 'COST' but no cost matrix is provided and if there is a scoring cost matrix associated with the model, then the associated costs are used for scoring.  See the Usage Notes and the Examples.

**Usage Notes**

- The predictive information you pass to COMPUTE\_LIFT may be generated using SQL PREDICTION functions, the DBMS\_DATA\_MINING.APPLY procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the lift.
- Instead of passing a cost matrix to COMPUTE\_LIFT, you can use a scoring cost matrix associated with the model. A scoring cost matrix can be embedded in the model or it can be defined dynamically when the model is applied. To use a scoring cost matrix, invoke the SQL PREDICTION\_COST function to populate the score criterion column.
- The predictions that you pass to COMPUTE\_LIFT are in a table or view specified in apply\_results\_table\_name.

```
CREATE TABLE apply_result_table_name AS (
    case_id_column_name          VARCHAR2,
    score_column_name            VARCHAR2,
    score_criterion_column_name  VARCHAR2);
```

- A cost matrix must have the columns described in [Table 44–32](#).

**Table 44–32 Columns in a Cost Matrix**

Column Name	Data Type
actual_target_value	NUMBER or VARCHAR2
predicted_target_value	NUMBER or VARCHAR2
cost	NUMBER

**See Also:** *Oracle Data Mining Concepts* for more information about cost matrixes

- The table created by COMPUTE\_LIFT has the columns described in [Table 44–33](#)

**Table 44-33 Columns in a Lift Table**

Column Name	Data Type
quantile_number	NUMBER
probability_threshold	NUMBER
gain_cumulative	NUMBER
quantile_total_count	NUMBER
quantile_target_count	NUMBER
percent_records_cumulative	NUMBER
lift_cumulative	NUMBER
target_density_cumulative	NUMBER
targets_cumulative	NUMBER
non_targets_cumulative	NUMBER
lift_quantile	NUMBER
target_density	NUMBER

**See Also:** *Oracle Data Mining Concepts* for details about the information in the lift table

- When a cost matrix is passed to `COMPUTE_LIFT`, the cost threshold is returned in the `probability_threshold` column of the lift table.

## Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`, which is created by one of the Oracle Data Mining sample programs.

The example illustrates lift based on probabilities. For examples that show computation based on costs, see "[COMPUTE\\_CONFUSION\\_MATRIX Procedure](#)" on page 44-34.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
  SELECT cust_id, t.prediction, t.probability
  FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using probabilities as the scoring criterion, you can compute lift as follows.

```
BEGIN
  DBMS_DATA_MINING.COMPUTE_LIFT (
    apply_result_table_name => 'nb_apply_results',
    target_table_name       => 'mining_data_test_v',
    case_id_column_name    => 'cust_id',
    target_column_name     => 'affinity_card',
    lift_table_name        => 'nb_lift',
    positive_target_value  => to_char(1),
    score_column_name     => 'PREDICTION',
    score_criterion_column_name => 'PROBABILITY',
    num_quantiles         => 10,
    cost_matrix_table_name => null,
    apply_result_schema_name => null,
    target_schema_name    => null,
```

```
cost_matrix_schema_name => null,  
score_criterion_type    => 'PROBABILITY');  
END;  
/
```

This query displays some of the statistics from the resulting lift table.

```
SQL>SELECT quantile_number, probability_threshold, gain_cumulative,  
           quantile_total_count  
           FROM nb_lift;
```

QUANTILE_NUMBER	PROBABILITY_THRESHOLD	GAIN_CUMULATIVE	QUANTILE_TOTAL_COUNT
1	.989335775	.15034965	55
2	.980534911	.26048951	55
3	.968506098	.374125874	55
4	.958975196	.493006993	55
5	.946705997	.587412587	55
6	.927454174	.66958042	55
7	.904403627	.748251748	55
8	.836482525	.839160839	55
10	.500184953	1	54



## COMPUTE\_ROC Procedure

This procedure computes receiver operating characteristic (ROC), stores the results in a table in the user's schema, and returns a measure of the model accuracy.

ROC is a test metric for binary classification models. To compute ROC, one of the target values must be designated as the positive class. COMPUTE\_ROC compares the predictions generated by the model with the actual target values in a set of test data.

ROC measures the impact of changes in the probability threshold. The probability threshold is the decision point used by the model for predictions. In binary classification, the default probability threshold is 0.5. The value predicted for each case is the one with a probability greater than 50%.

ROC can be plotted as a curve on an X-Y axis. The false positive rate is placed on the X axis. The true positive rate is placed on the Y axis. A false positive is a positive prediction for a case that is negative in the test data. A true positive is a positive prediction for a case that is positive in the test data.

COMPUTE\_ROC accepts two input streams:

- The predictions generated on the test data. The information is passed in three columns:
  - Case ID column
  - Prediction column
  - Scoring criterion column containing probabilities
- The known target values in the test data. The information is passed in two columns:
  - Case ID column
  - Target column containing the known target values

### See Also:

*Oracle Data Mining Concepts* for more details about ROC and test metrics for classification

["COMPUTE\\_CONFUSION\\_MATRIX Procedure"](#) on page 44-34

["COMPUTE\\_LIFT Procedure"](#) on page 44-40

## Syntax

```
DBMS_DATA_MINING.COMPUTE_ROC (
    roc_area_under_curve          OUT NUMBER,
    apply_result_table_name      IN  VARCHAR2,
    target_table_name            IN  VARCHAR2,
    case_id_column_name          IN  VARCHAR2,
    target_column_name           IN  VARCHAR2,
    roc_table_name               IN  VARCHAR2,
    positive_target_value        IN  VARCHAR2,
    score_column_name            IN  VARCHAR2 DEFAULT 'PREDICTION',
    score_criterion_column_name  IN  VARCHAR2 DEFAULT 'PROBABILITY',
    apply_result_schema_name     IN  VARCHAR2 DEFAULT NULL,
    target_schema_name           IN  VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 44–34 COMPUTE\_ROC Procedure Parameters**

Parameter	Description
<code>roc_area_under_the_curve</code>	Output parameter containing the area under the ROC curve (AUC). The AUC measures the likelihood that an actual positive will be predicted as positive.  The greater the AUC, the greater the flexibility of the model in accommodating trade-offs between positive and negative class predictions. AUC can be especially important when one target class is rarer or more important to identify than another.
<code>apply_result_table_name</code>	Table containing the predictions.
<code>target_table_name</code>	Table containing the known target values from the test data.
<code>case_id_column_name</code>	Case ID column in the apply results table. Must match the case identifier in the targets table.
<code>target_column_name</code>	Target column in the targets table. Contains the known target values from the test data.
<code>roc_table_name</code>	Table containing the ROC output. The table will be created by the procedure in the user's schema.  The columns in the ROC table are described in the Usage Notes.
<code>positive_target_value</code>	The positive class. This should be the class of interest, for which you want to calculate ROC.  If the target column is a NUMBER, you can use the <code>TO_CHAR()</code> operator to provide the value as a string.
<code>score_column_name</code>	Column containing the predictions in the apply results table.  The default column name is 'PREDICTION', which is the default name created by the <code>APPLY</code> procedure (See " <a href="#">APPLY Procedure</a> " on page 44-30).
<code>score_criterion_column_name</code>	Column containing the scoring criterion in the apply results table. Contains the probabilities that determine the predictions.  The default column name is 'PROBABILITY', which is the default name created by the <code>APPLY</code> procedure (See " <a href="#">APPLY Procedure</a> " on page 44-30).
<code>apply_result_schema_name</code>	Schema of the apply results table.  If null, the user's schema is assumed.
<code>target_schema_name</code>	Schema of the table containing the known targets.  If null, the user's schema is assumed.

## Usage Notes

- The predictive information you pass to `COMPUTE_ROC` may be generated using SQL PREDICTION functions, the `DBMS_DATA_MINING.APPLY` procedure, or some other mechanism. As long as you pass the appropriate data, the procedure can compute the receiver operating characteristic.
- The predictions that you pass to `COMPUTE_ROC` are in a table or view specified in `apply_results_table_name`.

```
CREATE TABLE apply_result_table_name AS (
    case_id_column_name          VARCHAR2,
    score_column_name            VARCHAR2,
    score_criterion_column_name  VARCHAR2);
```

- The table created by COMPUTE\_ROC has the columns shown in [Table 44–35](#).

**Table 44–35** COMPUTE\_ROC Output

Column	Data Type
probability	NUMBER
true_positives	NUMBER
false_negatives	NUMBER
false_positives	NUMBER
true_negatives	NUMBER
true_positive_fraction	NUMBER
false_positive_fraction	NUMBER

**See Also:** *Oracle Data Mining Concepts* for details about the output of COMPUTE\_ROC

- ROC is typically used to determine the most desirable probability threshold. This can be done by examining the true positive fraction and the false positive fraction. The true positive fraction is the percentage of all positive cases in the test data that were correctly predicted as positive. The false positive fraction is the percentage of all negative cases in the test data that were incorrectly predicted as positive.

Given a probability threshold, the following statement returns the positive predictions in an apply result table ordered by probability.

```
SELECT case_id_column_name
       FROM apply_result_table_name
       WHERE probability > probability_threshold
       ORDER BY probability DESC;
```

- There are two approaches to identifying the most desirable probability threshold. Which approach you use depends on whether or not you know the relative cost of positive versus negative class prediction errors.

If the costs are known, you can apply the relative costs to the ROC table to compute the minimum cost probability threshold. Suppose the relative cost ratio is: Positive Class Error Cost / Negative Class Error Cost = 20. Then execute a query like this.

```
WITH cost AS (
    SELECT probability_threshold, 20 * false_negatives + false_positives cost
    FROM ROC_table
    GROUP BY probability_threshold),
minCost AS (
    SELECT min(cost) minCost
    FROM cost)
SELECT max(probability_threshold) probability_threshold
FROM cost, minCost
WHERE cost = minCost;
```

If relative costs are not well known, you can simply scan the values in the ROC table (in sorted order) and make a determination about which of the displayed trade-offs (misclassified positives versus misclassified negatives) is most desirable.

```
SELECT * FROM ROC_table
       ORDER BY probability_threshold;
```

## Examples

This example uses the Naive Bayes model `nb_sh_clas_sample`, which is created by one of the Oracle Data Mining sample programs.

The following statement applies the model to the test data and stores the predictions and probabilities in a table.

```
CREATE TABLE nb_apply_results AS
  SELECT cust_id, t.prediction, t.probability
  FROM mining_data_test_v, TABLE(PREDICTION_SET(nb_sh_clas_sample USING *)) t;
```

Using the predictions and the target values from the test data, you can compute ROC as follows.

```
DECLARE
  v_area_under_curve NUMBER;
BEGIN
  DBMS_DATA_MINING.COMPUTE_ROC (
    roc_area_under_curve      => v_area_under_curve,
    apply_result_table_name   => 'nb_apply_results',
    target_table_name         => 'mining_data_test_v',
    case_id_column_name       => 'cust_id',
    target_column_name        => 'affinity_card',
    roc_table_name            => 'nb_roc',
    positive_target_value     => '1',
    score_column_name         => 'PREDICTION',
    score_criterion_column_name => 'PROBABILITY');
  DBMS_OUTPUT.PUT_LINE('**** AREA UNDER ROC CURVE ****: ' ||
    ROUND(v_area_under_curve,4));
END;
```

The resulting AUC and a selection of columns from the ROC table are shown as follows.

```
**** AREA UNDER ROC CURVE ****: .8212
```

```
SQL> SELECT probability, true_positive_fraction, false_positive_fraction
       FROM nb_roc;
```

PROBABILITY	TRUE_POSITIVE_FRACTION	FALSE_POSITIVE_FRACTION
.00000	1	1
.50018	.826589595	.227902946
.53851	.823699422	.221837088
.54991	.820809249	.217504333
.55628	.815028902	.215771231
.55628	.817919075	.215771231
.57563	.800578035	.214904679
.57563	.812138728	.214904679
.	.	.
.	.	.
.	.	.



## CREATE\_MODEL Procedure

This procedure creates a mining model with a given mining function.

By passing an `xform_list` to `CREATE_MODEL`, you can specify a list of transformations to be performed on the input data. If the `PREP_AUTO` setting is on, the transformations are used in addition to the automatic transformations. If the `PREP_AUTO` setting is off, the specified transformations are the only ones implemented by the model. In both cases, the transformation definitions are embedded in the model and executed automatically whenever the model is applied. See "[Automatic Data Preparation](#)" on page 44-11.

### Syntax

```
DBMS_DATA_MINING.CREATE_MODEL (
    model_name           IN VARCHAR2,
    mining_function      IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    case_id_column_name  IN VARCHAR2,
    target_column_name   IN VARCHAR2 DEFAULT NULL,
    settings_table_name  IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL,
    settings_schema_name IN VARCHAR2 DEFAULT NULL,
    xform_list           IN TRANSFORM_LIST DEFAULT NULL);
```

### Parameters

**Table 44–36 CREATE\_MODEL Procedure Parameters**

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is used. See also " <a href="#">Mining Model Naming Restrictions</a> " on page 44-5.
<code>mining_function</code>	The mining function. Values are listed in <a href="#">Table 44–2, "Mining Functions"</a> .
<code>data_table_name</code>	Table or view containing the build data.
<code>case_id_column_name</code>	Case identifier column in the build data.
<code>target_column_name</code>	For supervised models, the target column in the build data. <code>NULL</code> for unsupervised models.
<code>settings_table_name</code>	Table containing build settings for the model. <code>NULL</code> if there is no settings table (only default settings are used).
<code>data_schema_name</code>	Schema hosting the build data. If <code>NULL</code> , the user's schema is assumed.
<code>settings_schema_name</code>	Schema hosting the settings table. If <code>NULL</code> , the user's schema is assumed.

**Table 44–36 (Cont.) CREATE\_MODEL Procedure Parameters**

Parameter	Description
xform_list	<p>A list of transformations to be used in addition to or instead of automatic transformations, depending on the value of the PREP_AUTO setting. (See "<a href="#">Automatic Data Preparation</a>" on page 44-11.)</p> <p>The data type is TRANSFORM_LIST, which consists of records of type TRANSFORM_REC.</p> <pre> TYPE   TRANSFORM_REC    IS RECORD (     attribute_name  VARCHAR2(4000),     attribute_subname VARCHAR2(4000),     expression      EXPRESSION_REC,     reverse_expression EXPRESSION_REC,     attribute_spec   VARCHAR2(4000)); </pre> <p>Each TRANSFORM_REC describes the transformation and reverse transformation for an attribute. EXPRESSION_REC stores a SQL expression.</p> <p>The SQL expression stored in EXPRESSION_REC can be manipulated using routines in the DBMS_DATA_MINING_TRANSFORM package. Examples are: SET_EXPRESSION, GET_EXPRESSION, and SET_TRANSFORM.</p> <p>You can use ATTRIBUTE_SPEC to disable Automatic Data Preparation (ADP) for an individual attribute. Specify 'NOPREP' to prevent automatic preparation of this attribute when ADP is on. When ADP is off, the 'NOPREP' value is ignored.</p> <p>The 'NOPREP' value cannot be used for an individual subname of a nested attribute. If 'NOPREP' is specified for an individual subname when ADP is on, an error is generated.</p>

## Usage Notes

You can obtain information about a model by querying these data dictionary views.

```

ALL_MINING_MODELS
ALL_MINING_MODEL_ATTRIBUTES
ALL_MINING_MODEL_SETTINGS

```

Specify the USER prefix instead of ALL to obtain information about models in your own schema only.

**See Also:** *Oracle Data Mining Application Developer's Guide* for information on the data dictionary views

## Examples

The first example builds a classification model using the Support Vector Machine algorithm.

```

-- Create the settings table
CREATE TABLE svm_model_settings (
  setting_name VARCHAR2(30),
  setting_value VARCHAR2(30));

-- Populate the settings table
-- Specify SVM. By default, Naive Bayes is used for classification.
-- Specify ADP. By default, ADP is not used.

```

```

BEGIN
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.algo_name, dbms_data_mining.algo_support_vector_machines);
  INSERT INTO svm_model_settings (setting_name, setting_value) VALUES
    (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_on);
  COMMIT;
END;
/
-- Create the model using the specified settings
BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'svm_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name    => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name => 'affinity_card',
    settings_table_name => 'svm_model_settings');
END;
/

```

You can display the model settings with the following query.

```

SELECT * FROM user_mining_model_settings
       WHERE model_name IN 'SVM_MODEL';

```

MODEL_NAME	SETTING_NAME	SETTING_VALUE	SETTING
SVM_MODEL	ALGO_NAME	ALGO_SUPPORT_VECTOR_MACHINES	INPUT
SVM_MODEL	SVMS_KERNEL_CACHE_SIZE	50000000	DEFAULT
SVM_MODEL	SVMS_ACTIVE_LEARNING	SVMS_AL_ENABLE	DEFAULT
SVM_MODEL	SVMS_STD_DEV	3.004524	DEFAULT
SVM_MODEL	PREP_AUTO	ON	INPUT
SVM_MODEL	SVMS_COMPLEXITY_FACTOR	1.887389	DEFAULT
SVM_MODEL	SVMS_KERNEL_FUNCTION	SVMS_GAUSSIAN	DEFAULT
SVM_MODEL	SVMS_CONV_TOLERANCE	.001	DEFAULT

The second example creates an anomaly detection model. Anomaly detection uses SVM classification without a target. This example uses the same settings table created for the SVM classification model in the first example.

```

BEGIN
  DBMS_DATA_MINING.CREATE_MODEL(
    model_name          => 'anomaly_detect_model',
    mining_function     => dbms_data_mining.classification,
    data_table_name    => 'mining_data_build_v',
    case_id_column_name => 'cust_id',
    target_column_name => null,
    settings_table_name => 'svm_model_settings');
END;
/

```

This query shows that the models created in these examples are the only ones in your schema.

```

SELECT model_name, mining_function, algorithm FROM user_mining_models;

```

MODEL_NAME	MINING_FUNCTION	ALGORITHM
SVM_MODEL	CLASSIFICATION	SUPPORT_VECTOR_MACHINES
ANOMALY_DETECT_MODEL	CLASSIFICATION	SUPPORT_VECTOR_MACHINES



This query shows that only the SVM classification model has a target.

```
SELECT model_name, attribute_name, attribute_type, target
       FROM user_mining_model_attributes
       WHERE target = 'YES';
```

MODEL_NAME	ATTRIBUTE_NAME	ATTRIBUTE_TYPE	TARGET
-----	-----	-----	-----
SVM_MODEL	AFFINITY_CARD	CATEGORICAL	YES

## DROP\_MODEL Procedure

This procedure deletes the specified mining model.

### Syntax

```
DBMS_DATA_MINING.DROP_MODEL (model_name IN VARCHAR2,  
                             force       IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 44–37** *DROP\_MODEL Procedure Parameters*

Parameter	Description
model_name	Name of the mining model in the form [ <i>schema_name</i> .] <i>model_name</i> . If you do not specify a schema, your own schema is used.
force	Forces the mining model to be dropped even if it is invalid. A mining model may be invalid if a serious system error interrupted the model build process.

### Usage Note

To drop a mining model, you must be the owner or you must have the `DROP ANY MINING MODEL` privilege. See *Oracle Data Mining Administrator's Guide* for information about privileges for data mining.

### Example

You can use the following command to delete a valid mining model named `nb_sh_clas_sample` that exists in your schema.

```
BEGIN  
  DBMS_DATA_MINING.DROP_MODEL(model_name => 'nb_sh_clas_sample');  
END;  
/
```

## EXPORT\_MODEL Procedure

This procedure exports the specified data mining models to a dump file set. To import the models from the dump file set, use the [IMPORT\\_MODEL Procedure](#).

EXPORT\_MODEL and IMPORT\_MODEL use Oracle Data Pump technology.

When Oracle Data Pump is used to export/import an entire schema or database, the mining models in the schema or database are included. However, EXPORT\_MODEL and IMPORT\_MODEL are the only utilities that support the export/import of individual models.

### See Also:

*Oracle Database Utilities* for information about Oracle Data Pump

*Oracle Data Mining Administrator's Guide* for more information about exporting and importing mining models

## Syntax

```
DBMS_DATA_MINING.EXPORT_MODEL (
  filename          IN VARCHAR2,
  directory         IN VARCHAR2,
  model_filter      IN VARCHAR2 DEFAULT NULL,
  filesize          IN VARCHAR2 DEFAULT NULL,
  operation         IN VARCHAR2 DEFAULT NULL,
  remote_link       IN VARCHAR2 DEFAULT NULL,
  jobname           IN VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 44–38 EXPORT\_MODEL Procedure Parameters**

Parameter	Description
filename	<p>Name of the dump file set to which the models should be exported. The name must be unique within the schema.</p> <p>The dump file set can contain one or more files. The number of files in a dump file set is determined by the size of the models being exported (both metadata and data) and a specified or estimated maximum file size. You can specify the file size in the <code>filesize</code> parameter, or you can use the <code>operation</code> parameter to cause Oracle Data Pump to estimate the file size. If the size of the models to export is greater than the maximum file size, one or more additional files are created.</p> <p>When the export operation completes successfully, the name of the dump file set is automatically expanded to <code>filename01.dmp</code>, even if there is only one file in the dump set. If there are additional files, they are named sequentially as <code>filename02.dmp</code>, <code>filename03.dmp</code>, and so forth.</p>
directory	<p>Name of a pre-defined directory object that specifies where the dump file set should be created.</p> <p>The exporting user must have read/write privileges on the directory object and on the file system directory that it identifies.</p> <p>See <i>Oracle Database SQL Language Reference</i> for information about directory objects.</p>

**Table 44–38 (Cont.) EXPORT\_MODEL Procedure Parameters**

Parameter	Description
<code>model_filter</code>	<p>Optional parameter that specifies which model or models to export. If you do not specify a value for <code>model_filter</code>, all models in the schema are exported. You can also specify NULL (the default) or 'ALL' to export all models.</p> <p>You can export individual models by name and groups of models based on mining function or algorithm. For instance, you could export all regression models or all Naive Bayes models. Examples are provided in <a href="#">Table 44–39</a>.</p>
<code>filesize</code>	<p>Optional parameter that specifies the maximum size of a file in the dump file set. The size may be specified in bytes, kilobytes (K), megabytes (M), or gigabytes (G). The default size is 50 MB.</p> <p>If the size of the models to export is larger than <code>filesize</code>, one or more additional files are created within the dump set. See the description of the <code>filename</code> parameter for more information.</p>
<code>operation</code>	<p>Optional parameter that specifies whether or not to estimate the size of the files in the dump set. By default the size is not estimated and the value of the <code>filesize</code> parameter determines the size of the files.</p> <p>You can specify either of the following values for <code>operation</code>:</p> <ul style="list-style-type: none"> <li>■ 'EXPORT' — Export all or the specified models. (Default)</li> <li>■ 'ESTIMATE' — Estimate the size of the exporting models.</li> </ul>
<code>remote_link</code>	Optional parameter not used in this release. Set to NULL.
<code>jobname</code>	<p>Optional parameter that specifies the name of the export job. By default, the name has the form <code>username_exp_nnnn</code>, where <code>nnnn</code> is a number. For example, a job name in the SCOTT schema might be <code>SCOTT_exp_134</code>.</p> <p>If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters.</p> <p>A log file for the export job, named <code>jobname.log</code>, is created in the same directory as the dump file set.</p>

## Usage Notes

The `model_filter` parameter specifies which models to export. You can list the models by name, or you can specify all models that have the same mining function or algorithm. You can query the `USER_MINING_MODELS` view to list the models in your schema.

```
SQL> describe user_mining_models
```

Name	Null?	Type
MODEL_NAME	NOT NULL	VARCHAR2(30)
MINING_FUNCTION		VARCHAR2(30)
ALGORITHM		VARCHAR2(30)
CREATION_DATE	NOT NULL	DATE
BUILD_DURATION		NUMBER
MODEL_SIZE		NUMBER
COMMENTS		VARCHAR2(4000)

For more information on data dictionary views of mining models, see "[Mining Model Objects](#)" on page 44-5.

Examples of model filters are provided in [Table 44–39](#).

**Table 44–39 Sample Values for the Model Filter Parameter**

Sample Value	Meaning
'mymodel'	Export the model named mymodel
'name= ''mymodel'''	Export the model named mymodel
'name IN (''mymodel2'', ''mymodel3'')'	Export the models named mymodel2 and mymodel3
'ALGORITHM_NAME = ''NAIVE_BAYES'''	Export all Naive Bayes models. See <a href="#">Table 44–4</a> for a list of algorithm names.
'FUNCTION_NAME = ''CLASSIFICATION'''	Export all classification models. See <a href="#">Table 44–2</a> for a list of mining functions.

## Examples

The following statement exports all the models in the DMUSER3 schema to a dump file set called models\_out in the directory \$ORACLE\_HOME/rdbms/log. This directory is mapped to a directory object called DATA\_PUMP\_DIR. The DMUSER3 user has read/write access to the directory and to the directory object.

```
SQL>execute dbms_data_mining.export_model ('models_out', 'DATA_PUMP_DIR');
```

You can exit SQL\*Plus and list the resulting dump file and log file.

```
SQL>exit
>cd $ORACLE_HOME/rdbms/log
>ls
>DMUSER3_exp_1027.log  models_out01.dmp
```

The following example uses the same directory object and is executed by the same user. It exports the models called NMF\_SH\_SAMPLE and SVMR\_SH\_REGR\_SAMPLE to a different dump file set in the same directory.

```
SQL>execute dbms_data_mining.export_model ( 'models2_out', 'DATA_PUMP_DIR',
      'name in (''NMF_SH_SAMPLE'', ''SVMR_SH_REGR_SAMPLE'')');
SQL>exit
>cd $ORACLE_HOME/rdbms/log
>ls
>DMUSER3_exp_1027.log  models_out01.dmp
  DMUSER3_exp_924.log  models2_out01.dmp
```

## GET\_ASSOCIATION\_RULES Function

This table function returns the rules from an association model.

You can specify filtering criteria to cause GET\_ASSOCIATION\_RULES to return a subset of the rules. Filtering criteria can improve the performance of the table function. If the number of rules is large, the greatest performance improvement will result from specifying the `topn` parameter.

### Syntax

```
DBMS_DATA_MINING.GET_ASSOCIATION_RULES (
  model_name          IN VARCHAR2,
  topn                IN NUMBER DEFAULT NULL,
  rule_id             IN INTEGER DEFAULT NULL,
  min_confidence      IN NUMBER DEFAULT NULL,
  min_support         IN NUMBER DEFAULT NULL,
  max_rule_length     IN INTEGER DEFAULT NULL,
  min_rule_length     IN INTEGER DEFAULT NULL,
  sort_order          IN ORA_MINING_VARCHAR2_NT DEFAULT NULL,
  antecedent_items    IN DM_ITEMS DEFAULT NULL,
  consequent_items    IN DM_ITEMS DEFAULT NULL,
  min_lift            IN NUMBER DEFAULT NULL)
RETURN DM_RULES PIPELINED;
```

### Parameters

**Table 44–40** GET\_ASSOCIATION\_RULES Function Parameters

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is used.  This is the only required parameter of GET_ASSOCIATION_RULES. All other parameters specify optional filters on the rules to return.
<code>topn</code>	Return the <i>n</i> top rules ordered by confidence and then support, both descending. If you specify a sort order, the top <i>n</i> rules are derived after the sort is performed.  If <code>topn</code> is specified and no maximum or minimum rule length is specified, then the only columns allowed in the sort order are <code>RULE_CONFIDENCE</code> and <code>RULE_SUPPORT</code> . If <code>topn</code> is specified and a maximum or minimum rule length is specified, then <code>RULE_CONFIDENCE</code> , <code>RULE_SUPPORT</code> , and <code>NUMBER_OF_ITEMS</code> are allowed in the sort order.
<code>rule_id</code>	Identifier of the rule to return. If you specify a value for <code>rule_id</code> , do not specify values for the other filtering parameters.
<code>min_confidence</code>	Return the rules with confidence greater than or equal to this number
<code>min_support</code>	Return the rules with support greater than or equal to this number
<code>max_rule_length</code>	Return the rules with a length less than or equal to this number.  Rule length refers to the number of items in the rule (See <code>NUMBER_OF_ITEMS</code> in Table 44–41). For example, in the rule <code>A=&gt;B</code> (if A, then B), the number of items is 2.  If <code>max_rule_length</code> is specified, then the <code>NUMBER_OF_ITEMS</code> column is permitted in the sort order.

**Table 44–40 (Cont.) GET\_ASSOCIATION\_RULES Function Parameters**

Parameter	Description
min_rule_length	Return the rules with a length greater than or equal to this number. See max_rule_length for a description of rule length.  If min_rule_length is specified, then the NUMBER_OF_ITEMS column is permitted in the sort order.
sort_order	Sort the rules by the values in one or more of the returned columns. Specify one or more column names, each followed by ASC for ascending order or DESC for descending order.  For example, to sort the result set in descending order first by the NUMBER_OF_ITEMS column, then by the RULE_CONFIDENCE column, you would specify:  ORA_MINING_VARCHAR2_NT ( 'NUMBER_OF_ITEMS DESC' , 'RULE_CONFIDENCE DESC' )  If you specify topn, the results will vary depending on the sort order.  By default, the results are sorted by confidence in descending order, then by support in descending order.  See the examples.
antecedent_items	Return the rules with these items in the antecedent. See the examples.
consequent_items	Return the rules with this item in the consequent. See the examples.
min_lift	Return the rules with lift greater than or equal to this number.

## Return Values

**Table 44–41 GET\_ASSOCIATION\_RULES Function Return Values**

Return Value	Description																																
DM_RULES	Represents a set of rows of type DM_RULE. The rows have the following columns:  <table border="0"> <tr> <td>(rule_id</td> <td>INTEGER,</td> </tr> <tr> <td>antecedent</td> <td>DM_PREDICATES,</td> </tr> <tr> <td>consequent</td> <td>DM_PREDICATES,</td> </tr> <tr> <td>rule_support</td> <td>NUMBER,</td> </tr> <tr> <td>rule_confidence</td> <td>NUMBER,</td> </tr> <tr> <td>rule_lift</td> <td>NUMBER,</td> </tr> <tr> <td>antecedent_support</td> <td>NUMBER,</td> </tr> <tr> <td>consequent_support</td> <td>NUMBER,</td> </tr> <tr> <td>number_of_items</td> <td>INTEGER )</td> </tr> </table> <p>The antecedent and consequent columns each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:</p> <table border="0"> <tr> <td>(attribute_name</td> <td>VARCHAR2 (4000) ,</td> </tr> <tr> <td>attribute_subname</td> <td>VARCHAR2 (4000) ,</td> </tr> <tr> <td>conditional_operator</td> <td>CHAR (2) / * = , &lt; &gt; , &lt; , &gt; , &lt; = , &gt; = * / ,</td> </tr> <tr> <td>attribute_num_value</td> <td>NUMBER,</td> </tr> <tr> <td>attribute_str_value</td> <td>VARCHAR2 (4000) ,</td> </tr> <tr> <td>attribute_support</td> <td>NUMBER,</td> </tr> <tr> <td>attribute_confidence</td> <td>NUMBER)</td> </tr> </table>	(rule_id	INTEGER,	antecedent	DM_PREDICATES,	consequent	DM_PREDICATES,	rule_support	NUMBER,	rule_confidence	NUMBER,	rule_lift	NUMBER,	antecedent_support	NUMBER,	consequent_support	NUMBER,	number_of_items	INTEGER )	(attribute_name	VARCHAR2 (4000) ,	attribute_subname	VARCHAR2 (4000) ,	conditional_operator	CHAR (2) / * = , < > , < , > , < = , > = * / ,	attribute_num_value	NUMBER,	attribute_str_value	VARCHAR2 (4000) ,	attribute_support	NUMBER,	attribute_confidence	NUMBER)
(rule_id	INTEGER,																																
antecedent	DM_PREDICATES,																																
consequent	DM_PREDICATES,																																
rule_support	NUMBER,																																
rule_confidence	NUMBER,																																
rule_lift	NUMBER,																																
antecedent_support	NUMBER,																																
consequent_support	NUMBER,																																
number_of_items	INTEGER )																																
(attribute_name	VARCHAR2 (4000) ,																																
attribute_subname	VARCHAR2 (4000) ,																																
conditional_operator	CHAR (2) / * = , < > , < , > , < = , > = * / ,																																
attribute_num_value	NUMBER,																																
attribute_str_value	VARCHAR2 (4000) ,																																
attribute_support	NUMBER,																																
attribute_confidence	NUMBER)																																

## Usage Notes

This table function pipes out rows of type `DM_RULES`. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

The `ORA_MINING_VARCHAR2_NT` type is defined as a table of `VARCHAR2(4000)`.

## Examples

The following example demonstrates an Association model build followed by several invocations of the `GET_ASSOCIATION_RULES` table function.

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
 WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE census_settings
   SET setting_value = TO_CHAR(0.081)
 WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

-- build an AR model
DBMS_DATA_MINING.CREATE_MODEL(
  model_name => 'market_model',
  function => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name => NULL,
  settings_table_name => 'census_settings');
END;
/
-- View the (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model'));
```

In the previous example, you view all rules. To view just the top 20 rules, use the following statement.

```
-- View the top 20 (unformatted) rules
SELECT rule_id, antecedent, consequent, rule_support,
       rule_confidence
  FROM TABLE(DBMS_DATA_MINING.GET_ASSOCIATION_RULES('market_model', 20));
```

The following query uses the association model `AR_SH_SAMPLE`, which is created from one of the Oracle Data Mining sample programs. (See *Oracle Data Mining Administrator's Guide* for information about the sample programs.)

```
SELECT * FROM TABLE (
  DBMS_DATA_MINING.GET_ASSOCIATION_RULES (
    'AR_SH_SAMPLE', 10, NULL, 0.5, 0.01, 2, 1,
    ORA_MINING_VARCHAR2_NT (
      'NUMBER_OF_ITEMS DESC', 'RULE_CONFIDENCE DESC', 'RULE_SUPPORT DESC'),
    DM_ITEMS(DM_ITEM('CUSTPRODS', 'Mouse Pad', 1, NULL),
              DM_ITEM('CUSTPRODS', 'Standard Mouse', 1, NULL)),
    DM_ITEMS(DM_ITEM('CUSTPRODS', 'Extension Cable', 1, NULL))));
```

The query returns three rules, shown as follows.

```
13  DM_PREDICATES(
```



```
DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL),
DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
DM_PREDICATES(
  DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
.15532      .84393  2.7075   .18404   .3117   2

11 DM_PREDICATES(
  DM_PREDICATE('CUSTPRODS', 'Standard Mouse', '= ', 1, NULL, NULL, NULL))
DM_PREDICATES(
  DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
.18085      .56291  1.8059   .32128   .3117   1

9  DM_PREDICATES(
  DM_PREDICATE('CUSTPRODS', 'Mouse Pad', '= ', 1, NULL, NULL, NULL))
DM_PREDICATES(
  DM_PREDICATE('CUSTPRODS', 'Extension Cable', '= ', 1, NULL, NULL, NULL))
.17766      .55116  1.7682   .32234   .3117   1
```

## GET\_DEFAULT\_SETTINGS Function

The GET\_DEFAULT\_SETTINGS function was deprecated in Oracle Data Mining 11g Release 1 (11.1). It was replaced with the data dictionary view \*\_MINING\_MODEL\_SETTINGS. USER\_, ALL\_, and DBA\_ versions of the view are available. See *Oracle Data Mining Application Developer's Guide*.

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---



---

**Note:** See ["Deprecated Subprograms"](#) on page 44-8

---



---

This table function returns the default settings for all mining functions and algorithms supported in the DBMS\_DATA\_MINING package.

### Syntax

```
DBMS_DATA_MINING.GET_DEFAULT_SETTINGS
  RETURN DM_MODEL_SETTINGS PIPELINED;
```

### Return Values

**Table 44-42** GET\_DEFAULT\_SETTINGS Function Return Values

Return Value	Description
DM_MODEL_SETTINGS	Represents a set of rows of type DM_MODEL_SETTING. The rows have the following columns:  (setting_name     VARCHAR2(30), setting_value    VARCHAR2(128))

### Usage Notes

This table function pipes out rows of type DM\_MODEL\_SETTING. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

This function is particularly useful if you do not know what settings are associated with a particular function or algorithm, and you want to override some or all of them.

### Examples

For example, if you want to override some or all of *k*-Means clustering settings, you can create a settings table as shown, and update individual settings as required.

```
BEGIN
  CREATE TABLE mysettings AS
  SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
  WHERE setting_name LIKE 'KMNS%';
  -- now update individual settings as required
  UPDATE mysettings
     SET setting_value = 0.02
     WHERE setting_name = DBMS_DATA_MINING.KMNS_MIN_PCT_ATTR_SUPPORT;
END;
/
```

## GET\_FREQUENT\_ITEMSETS Function

This table function returns a set of rows that represent the frequent itemsets from an Association model. For a detailed description of frequent itemsets, consult *Oracle Data Mining Concepts*.

### Syntax

```
DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS (
  model_name          IN VARCHAR2,
  topn                IN NUMBER DEFAULT NULL,
  max_itemset_length IN NUMBER DEFAULT NULL)
RETURN DM_ITEMSETS PIPELINED;
```

### Parameters

**Table 44–43** GET\_FREQUENT\_ITEMSETS Function Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.
topn	When not NULL, return the top <i>n</i> rows ordered by support in descending order
max_itemset_length	Maximum length of an item set.

### Return Values

**Table 44–44** GET\_FREQUENT\_ITEMSETS Function Return Values

Return Value	Description
DM_ITEMSETS	Represents a set of rows of type DM_ITEMSET. The rows have the following columns: (itemsets_id       NUMBER, items             DM_ITEMS, support           NUMBER, number_of_items   NUMBER) The items column returns a nested table of type DM_ITEMS. The rows have type DM_ITEM: (attribute_name    VARCHAR2(4000), attribute_subname  VARCHAR2(4000), attribute_num_value NUMBER, attribute_str_value VARCHAR2(4000))

### Usage Notes

This table function pipes out rows of type DM\_ITEMSETS. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

### Examples

The following example demonstrates an Association model build followed by an invocation of GET\_FREQUENT\_ITEMSETS table function from Oracle SQL.

```
-- prepare a settings table to override default settings
CREATE TABLE market_settings AS
```

```
        SELECT *
        FROM TABLE(DBMS_DATA_MINING.GET_DEFAULT_SETTINGS)
        WHERE setting_name LIKE 'ASSO_%';
BEGIN
-- update the value of the minimum confidence
UPDATE market_settings
   SET setting_value = TO_CHAR(0.081)
   WHERE setting_name = DBMS_DATA_MINING.asso_min_confidence;

/* build a AR model */
DBMS_DATA_MINING.CREATE_MODEL(
  model_name          => 'market_model',
  function            => DBMS_DATA_MINING.ASSOCIATION,
  data_table_name     => 'market_build',
  case_id_column_name => 'item_id',
  target_column_name  => NULL,
  settings_table_name => 'census_settings');
END;
/

-- View the (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
   FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model'));
```

In the example above, you view all itemsets. To view just the top 20 itemsets, use the following statement:

```
-- View the top 20 (unformatted) Itemsets from SQL*Plus
SELECT itemset_id, items, support, number_of_items
   FROM TABLE(DBMS_DATA_MINING.GET_FREQUENT_ITEMSETS('market_model', 20));
```

## GET\_MODEL\_COST\_MATRIX Function

This function returns the rows of the default scoring matrix associated with the specified model.

By default, this function returns the scoring matrix that was added to the model with the `ADD_COST_MATRIX` procedure. If you wish to obtain the cost matrix used to create a model, specify `cost_matrix_type_create` as the `matrix_type`. See [Table 44-45](#).

See also [ADD\\_COST\\_MATRIX Procedure](#).

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_COST_MATRIX (
    model_name          IN VARCHAR2,
    matrix_type         IN VARCHAR2 DEFAULT cost_matrix_type_score)
RETURN DM_COST_MATRIX PIPELINED;
```

### Parameters

**Table 44-45** GET\_MODEL\_COST\_MATRIX Function Parameters

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is used.
<code>matrix_type</code>	The type of cost matrix.  <code>COST_MATRIX_TYPE_SCORE</code> — cost matrix used for scoring. (Default.)  <code>COST_MATRIX_TYPE_CREATE</code> — cost matrix used to create the model (Decision Tree only).

### Return Values

**Table 44-46** GET\_MODEL\_COST\_MATRIX Function Return Values

Return Value	Description						
<code>DM_COST_MATRIX</code>	Represents a set of rows of type <code>DM_COST_ELEMENT</code> . The rows have the following columns: <table border="1" data-bbox="760 1415 1143 1505"> <tbody> <tr> <td><code>actual</code></td> <td><code>VARCHAR2(4000)</code>,</td> </tr> <tr> <td><code>predicted</code></td> <td><code>VARCHAR2(4000)</code>,</td> </tr> <tr> <td><code>cost</code></td> <td><code>NUMBER</code></td> </tr> </tbody> </table>	<code>actual</code>	<code>VARCHAR2(4000)</code> ,	<code>predicted</code>	<code>VARCHAR2(4000)</code> ,	<code>cost</code>	<code>NUMBER</code>
<code>actual</code>	<code>VARCHAR2(4000)</code> ,						
<code>predicted</code>	<code>VARCHAR2(4000)</code> ,						
<code>cost</code>	<code>NUMBER</code>						

### Usage Notes

Only Decision Tree models can be built with a cost matrix. If you want to build a Decision Tree model with a cost matrix, specify the cost matrix table name in the `CLAS_COST_TABLE_NAME` setting in the settings table for the model. See [Table 44-7](#), "Mining Function Settings".

The cost matrix used to create a Decision Tree model becomes the default scoring matrix for the model. If you want to specify different costs for scoring, you can modify the values in the cost matrix table or you can use the `REMOVE_COST_MATRIX` procedure to remove the cost matrix and the `ADD_COST_MATRIX` procedure to add a new one

**Example**

This example returns the scoring cost matrix associated with the Naive Bayes model NB\_SH\_CLAS\_SAMPLE.

```
column actual format a10
column predicted format a10
SELECT *
   FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
   ORDER BY predicted, actual;
```

ACTUAL	PREDICTED	COST
-----	-----	-----
0	0	.00
1	0	.75
0	1	.25
1	1	.00

## GET\_MODEL\_DETAILS\_ABN Function

The Adaptive Bayes Network algorithm ABN algorithm was deprecated in Oracle Data Mining 11g Release 1 (11.1).

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---



---

**Note:** See ["Deprecated Subprograms"](#) on page 44-8

---



---

This table function returns a set of rows that provide the details of an Adaptive Bayes Network model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_ABN (
    model_name          IN VARCHAR2)
RETURN DM_ABN_DETAILS PIPELINED;
```

### Parameters

**Table 44-47** GET\_MODEL\_DETAILS\_ABN Function Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.

### Return Values

**Table 44-48** GET\_MODEL\_DETAILS\_ABN Function Return Values

Return Value	Description
DM_ABN_DETAILS	<p>Represents a set of rows of type DM_ABN_DETAIL. The rows have the following columns:</p> <pre>(rule_id          INTEGER,  antecedent       DM_PREDICATES,  consequent       DM_PREDICATES,  rule_support     NUMBER)</pre> <p>The antecedent and consequent columns of DM_ABN_DETAIL each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:</p> <pre>(attribute_name   VARCHAR2(4000),  attribute_subname VARCHAR2(4000),  conditional_operator CHAR(2), /*=, &lt;&gt;, &lt;, &gt;, &lt;=, &gt;= */  attribute_num_value NUMBER,  attribute_str_value VARCHAR2(4000),  attribute_support NUMBER,  attribute_confidence NUMBER)</pre>

### Usage Notes

This table function pipes out rows of type DM\_ABN\_DETAIL. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

This function returns details only for a single feature ABN model.

## Examples

The following example demonstrates an ABN model build followed by an invocation of GET\_MODEL\_DETAILS\_ABN table function from Oracle SQL.

```
BEGIN
  -- prepare a settings table to override default algorithm and model type
  CREATE TABLE abn_settings (setting_name VARCHAR2(30),
    setting_value
  VARCHAR2(128));
  INSERT INTO abn_settings VALUES (DBMS_DATA_MINING.ALGO_NAME,
    DBMS_DATA_MINING.ALGO_ADAPTIVE_BAYES_NETWORK);
  INSERT INTO abn_settings VALUES
    (DBMS_DATA_MINING.ABNS_MODEL_TYPE,
    DBMS_DATA_MINING.ABNS_SINGLE_FEATURE);
  COMMIT;
  -- create a model
  DBMS_DATA_MINING.CREATE_MODEL (
    model_name          => 'abn_model',
    function             => DBMS_DATA_MINING.CLASSIFICATION,
    data_table_name     => 'abn_build',
    case_id_column_name => 'id',
    target_column_name  => NULL,
    settings_table_name => 'abn_settings');
END;
/
-- View the (unformatted) results from SQL*Plus
SELECT *
  FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_ABN('abn_model'));
```



## GET\_MODEL\_DETAILS\_AI Function

This table function returns a set of rows that provide the details of an Attribute Importance model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_AI (
  model_name          IN VARCHAR2)
RETURN DM_RANKED_ATTRIBUTES PIPELINED;
```

### Parameters

**Table 44–49** GET\_MODEL\_DETAILS\_AI Function Parameters

Parameter	Description
model_name	Name of the model in the form [ <i>schema_name</i> .] <i>model_name</i> . If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–50** GET\_MODEL\_DETAILS\_AI Function Return Values

Return Value	Description								
DM_RANKED_ATTRIBUTES	Represents a set of rows of type DM_RANKED_ATTRIBUTE. The rows have the following columns: <table border="0" style="margin-left: 20px;"> <tr> <td>attribute_name</td> <td>VARCHAR2(4000,</td> </tr> <tr> <td>attribute_subname</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>importance_value</td> <td>NUMBER,</td> </tr> <tr> <td>rank</td> <td>NUMBER(38))</td> </tr> </table>	attribute_name	VARCHAR2(4000,	attribute_subname	VARCHAR2(4000),	importance_value	NUMBER,	rank	NUMBER(38))
attribute_name	VARCHAR2(4000,								
attribute_subname	VARCHAR2(4000),								
importance_value	NUMBER,								
rank	NUMBER(38))								

### Examples

The following example returns model details for the attribute importance model `AI_SH_sample`, which was created by the sample program `dmaidemo.sql`. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

```
SELECT attribute_name, importance_value, rank
       FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_AI('AI_SH_sample'))
       ORDER BY RANK;
```

ATTRIBUTE_NAME	IMPORTANCE_VALUE	RANK
HOUSEHOLD_SIZE	.151685183	1
CUST_MARITAL_STATUS	.145294546	2
YRS_RESIDENCE	.07838928	3
AGE	.075027496	4
Y_BOX_GAMES	.063039952	5
EDUCATION	.059605314	6
HOME_THEATER_PACKAGE	.056458722	7
OCCUPATION	.054652937	8
CUST_GENDER	.035264741	9
BOOKKEEPING_APPLICATION	.019204751	10
PRINTER_SUPPLIES	0	11
OS_DOC_SET_KANJI	-.00050013	12
FLAT_PANEL_MONITOR	-.00509564	13
BULK_PACK_DISKETTES	-.00540822	14

## GET\_MODEL\_DETAILS\_AI Function

---

COUNTRY_NAME	-.01201116	15
CUST_INCOME_LEVEL	-.03951311	16

## GET\_MODEL\_DETAILS\_GLM Function

This table function returns the coefficient statistics for a Generalized Linear Model.

The same set of statistics is returned for both linear and logistic regression, but statistics that do not apply to the mining function are returned as NULL. For more details, see the Usage Notes.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_GLM (
    model_name          VARCHAR2)
RETURN DM_GLM_COEFF_SET PIPELINED;
```

### Parameters

**Table 44–51** GET\_MODEL\_DETAILS\_GLM Function Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–52** GET\_MODEL\_DETAILS\_GLM Return Values

Return Value	Description																														
DM_GLM_COEFF_SET	Represents a set of rows of type DM_GLM_COEFF. The rows have the following columns: <table border="0" style="margin-left: 20px;"> <tr> <td>(class</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>attribute_name</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>attribute_subname</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>attribute_value</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>coefficient</td> <td>NUMBER,</td> </tr> <tr> <td>std_error</td> <td>NUMBER,</td> </tr> <tr> <td>test_statistic</td> <td>NUMBER,</td> </tr> <tr> <td>p_value</td> <td>NUMBER,</td> </tr> <tr> <td>VIF</td> <td>NUMBER,</td> </tr> <tr> <td>std_coefficient</td> <td>NUMBER,</td> </tr> <tr> <td>lower_coeff_limit</td> <td>NUMBER,</td> </tr> <tr> <td>upper_coeff_limit</td> <td>NUMBER,</td> </tr> <tr> <td>exp_coefficient</td> <td>BINARY_DOUBLE,</td> </tr> <tr> <td>exp_lower_coeff_limit</td> <td>BINARY_DOUBLE,</td> </tr> <tr> <td>exp_upper_coeff_limit</td> <td>BINARY_DOUBLE)</td> </tr> </table>	(class	VARCHAR2(4000),	attribute_name	VARCHAR2(4000),	attribute_subname	VARCHAR2(4000),	attribute_value	VARCHAR2(4000),	coefficient	NUMBER,	std_error	NUMBER,	test_statistic	NUMBER,	p_value	NUMBER,	VIF	NUMBER,	std_coefficient	NUMBER,	lower_coeff_limit	NUMBER,	upper_coeff_limit	NUMBER,	exp_coefficient	BINARY_DOUBLE,	exp_lower_coeff_limit	BINARY_DOUBLE,	exp_upper_coeff_limit	BINARY_DOUBLE)
(class	VARCHAR2(4000),																														
attribute_name	VARCHAR2(4000),																														
attribute_subname	VARCHAR2(4000),																														
attribute_value	VARCHAR2(4000),																														
coefficient	NUMBER,																														
std_error	NUMBER,																														
test_statistic	NUMBER,																														
p_value	NUMBER,																														
VIF	NUMBER,																														
std_coefficient	NUMBER,																														
lower_coeff_limit	NUMBER,																														
upper_coeff_limit	NUMBER,																														
exp_coefficient	BINARY_DOUBLE,																														
exp_lower_coeff_limit	BINARY_DOUBLE,																														
exp_upper_coeff_limit	BINARY_DOUBLE)																														

GET\_MODEL\_DETAILS\_GLM returns a row of statistics for each attribute and one extra row for the intercept, which is identified by a null value in the attribute name. Each row has the DM\_GLM\_COEFF data type. The statistics are described in [Table 44–53](#).

**Table 44–53 DM\_GLM\_COEFF Data Type Description**

Column	Description
class	<p>The non-reference target class for logistic regression. The model is built to predict the probability of this class.</p> <p>The other class (the reference class) is specified in the model setting <code>GLMS_REFERENCE_CLASS_NAME</code>. See <a href="#">Table 44–11, "GLM Settings"</a>.</p> <p>For linear regression, <code>class</code> is null.</p>
attribute_name	<p>The attribute name when there is no subname, or first part of the attribute name when there is a subname. The value of <code>attribute_name</code> is also the name of the column in the case table that is the source for this attribute.</p> <p>For the intercept, <code>attribute_name</code> is null. Intercepts are equivalent to the bias term in SVM models.</p>
attribute_subname	<p>The name of an attribute in a nested table. The full name of a nested attribute has the form:</p> <p><i>attribute_name.attribute_subname</i></p> <p>where <i>attribute_name</i> is the name of the nested column in the case table that is the source for this attribute.</p> <p>If the attribute is not nested, <code>attribute_subname</code> is null. If the attribute is an intercept, both the <code>attribute_name</code> and the <code>attribute_subname</code> are null.</p>
attribute_value	<p>The value of the attribute (categorical attribute only).</p> <p>For numerical attributes, <code>attribute_value</code> is null.</p>
coefficient	The linear coefficient estimate.
std_error	Standard error of the coefficient estimate.
test_statistic	<p>For linear regression, the t-value of the coefficient estimate.</p> <p>For logistic regression, the Wald chi-square value of the coefficient estimate.</p>
p-value	Probability of the <code>test_statistic</code> . Used to analyze the significance of specific attributes in the model.
VIF	Variance Inflation Factor. The value is zero for the intercept. For logistic regression, <code>VIF</code> is null.
std_coefficient	Standardized estimate of the coefficient.
lower_coeff_limit	Lower confidence bound of the coefficient.
upper_coeff_limit	Upper confidence bound of the coefficient.
exp_coefficient	Exponentiated coefficient for logistic regression. For linear regression, <code>exp_coefficient</code> is null.
exp_lower_coeff_limit	Exponentiated coefficient for lower confidence bound of the coefficient for logistic regression. For linear regression, <code>exp_lower_coeff_limit</code> is null.
exp_upper_coeff_limit	Exponentiated coefficient for upper confidence bound of the coefficient for logistic regression. For linear regression, <code>exp_lower_coeff_limit</code> is null.

## Usage Notes

Not all statistics are necessarily returned for each coefficient. Statistics will be null if:

- They do not apply to the mining function. For example, `exp_coefficient` does not apply to linear regression.
- They cannot be computed from a theoretical standpoint. For example, when ridge regression is enabled, the coefficient values are returned with no statistics except VIF if it is enabled. (For information on ridge regression, see [Table 44-11, "GLM Settings"](#).)
- They cannot be computed because of limitations in system resources.
- Their values would be infinity.

## Examples

The following example returns some of the model details for the GLM regression model `GLMR_SH_Regr_sample`, which was created by the sample program `dmglrdem.sql`. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

```
SQL> SELECT *
      FROM (SELECT class, attribute_name, attribute_value, coefficient, std_error
            FROM TABLE(dbms_data_mining.get_model_details_glm(
                        'GLMR_SH_Regr_sample'))
            ORDER BY class, attribute_name, attribute_value)
      WHERE ROWNUM < 11;
```

CLASS	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT	STD_ERROR
	AFFINITY_CARD		-.60686139	.531250033
	BULK_PACK_DISKETTES		-1.9729645	.924531227
	COUNTRY_NAME	Argentina	-1.3340963	1.1942193
	COUNTRY_NAME	Australia	-.340504	5.13688361
	COUNTRY_NAME	Brazil	5.3855269	1.93197882
	COUNTRY_NAME	Canada	4.13393291	2.41283125
	COUNTRY_NAME	China	.74409259	3.59923638
	COUNTRY_NAME	Denmark	-2.5287879	3.18569293
	COUNTRY_NAME	France	-1.0908689	7.18471003
	COUNTRY_NAME	Germany	-1.7472166	2.53689456

## GET\_MODEL\_DETAILS\_GLOBAL Function

This table function returns statistics about the model as a whole. Global details are available for GLM and for association rules.

Separate global details are returned for linear and logistic regression. When ridge regression is enabled, fewer global details are returned. For information about ridge, see [Table 44–11, "GLM Settings"](#).

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_GLOBAL (
    model_name      IN VARCHAR2)
RETURN DM_MODEL_GLOBAL_DETAILS PIPELINED;
```

### Parameters

**Table 44–54** GET\_MODEL\_DETAILS\_GLOBAL Function Parameters

Parameter	Description
model_name	Name of the model in the form [ <i>schema_name</i> .] <i>model_name</i> . If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–55** GET\_MODEL\_DETAILS\_GLOBAL Function Return Values

Return Value	Description
DM_MODEL_GLOBAL_DETAILS	A collection of rows of type DM_MODEL_GLOBAL_DETAIL. The rows have the following columns:  (global_detail_name VARCHAR2(30), global_detail_value NUMBER)

### Global Details for GLM: Linear Regression

**Table 44–56** Global Details for Linear Regression

GLOBAL_DETAIL_NAME	Description
MODEL_DF	Model degrees of freedom
MODEL_SUM_SQUARES	Model sum of squares
MODEL_MEAN_SQUARE	Model mean square
F_VALUE	Model <i>F</i> value statistic
MODEL_F_P_VALUE	Model <i>F</i> value probability
ERROR_DF	Error degrees of freedom
ERROR_SUM_SQUARES	Error sum of squares
ERROR_MEAN_SQUARE	Error mean square
CORRECTED_TOTAL_DF	Corrected total degrees of freedom
CORRECTED_TOT_SS	Corrected total sum of squares
ROOT_MEAN_SQ	Root mean square error

**Table 44–56 (Cont.) Global Details for Linear Regression**

<b>GLOBAL_DETAIL_NAME</b>	<b>Description</b>
DEPENDENT_MEAN	Dependent mean
COEFF_VAR	Coefficient of variation
R_SQ	R-Square
ADJUSTED_R_SQUARE	Adjusted R-Square
AIC	Akaike's information criterion
SBIC	Schwarz's Bayesian information criterion
GMSEP	Estimated mean square error of the prediction, assuming multivariate normality
HOCKING_SP	Hocking Sp statistic
J_P	JP statistic (the final prediction error)
NUM_PARAMS	Number of parameters (the number of coefficients, including the intercept)
NUM_ROWS	Number of rows
MODEL_CONVERGED	Whether or not the model converged. Value is 1 if it converged, or 0 if it did not converge
VALID_COVARIANCE_MATRIX	Valid covariance matrix. Value is 1 if the covariance matrix was computed, or 0 if it was not computed

### Global Details for GLM: Logistic Regression

**Table 44–57 Global Details for Logistic Regression**

<b>GLOBAL_DETAIL_NAME</b>	<b>Description</b>
AIC_INTERCEPT	Akaike's criterion for the fit of the intercept only model
AIC_MODEL	Akaike's criterion for the fit of the intercept and the covariates (predictors) mode
SC_INTERCEPT	Schwarz's Criterion for the fit of the intercept only model
SC_MODEL	Schwarz's Criterion for the fit of the intercept and the covariates (predictors) model
NEG2_LL_INTERCEPT	-2 log likelihood of the intercept only model
NEG2_LL_MODEL	-2 log likelihood of the model
LR_DF	Likelihood ratio degrees of freedom
LR_CHI_SQ	Likelihood ratio chi-square value
LR_CHI_SQ_P_VALUE	Likelihood ratio chi-square probability value
PSEUDO_R_SQ_CS	Pseudo R-square Cox and Snell
PSEUDO_R_SQ_N	Pseudo R-square Nagelkerke
DEPENDENT_MEAN	Dependent mean
PCT_CORRECT	Percent of correct predictions
PCT_INCORRECT	Percent of incorrectly predicted rows
PCT_TIED	Percent of cases where probability for both cases is the same
NUM_PARAMS	Number of parameters (the number of coefficients, including the intercept)

**Table 44–57 (Cont.) Global Details for Logistic Regression**

GLOBAL_DETAIL_NAME	Description
NUM_ROWS	Number of rows
MODEL_CONVERGED	Whether or not the model converged. Value is 1 if it converged, or 0 if it did not converge.
VALID_COVARIANCE_MATRIX	Valid covariance matrix. Value is 1 if the covariance matrix was computed, or 0 if the covariance matrix not computed

## Global Detail for Association Rules

A single global detail is produced by an Association model.

**Table 44–58 Global Detail for Association Rules**

GLOBAL_DETAIL_NAME	Description
RULE_COUNT	The number of association rules in the model.

## Examples

The following example returns the global model details for the GLM regression model `GLMR_SH_Regr_sample`, which was created by the sample program `dmglrdem.sql`. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

```
SELECT *
  FROM TABLE(dbms_data_mining.get_model_details_global(
              'GLMR_SH_Regr_sample'))
ORDER BY global_detail_name;
GLOBAL_DETAIL_NAME          GLOBAL_DETAIL_VALUE
-----
ADJUSTED_R_SQUARE          .731412557
AIC                        5931.814
COEFF_VAR                  18.1711243
CORRECTED_TOTAL_DF        1499
CORRECTED_TOT_SS          278740.504
DEPENDENT_MEAN             38.892
ERROR_DF                   1433
ERROR_MEAN_SQUARE          49.9440956
ERROR_SUM_SQUARES          71569.8891
F_VALUE                    62.8492452
GMSEP                      52.280819
HOCKING_SP                 .034877162
J_P                        52.1749319
MODEL_CONVERGED            1
MODEL_DF                   66
MODEL_F_P_VALUE            0
MODEL_MEAN_SQUARE          3138.94871
MODEL_SUM_SQUARES          207170.615
NUM_PARAMS                 67
NUM_ROWS                   1500
ROOT_MEAN_SQ               7.06711367
R_SQ                       .743238288
SBIC                       6287.79977
VALID_COVARIANCE_MATRIX    1
```



## GET\_MODEL\_DETAILS\_KM Function

This table function returns a set of rows that provide the details of a *k*-Means clustering model.

You can provide input to `GET_MODEL_DETAILS_KM` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, `GET_MODEL_DETAILS_KM` returns all the information about the model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_KM (
    model_name          VARCHAR2,
    cluster_id         NUMBER   DEFAULT NULL,
    attribute           VARCHAR2 DEFAULT NULL,
    centroid            NUMBER   DEFAULT 1,
    histogram           NUMBER   DEFAULT 1,
    rules               NUMBER   DEFAULT 2,
    attribute_subname   VARCHAR2 DEFAULT NULL)
RETURN DM_CLUSTERS PIPELINED;
```

### Parameters

**Table 44–59** *GET\_MODEL\_DETAILS\_KM Function Parameters*

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is used.
<code>cluster_id</code>	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
<code>attribute</code>	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise the details for all attributes are returned.
<code>centroid</code>	This parameter accepts the following values: <ul style="list-style-type: none"> <li>■ 1 — Details about centroids are returned (default)</li> <li>■ 0 — Details about centroids are not returned</li> </ul>
<code>histogram</code>	This parameter accepts the following values: <ul style="list-style-type: none"> <li>■ 1 — Details about histograms are returned (default)</li> <li>■ 0 — Details about histograms are not returned</li> </ul>
<code>rules</code>	This parameter accepts the following values: <ul style="list-style-type: none"> <li>■ 2 — Details about rules are returned (default)</li> <li>■ 1 — Rule summaries are returned</li> <li>■ 0 — No information about rules is returned</li> </ul>

## Return Values

**Table 44–60** GET\_MODEL\_DETAILS\_KM Function Return Values

Return Value	Description
DM_CLUSTERS	<p>Represents a set of rows of type DM_CLUSTER. The rows have the following columns:</p> <pre>(id                NUMBER,  cluster_id        VARCHAR2(4000),  record_count      NUMBER,  parent            NUMBER,  tree_level        NUMBER,  dispersion        NUMBER,  split_predicate   DM_PREDICATES,  child             DM_CHILDREN,  centroid          DM_CENTROIDS,  histogram         DM_HISTOGRAMS,  rule              DM_RULE)</pre>

The `split_predicate` column of `DM_CLUSTER` returns a nested table of type `DM_PREDICATES`. Each row, of type `DM_PREDICATE`, has the following columns:

```
(attribute_name    VARCHAR2(4000),
 attribute_subname VARCHAR2(4000),
 conditional_operator CHAR(2) /*=, <>, <, >, <=, >=*/,
 attribute_num_value NUMBER,
 attribute_str_value VARCHAR2(4000),
 attribute_support  NUMBER,
 attribute_confidence NUMBER)
```

The `child` column of `DM_CLUSTER` returns a nested table of type `DM_CHILDREN`. The rows, of type `DM_CHILD`, have a single column of type `NUMBER`, which contains the identifiers of each child.

The `centroid` column of `DM_CLUSTER` returns a nested table of type `DM_CENTROIDS`. The rows, of type `DM_CENTROID`, have the following columns:

```
(attribute_name    VARCHAR2(4000),
 attribute_subname VARCHAR2(4000),
 mean              NUMBER,
 mode_value        VARCHAR2(4000),
 variance          NUMBER)
```

The `histogram` column of `DM_CLUSTER` returns a nested table of type `DM_HISTOGRAMS`. The rows, of type `DM_HISTOGRAM_BIN`, have the following columns:

```
(attribute_name    VARCHAR2(4000),
 attribute_subname VARCHAR2(4000),
 bin_id            NUMBER,
 lower_bound       NUMBER,
 upper_bound       NUMBER,
 label             VARCHAR2(4000),
 count             NUMBER)
```

**Table 44–60 (Cont.) GET\_MODEL\_DETAILS\_KM Function Return Values**

Return Value	Description
	The rule column of DM_CLUSTER returns a single row of type DM_RULE. The columns are:
	(rule_id                    INTEGER, antecedent                DM_PREDICATES, consequent                DM_PREDICATES, rule_support               NUMBER, rule_confidence          NUMBER, rule_lift                  NUMBER, antecedent_support       NUMBER, consequent_support       NUMBER, number_of_items          INTEGER)
	The antecedent and consequent columns of DM_RULE each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:
	(attribute_name            VARCHAR2(4000), attribute_subname         VARCHAR2(4000), conditional_operator      CHAR(2)/*=,<>,<,>,<=,>=*/, attribute_num_value        NUMBER, attribute_str_value        VARCHAR2(4000), attribute_support          NUMBER, attribute_confidence       NUMBER)

## Usage Notes

The table function pipes out rows of type DM\_CLUSTERS. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

## Examples

The following example returns model details for the *k*-Means clustering model KM\_SH\_Clus\_sample, which was created by the sample program dmkmdemo.sql. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

```
SELECT T.id                clu_id,
       T.record_count     rec_cnt,
       T.parent            parent,
       T.tree_level       tree_level,
       T.dispersion        dispersion
FROM (SELECT *
      FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_KM(
                  'KM_SH_Clus_sample'))
      ORDER BY id) T
WHERE ROWNUM < 6;
```

CLU_ID	REC_CNT	PARENT	TREE_LEVEL	DISPERSION
1	1500		1	5.9152211
2	638	1	2	3.98458982
3	862	1	2	5.83732097
4	376	3	3	5.05192137
5	486	3	3	5.42901522

## GET\_MODEL\_DETAILS\_NB Function

This table function returns a set of rows that provide the details of a Naive Bayes model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_NB (
    model_name      IN      VARCHAR2)
RETURN DM_NB_DETAILS PIPELINED;
```

### Parameters

**Table 44–61 GET\_MODEL\_DETAILS\_NB Function Parameters**

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–62 GET\_MODEL\_DETAILS\_NB Function Return Values**

Return Value	Description																				
DM_NB_DETAILS	<p>Represents a set of rows of type DM_NB_DETAIL. The rows have the following columns:</p> <table border="0"> <tr> <td>target_attribute_name</td> <td>VARCHAR2(30),</td> </tr> <tr> <td>target_attribute_str_value</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>target_attribute_num_value</td> <td>NUMBER,</td> </tr> <tr> <td>prior_probability</td> <td>NUMBER,</td> </tr> <tr> <td>conditionals</td> <td>DM_CONDITIONALS)</td> </tr> </table> <p>The conditionals column of DM_NB_DETAIL returns a nested table of type DM_CONDITIONALS. The rows, of type DM_CONDITIONAL, have the following columns:</p> <table border="0"> <tr> <td>attribute_name</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>attribute_subname</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>attribute_str_value</td> <td>VARCHAR2(4000),</td> </tr> <tr> <td>attribute_num_value</td> <td>NUMBER,</td> </tr> <tr> <td>conditional_probability</td> <td>NUMBER)</td> </tr> </table>	target_attribute_name	VARCHAR2(30),	target_attribute_str_value	VARCHAR2(4000),	target_attribute_num_value	NUMBER,	prior_probability	NUMBER,	conditionals	DM_CONDITIONALS)	attribute_name	VARCHAR2(4000),	attribute_subname	VARCHAR2(4000),	attribute_str_value	VARCHAR2(4000),	attribute_num_value	NUMBER,	conditional_probability	NUMBER)
target_attribute_name	VARCHAR2(30),																				
target_attribute_str_value	VARCHAR2(4000),																				
target_attribute_num_value	NUMBER,																				
prior_probability	NUMBER,																				
conditionals	DM_CONDITIONALS)																				
attribute_name	VARCHAR2(4000),																				
attribute_subname	VARCHAR2(4000),																				
attribute_str_value	VARCHAR2(4000),																				
attribute_num_value	NUMBER,																				
conditional_probability	NUMBER)																				

### Usage Notes

The table function pipes out rows of type DM\_NB\_DETAILS. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

### Examples

The following query is from the sample program `dmnbdemo.sql`. It returns model details about the model `NB_SH_Clas_sample`. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

The query creates labels from the bin boundary tables that were used to bin the training data. It replaces the attribute values with the labels. For numeric bins, the labels are (*lower\_boundary,upper\_boundary*); for categorical bins, the label matches the value it represents. (This method of categorical label representation will

only work for cases where one value corresponds to one bin.) The target was not binned.

```

WITH
  bin_label_view AS (
    SELECT col, bin, (DECODE(bin,'1','[', '(') || lv || ',' || val || ']') label
      FROM (SELECT col,
                   bin,
                   LAST_VALUE(val) OVER (
                     PARTITION BY col ORDER BY val
                     ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING) lv,
                   val
             FROM nb_sh_sample_num)
    UNION ALL
    SELECT col, bin, val label
      FROM nb_sh_sample_cat
  ),
  model_details AS (
    SELECT T.target_attribute_name                                tname,
           TO_CHAR(
             NVL(T.target_attribute_num_value,T.target_attribute_str_value)) tval,
           C.attribute_name                                       pname,
           NVL(L.label, NVL(C.attribute_str_value, C.attribute_num_value)) pval,
           T.prior_probability                                    priorp,
           C.conditional_probability                            condp
      FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NB('NB_SH_Clas_sample')) T,
           TABLE(T.conditionals) C,
           bin_label_view L
     WHERE C.attribute_name = L.col (+) AND
           (NVL(C.attribute_str_value,C.attribute_num_value) = L.bin(+))
     ORDER BY 1,2,3,4,5,6
  )
  SELECT tname, tval, pname, pval, priorp, condp
     FROM model_details
     WHERE ROWNUM < 11;

```

TNAME	TVAL	PNAME	PVAL	PRIORP	CONDP
AFFINITY_CARD	0	AGE	(24,30]	.6500	.1714
AFFINITY_CARD	0	AGE	(30,35]	.6500	.1509
AFFINITY_CARD	0	AGE	(35,40]	.6500	.1125
AFFINITY_CARD	0	AGE	(40,46]	.6500	.1134
AFFINITY_CARD	0	AGE	(46,53]	.6500	.1071
AFFINITY_CARD	0	AGE	(53,90]	.6500	.1312
AFFINITY_CARD	0	AGE	[17,24]	.6500	.2134
AFFINITY_CARD	0	BOOKKEEPING_APPLICATION	0	.6500	.1500
AFFINITY_CARD	0	BOOKKEEPING_APPLICATION	1	.6500	.8500
AFFINITY_CARD	0	BULK_PACK_DISKETTES	0	.6500	.3670

## GET\_MODEL\_DETAILS\_NMF Function

This table function returns a set of rows that provide the details of a Non-Negative Matrix Factorization model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF (
    model_name      IN      VARCHAR2)
RETURN DM_NMF_FEATURE_SET PIPELINED;
```

### Parameters

**Table 44–63** GET\_MODEL\_DETAILS\_NMF Function Parameters

Parameter	Description
model_name	Name of the model in the form <i>[schema_name.]model_name</i> . If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–64** GET\_MODEL\_DETAILS\_NMF Function Return Values

Return Value	Description
DM_NMF_FEATURE_SET	<p>Represents a set of rows of DM_NMF_FEATURE. The rows have the following columns:</p> <pre>(feature_id      NUMBER,  mapped_feature_id VARCHAR2(4000),  attribute_set    DM_NMF_ATTRIBUTE_SET)</pre> <p>The attribute_set column of DM_NMF_FEATURE returns a nested table of type DM_NMF_ATTRIBUTE_SET. The rows, of type DM_NMF_ATTRIBUTE, have the following columns:</p> <pre>(attribute_name  VARCHAR2(4000),  attribute_subname VARCHAR2(4000),  attribute_value  VARCHAR2(4000),  coefficient      NUMBER)</pre>

### Usage Notes

The table function pipes out rows of type DM\_NMF\_FEATURE\_SET. For information on Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

### Examples

The following example returns model details for the feature extraction model NMF\_SH\_Sample, which was created by the sample program `dmmdemo.sql`. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

```
SELECT * FROM (
SELECT F.feature_id,
       A.attribute_name,
       A.attribute_value,
       A.coefficient
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_NMF('NMF_SH_Sample')) F,
TABLE(F.attribute_set) A
```

```
ORDER BY feature_id,attribute_name,attribute_value
) WHERE ROWNUM < 11;
```

FEATURE_ID	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
1	AFFINITY_CARD		.051208078859308
1	AGE		.0390513260041573
1	BOOKKEEPING_APPLICATION		.0512734004239326
1	BULK_PACK_DISKETTES		.232471260895683
1	COUNTRY_NAME	Argentina	.00766817464479959
1	COUNTRY_NAME	Australia	.000157637881096675
1	COUNTRY_NAME	Brazil	.0031409632415604
1	COUNTRY_NAME	Canada	.00144213099311427
1	COUNTRY_NAME	China	.000102279310968754
1	COUNTRY_NAME	Denmark	.000242424084307513

## GET\_MODEL\_DETAILS\_OC Function

This table function returns a set of rows that provide the details of an O-Cluster clustering model. The rows are an enumeration of the clustering patterns generated during the creation of the model.

You can provide input to `GET_MODEL_DETAILS_OC` to request specific information about the model, thus improving the performance of the query. If you do not specify filtering parameters, `GET_MODEL_DETAILS_OC` returns all the information about the model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_OC (
    model_name          VARCHAR2,
    cluster_id         NUMBER   DEFAULT NULL,
    attribute           VARCHAR2 DEFAULT NULL,
    centroid            NUMBER   DEFAULT 1,
    histogram           NUMBER   DEFAULT 1,
    rules              NUMBER   DEFAULT 2)
RETURN DM_CLUSTERS PIPELINED;
```

### Parameters

**Table 44–65** *GET\_MODEL\_DETAILS\_OC Function Parameters*

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is used.
<code>cluster_id</code>	The ID of a cluster in the model. When a valid cluster ID is specified, only the details of this cluster are returned. Otherwise the details for all clusters are returned.
<code>attribute</code>	The name of an attribute. When a valid attribute name is specified, only the details of this attribute are returned. Otherwise the details for all attributes are returned.
<code>centroid</code>	This parameter accepts the following values: <ul style="list-style-type: none"> <li>■ 1 — Details about centroids are returned (default)</li> <li>■ 0 — Details about centroids are not returned</li> </ul>
<code>histogram</code>	This parameter accepts the following values: <ul style="list-style-type: none"> <li>■ 1 — Details about histograms are returned (default)</li> <li>■ 0 — Details about histograms are not returned</li> </ul>
<code>rules</code>	This parameter accepts the following values: <ul style="list-style-type: none"> <li>■ 2 — Details about rules are returned (default)</li> <li>■ 1 — Rule summaries are returned</li> <li>■ 0 — No information about rules is returned</li> </ul>



## Return Values

**Table 44–66** *GET\_MODEL\_DETAILS\_OC Function Return Values*

Return Value	Description
DM_CLUSTERS	<p>Represents a set of rows of type DM_CLUSTER. The rows have the following columns:</p> <pre>(id          NUMBER,  cluster_id  VARCHAR2(4000),  record_count NUMBER,  parent      NUMBER,  tree_level  NUMBER,  dispersion  NUMBER,  split_predicate DM_PREDICATES,  child       DM_CHILDREN,  centroid    DM_CENTROIDS,  histogram   DM_HISTOGRAMS,  rule        DM_RULE)</pre>

The `split_predicate` column of `DM_CLUSTER` returns a nested table of type `DM_PREDICATES`. Each row, of type `DM_PREDICATE`, has the following columns:

```
(attribute_name  VARCHAR2(4000),
 attribute_subname VARCHAR2(4000),
 conditional_operator CHAR(2) /*=, <>, <, >, <=, >=*/,
 attribute_num_value NUMBER,
 attribute_str_value VARCHAR2(4000),
 attribute_support NUMBER,
 attribute_confidence NUMBER)
```

The `child` column of `DM_CLUSTER` returns a nested table of type `DM_CHILDREN`. The rows, of type `DM_CHILD`, have a single column of type `NUMBER`, which contains the identifiers of each child.

The `centroid` column of `DM_CLUSTER` returns a nested table of type `DM_CENTROIDS`. The rows, of type `DM_CENTROID`, have the following columns:

```
(attribute_name  VARCHAR2(4000),
 attribute_subname VARCHAR2(4000),
 mean            NUMBER,
 mode_value      VARCHAR2(4000),
 variance        NUMBER)
```

The `histogram` column of `DM_CLUSTER` returns a nested table of type `DM_HISTOGRAMS`. The rows, of type `DM_HISTOGRAM_BIN`, have the following columns:

```
(attribute_name  VARCHAR2(4000),
 attribute_subname VARCHAR2(4000),
 bin_id          NUMBER,
 lower_bound     NUMBER,
 upper_bound     NUMBER,
 label           VARCHAR2(4000),
 count           NUMBER)
```

**Table 44-66 (Cont.) GET\_MODEL\_DETAILS\_OC Function Return Values**

Return Value	Description
	The rule column of DM_CLUSTER returns a single row of type DM_RULE. The columns are:
	(rule_id                    INTEGER, antecedent                DM_PREDICATES, consequent                DM_PREDICATES, rule_support              NUMBER, rule_confidence          NUMBER, rule_lift                 NUMBER, antecedent_support       NUMBER, consequent_support       NUMBER, number_of_items          INTEGER)
	The antecedent and consequent columns each return nested tables of type DM_PREDICATES. The rows, of type DM_PREDICATE, have the following columns:
	(attribute_name            VARCHAR2(4000), attribute_subname         VARCHAR2(4000), conditional_operator      CHAR(2)/*=,<>,<,>,<=,>=*/, attribute_num_value       NUMBER, attribute_str_value       VARCHAR2(4000), attribute_support          NUMBER, attribute_confidence      NUMBER)

## Usage Notes

The table function pipes out rows of type DM\_CLUSTER. For information about Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

## Examples

The following example returns model details for the clustering model OC\_SH\_Clus\_sample, which was created by the sample program dmocdemo.sql. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

For each cluster in this example, the split predicate indicates the attribute and the condition used to assign records to the cluster's children during model build. It provides an important piece of information on how the population within a cluster can be divided up into two smaller clusters.

```
SELECT clu_id, attribute_name, op, s_value
   FROM (SELECT a.id clu_id, sp.attribute_name, sp.conditional_operator op,
              sp.attribute_str_value s_value
          FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_OC(
                    'OC_SH_Clus_sample')) a,
              TABLE(a.split_predicate) sp
          ORDER BY a.id, op, s_value)
 WHERE ROWNUM < 11;
```

CLU_ID	ATTRIBUTE_NAME	OP	S_VALUE
1	OCCUPATION	IN	?
1	OCCUPATION	IN	Armed-F
1	OCCUPATION	IN	Cleric.
1	OCCUPATION	IN	Crafts
2	OCCUPATION	IN	?

2 OCCUPATION	IN Armed-F
2 OCCUPATION	IN Cleric.
3 OCCUPATION	IN Exec.
3 OCCUPATION	IN Farming
3 OCCUPATION	IN Handler

## GET\_MODEL\_DETAILS\_SVM Function

This table function returns a set of rows that provide the details of a linear Support Vector Machine (SVM) model. If invoked for nonlinear SVM, it returns ORA-40215.

In linear SVM models, only nonzero coefficients are stored. This reduces storage and speeds up model loading. As a result, if an attribute is missing in the coefficient list returned by GET\_MODEL\_DETAILS\_SVM, then the coefficient of this attribute should be interpreted as zero.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM (
    model_name          VARCHAR2,
    reverse_coef       NUMBER DEFAULT 0)
RETURN DM_SVM_LINEAR_COEFF_SET PIPELINED;
```

### Parameters

**Table 44–67** GET\_MODEL\_DETAILS\_SVM Function Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.
reverse_coef	Whether or not GET_MODEL_DETAILS_SVM should transform the attribute coefficients using the original attribute transformations.  When reverse_coef is set to 0 (default), GET_MODEL_DETAILS_SVM returns the coefficients directly from the model without applying transformations.  When reverse_coef is set to 1, GET_MODEL_DETAILS_SVM transforms the coefficients and bias by applying the normalization shifts and scales that were generated using automatic data preparation.  See Usage Note 4.

### Return Values

**Table 44–68** GET\_MODEL\_DETAILS\_SVM Function Return Values

Return Value	Description
DM_SVM_LINEAR_COEFF_SET	Represents a set of rows of type DM_SVM_LINEAR_COEFF. The rows have the following columns:  (class            VARCHAR2(4000), attribute_set    DM_SVM_ATTRIBUTE_SET)  The attribute_set column returns a nested table of type DM_SVM_ATTRIBUTE_SET. The rows, of type DM_SVM_ATTRIBUTE, have the following columns:  (attribute_name    VARCHAR2(4000), attribute_subname  VARCHAR2(4000), attribute_value    VARCHAR2(4000), coefficient        NUMBER)  See Usage Notes.

## Usage Notes

1. This table function pipes out rows of type `DM_SVM_LINEAR_COEFF`. For information on Data Mining data types and piped output from table functions, see "Data Types" on page 44-20.
2. The `class` column of `DM_SVM_LINEAR_COEFF` contains classification target values. For SVM regression models, `class` is null. For each classification target value, a set of coefficients is returned. For binary classification, one-class classification, and regression models, only a single set of coefficients is returned.
3. The `attribute_value` column in `DM_SVM_ATTRIBUTE_SET` is used for categorical attributes.
4. `GET_MODEL_DETAILS` functions preserve model transparency by automatically reversing the transformations applied during the build process. Thus the attributes returned in the model details are the original attributes (or a close approximation of the original attributes) used to build the model.

The coefficients are related to the transformed, not the original, attributes. When returned directly with the model details, the coefficients may not provide meaningful information. If you want `GET_MODEL_DETAILS_SVM` to transform the coefficients such that they relate to the original attributes, set the `reverse_coef` parameter to 1.

## Examples

The following example returns model details for the SVM classification model `SVMC_SH_Clas_sample`, which was created by the sample program `dmsvcdem.sql`. For information about the sample programs, see *Oracle Data Mining Administrator's Guide*.

```
WITH
  mod_dtls AS (
    SELECT *
      FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('SVMC_SH_Clas_sample'))
  ),
  model_details AS (
    SELECT D.class, A.attribute_name, A.attribute_value, A.coefficient
      FROM mod_dtls D,
           TABLE(D.attribute_set) A
     ORDER BY D.class, ABS(A.coefficient) DESC
  )
SELECT class, attribute_name aname, attribute_value aval, coefficient coeff
  FROM model_details
 WHERE ROWNUM < 11;
```

CLASS	ANAME	AVAL	COEFF
1			-2.85
1	BOOKKEEPING_APPLICATION		1.11
1	OCCUPATION	Other	-.94
1	HOUSEHOLD_SIZE	4-5	.88
1	CUST_MARITAL_STATUS	Married	.82
1	YRS_RESIDENCE		.76
1	HOUSEHOLD_SIZE	6-8	-.74
1	OCCUPATION	Exec.	.71
1	EDUCATION	11th	-.71
1	EDUCATION	Masters	.63

## GET\_MODEL\_DETAILS\_XML Function

This function returns an XML object that provides the details of a Decision Tree model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_DETAILS_XML (
    model_name      IN      VARCHAR2)
RETURN XMLTYPE;
```

### Parameters

**Table 44–69** GET\_MODEL\_DETAILS\_XML Function Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–70** GET\_MODEL\_DETAILS\_XML Function Return Value

Return Value	Description
XMLTYPE	The XML definition for the decision tree model. See <a href="#">Chapter 259, "XMLTYPE"</a> for details.  The XML conforms to the Data Mining Group Predictive Model Markup Language (PMML) version 2.1 specification. The specification is available at <a href="http://www.dmg.org">http://www.dmg.org</a> .

### Usage Notes

Special characters that cannot be displayed by Oracle XML are converted to '#'.

### Examples

The following statements in SQL\*Plus return the details of the decision tree model dt\_sh\_clas\_sample. This model is created by the program dmdtdemo.sql, one of the sample data mining programs provided with Oracle Database Examples.

Note: The "&quot;" characters you will see in the XML output are a result of SQL\*Plus behavior. To display the XML in proper format, cut and past it into a file and open the file in a browser.

```
column dt_details format a320
SELECT
  dbms_data_mining.get_model_details_xml('dt_sh_clas_sample')
  AS DT_DETAILS
FROM dual;
```

```
DT_DETAILS
```

```
-----
<PMML version="2.1">
  <Header copyright="Copyright (c) 2004, Oracle Corporation. All rights
    reserved."/>
  <DataDictionary numberOfFields="9">
    <DataField name="AFFINITY_CARD" optype="categorical"/>
    <DataField name="AGE" optype="continuous"/>
```

```

<DataField name="BOOKKEEPING_APPLICATION" optype="continuous" />
<DataField name="CUST_MARITAL_STATUS" optype="categorical" />
<DataField name="EDUCATION" optype="categorical" />
<DataField name="HOUSEHOLD_SIZE" optype="categorical" />
<DataField name="OCCUPATION" optype="categorical" />
<DataField name="YRS_RESIDENCE" optype="continuous" />
<DataField name="Y_BOX_GAMES" optype="continuous" />
</DataDictionary>
<TreeModel modelName="DT_SH_CLAS_SAMPLE" functionName="classification"
  splitCharacteristic="binarySplit">
  <Extension name="buildSettings">
    <Setting name="TREE_IMPURITY_METRIC" value="TREE_IMPURITY_GINI" />
    <Setting name="TREE_TERM_MAX_DEPTH" value="7" />
    <Setting name="TREE_TERM_MINPCT_NODE" value=".05" />
    <Setting name="TREE_TERM_MINPCT_SPLIT" value=".1" />
    <Setting name="TREE_TERM_MINREC_NODE" value="10" />
    <Setting name="TREE_TERM_MINREC_SPLIT" value="20" />
    <costMatrix>
      <costElement>
        <actualValue>0</actualValue>
        <predictedValue>0</predictedValue>
        <cost>0</cost>
      </costElement>
      <costElement>
        <actualValue>0</actualValue>
        <predictedValue>1</predictedValue>
        <cost>1</cost>
      </costElement>
      <costElement>
        <actualValue>1</actualValue>
        <predictedValue>0</predictedValue>
        <cost>8</cost>
      </costElement>
      <costElement>
        <actualValue>1</actualValue>
        <predictedValue>1</predictedValue>
        <cost>0</cost>
      </costElement>
    </costMatrix>
  </Extension>
</MiningSchema>
.
.
.
.
.
.
</Node>
</Node>
</TreeModel>
</PMML>

```

## GET\_MODEL\_SETTINGS Function

The `GET_MODEL_SETTINGS` function was deprecated in Oracle Data Mining 11g Release 1 (11.1). It was replaced with the data dictionary view `*_MINING_MODEL_SETTINGS`. `USER_`, `ALL_`, and `DBA_` versions of the view are available. See *Oracle Data Mining Application Developer's Guide*.

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---



---

**Note:** See ["Deprecated Subprograms"](#) on page 44-8

---



---

This table function returns the list of settings that were used to build the model.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_SETTINGS (
    model_name          IN VARCHAR2)
RETURN DM_MODEL_SETTINGS PIPELINED;
```

### Parameters

**Table 44–71** *GET\_MODEL\_SETTINGS Function Parameters*

Parameter	Description
<code>model_name</code>	Name of the model in the form <code>[schema_name.]model_name</code> . If you do not specify a schema, your own schema is used.

### Return Values

**Table 44–72** *GET\_MODEL\_SETTINGS Function Return Values*

Return Value	Description
<code>DM_MODEL_SETTINGS</code>	Represents a set of rows of type <code>DM_MODEL_SETTING</code> . The rows have the following columns: <div style="margin-left: 20px;"> <code>(setting_name    VARCHAR2(30) ,</code>  <code>setting_value    VARCHAR2(128))</code> </div>

### Usage Notes

The table function pipes out rows of type `DM_MODEL_SETTING`. For information about Data Mining data types and piped output from table functions, see ["Data Types"](#) on page 44-20.

You can use this table function to determine the settings that were used to build the model. This is purely for informational purposes only — you cannot alter the model to adopt new settings.

### Examples

Assume that you have built a classification model `census_model` using the Naive Bayes algorithm. You can retrieve the model settings using Oracle SQL as follows:

```
SELECT setting_name, setting_value
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_SETTINGS('census_model'));
```



## GET\_MODEL\_SIGNATURE Function

The GET\_MODEL\_SIGNATURE function was deprecated in Oracle Data Mining 11g Release 1 (11.1). It was replaced with the data dictionary view \*\_MINING\_MODEL\_ATTRIBUTES. USER\_, ALL\_, and DBA\_ versions of the view are available. See *Oracle Data Mining Application Developer's Guide*.

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

**Note:** See "[Deprecated Subprograms](#)" on page 44-8

---

This table function returns the model signature, which lists the column attributes used to build the model and which should be present in the scoring data.

The case identifier is not considered a mining attribute. For classification and regression models, the target attribute is also not considered part of the model signature.

**See Also:** Instead of using the GET\_MODEL\_SIGNATURE function, you can query the data dictionary view, ALL\_MINING\_MODEL\_ATTRIBUTES. See the *Oracle Data Mining Application Developer's Guide*.

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_SIGNATURE (
    model_name          IN VARCHAR2)
RETURN DM_MODEL_SIGNATURE PIPELINED;
```

### Parameters

**Table 44-73** GET\_MODEL\_SIGNATURE Function Parameters

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.

### Return Values

**Table 44-74** GET\_MODEL\_SIGNATURE Function Return Values

Return Value	Description
DM_MODEL_SIGNATURE	Represents a set of rows of type DM_MODEL_SIGNATURE_ATTRIBUTE. The rows have the following columns: (attribute_name        VARCHAR2 (30) , attribute_type        VARCHAR2 (106))

### Usage Notes

This table function pipes out rows of type DM\_MODEL\_SIGNATURE. For information on Data Mining data types and piped output from table functions, see "[Data Types](#)" on page 44-20.

**Examples**

Assume that you have built a classification model `census_model` using the Naive Bayes algorithm. You can retrieve the model details using Oracle SQL as follows:

```
SELECT attribute_name, attribute_type
FROM TABLE(DBMS_DATA_MINING.GET_MODEL_SIGNATURE('census_model'));
```

## GET\_MODEL\_TRANSFORMATIONS Function

This function returns the transformation expressions embedded in the specified model.

### See Also:

["About Transformation Lists"](#) on page 45-7

["GET\\_TRANSFORM\\_LIST Procedure"](#) on page 44-97

["CREATE\\_MODEL Procedure"](#) on page 44-50

### Syntax

```
DBMS_DATA_MINING.GET_MODEL_TRANSFORMATIONS (
    model_name      IN VARCHAR2)
RETURN DM_TRANSFORMS PIPELINED;
```

### Parameters

**Table 44-75 GET\_MODEL\_TRANSFORMATIONS Function Parameters**

Parameter	Description
model_name	Name of the model in the form <i>[schema_name.]model_name</i> . If you do not specify a schema, your own schema is used.

### Return Values

**Table 44-76 GET\_MODEL\_TRANSFORMATIONS Function Return Value**

Return Value	Description										
DM_TRANSFORMS	The transformation expressions embedded in <i>model_name</i> . The DM_TRANSFORMS type is a table of DM_TRANSFORM objects. Each DM_TRANSFORM has these fields: <table border="1" data-bbox="649 1249 1088 1375"> <thead> <tr> <th>Field Name</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>attribute_name</td> <td>VARCHAR2(4000)</td> </tr> <tr> <td>attribute_subname</td> <td>VARCHAR2(4000)</td> </tr> <tr> <td>expression</td> <td>CLOB</td> </tr> <tr> <td>reverse_expression</td> <td>CLOB</td> </tr> </tbody> </table>	Field Name	Type	attribute_name	VARCHAR2(4000)	attribute_subname	VARCHAR2(4000)	expression	CLOB	reverse_expression	CLOB
Field Name	Type										
attribute_name	VARCHAR2(4000)										
attribute_subname	VARCHAR2(4000)										
expression	CLOB										
reverse_expression	CLOB										

### Usage Notes

When Automatic Data Preparation (ADP) is enabled, both automatic and user-defined transformations may be associated with an attribute. In this case, the user-defined transformations are evaluated before the automatic transformations.

### Examples

In this example, several columns in the SH.CUSTOMERS table are used to create a Naive Bayes model. A transformation expression is specified for one of the columns. The model does not use ADP.

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_income_level, cust_credit_limit
  FROM sh.customers;
```

```
describe mining_data
Name                                         Null?      Type
```

```

-----
CUST_ID                                NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                     NOT NULL NUMBER(4)
CUST_INCOME_LEVEL                       VARCHAR2(30)
CUST_CREDIT_LIMIT                       NUMBER

CREATE TABLE settings_nb(
    setting_name VARCHAR2(30),
    setting_value VARCHAR2(30));
BEGIN
    INSERT INTO settings_nb (setting_name, setting_value) VALUES
        (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
    INSERT INTO settings_nb (setting_name, setting_value) VALUES
        (dbms_data_mining.prep_auto, dbms_data_mining.prep_auto_off);
    COMMIT;
END;
/
DECLARE
    mining_data_xforms    dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.SET_TRANSFORM (
        xform_list        => mining_data_xforms,
        attribute_name    => 'cust_year_of_birth',
        attribute_subname => null,
        expression        => 'cust_year_of_birth + 10',
        reverse_expression => 'cust_year_of_birth - 10');
    dbms_data_mining.CREATE_MODEL (
        model_name        => 'new_model',
        mining_function   => dbms_data_mining.classification,
        data_table_name   => 'mining_data',
        case_id_column_name => 'cust_id',
        target_column_name => 'cust_income_level',
        settings_table_name => 'settings_nb',
        data_schema_name  => null,
        settings_schema_name => null,
        xform_list        => mining_data_xforms );
END;
/
SELECT attribute_name, TO_CHAR(expression), TO_CHAR(reverse_expression)
       FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('new_model'));

ATTRIBUTE_NAME    TO_CHAR(EXPRESSION)    TO_CHAR(REVERSE_EXPRESSION)
-----
CUST_YEAR_OF_BIRTH cust_year_of_birth + 10    cust_year_of_birth - 10

```

## GET\_TRANSFORM\_LIST Procedure

This procedure converts transformation expressions specified as `DM_TRANSFORMS` to a transformation list (`TRANSFORM_LIST`) that can be used in creating a model. `DM_TRANSFORMS` is returned by the `GET_MODEL_TRANSFORMATIONS` function.

You can also use routines in the `DBMS_DATA_MINING_TRANSFORM` package to construct a transformation list.

### See Also:

["About Transformation Lists"](#) on page 45-7

["GET\\_MODEL\\_TRANSFORMATIONS Function"](#) on page 44-95

["CREATE\\_MODEL Procedure"](#) on page 44-50

## Syntax

```
DBMS_DATA_MINING.GET_TRANSFORM_LIST (
    xform_list          OUT NOCOPY TRANSFORM_LIST,
    model_xforms       IN  DM_TRANSFORMS);
```

## Parameters

**Table 44-77** *GET\_TRANSFORM\_LIST Procedure Parameters*

Parameter	Description										
<code>xform_list</code>	<p>A list of transformation specifications that can be embedded in a model. Accepted as a parameter to the <a href="#">CREATE_MODEL Procedure</a>.</p> <p>The <code>TRANSFORM_LIST</code> type is a table of <code>TRANSFORM_REC</code> objects. Each <code>TRANSFORM_REC</code> has these fields:</p> <table border="0"> <tr> <td><code>attribute_name</code></td> <td><code>VARCHAR2(30)</code></td> </tr> <tr> <td><code>attribute_subname</code></td> <td><code>VARCHAR2(4000)</code></td> </tr> <tr> <td><code>expression</code></td> <td><code>EXPRESSION_REC</code></td> </tr> <tr> <td><code>reverse_expression</code></td> <td><code>EXPRESSION_REC</code></td> </tr> <tr> <td><code>attribute_spec</code></td> <td><code>VARCHAR2(4000)</code></td> </tr> </table> <p>For details about the <code>TRANSFORM_LIST</code> collection type, see <a href="#">Table 45-1, "Data Types in DBMS_DATA_MINING_TRANSFORM"</a>.</p>	<code>attribute_name</code>	<code>VARCHAR2(30)</code>	<code>attribute_subname</code>	<code>VARCHAR2(4000)</code>	<code>expression</code>	<code>EXPRESSION_REC</code>	<code>reverse_expression</code>	<code>EXPRESSION_REC</code>	<code>attribute_spec</code>	<code>VARCHAR2(4000)</code>
<code>attribute_name</code>	<code>VARCHAR2(30)</code>										
<code>attribute_subname</code>	<code>VARCHAR2(4000)</code>										
<code>expression</code>	<code>EXPRESSION_REC</code>										
<code>reverse_expression</code>	<code>EXPRESSION_REC</code>										
<code>attribute_spec</code>	<code>VARCHAR2(4000)</code>										
<code>model_xforms</code>	<p>A list of embedded transformation expressions returned by the <a href="#">GET_MODEL_TRANSFORMATIONS Function</a> for a specific model.</p> <p>The <code>DM_TRANSFORMS</code> type is a table of <code>DM_TRANSFORM</code> objects. Each <code>DM_TRANSFORM</code> has these fields:</p> <table border="0"> <tr> <td><code>attribute_name</code></td> <td><code>VARCHAR2(4000)</code></td> </tr> <tr> <td><code>attribute_subname</code></td> <td><code>VARCHAR2(4000)</code></td> </tr> <tr> <td><code>expression</code></td> <td><code>CLOB</code></td> </tr> <tr> <td><code>reverse_expression</code></td> <td><code>CLOB</code></td> </tr> </table>	<code>attribute_name</code>	<code>VARCHAR2(4000)</code>	<code>attribute_subname</code>	<code>VARCHAR2(4000)</code>	<code>expression</code>	<code>CLOB</code>	<code>reverse_expression</code>	<code>CLOB</code>		
<code>attribute_name</code>	<code>VARCHAR2(4000)</code>										
<code>attribute_subname</code>	<code>VARCHAR2(4000)</code>										
<code>expression</code>	<code>CLOB</code>										
<code>reverse_expression</code>	<code>CLOB</code>										

## Examples

In this example, a model `mod1` is trained using several columns in the `SH.CUSTOMERS` table. The model uses ADP, which automatically bins one of the columns.

A second model `mod2` is trained on the same data without ADP, but it uses a transformation list that was obtained from `mod1`. As a result, both `mod1` and `mod2` have the same embedded transformation expression.

## GET\_TRANSFORM\_LIST Procedure

---

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_income_level, cust_credit_limit
  FROM sh.customers;

describe mining_data
Name                                     Null?   Type
-----
CUST_ID                                 NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                       NOT NULL NUMBER(4)
CUST_INCOME_LEVEL                         VARCHAR2(30)
CUST_CREDIT_LIMIT                         NUMBER

CREATE TABLE setmod1(setting_name VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
  INSERT INTO setmod1 VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
  INSERT INTO setmod1 VALUES (dbms_data_mining.prep_auto,dbms_data_mining.prep_auto_on);
  dbms_data_mining.CREATE_MODEL (
    model_name           => 'mod1',
    mining_function      => dbms_data_mining.classification,
    data_table_name     => 'mining_data',
    case_id_column_name => 'cust_id',
    target_column_name  => 'cust_income_level',
    settings_table_name => 'setmod1');
  COMMIT;
END;
/
CREATE TABLE setmod2(setting_name VARCHAR2(30),setting_value VARCHAR2(30));
BEGIN
  INSERT INTO setmod2
    VALUES (dbms_data_mining.algo_name, dbms_data_mining.algo_naive_bayes);
  COMMIT;
END;
/
DECLARE
  v_xform_list      dbms_data_mining_transform.TRANSFORM_LIST;
  dmxf              DM_TRANSFORMS;
BEGIN
  EXECUTE IMMEDIATE
    'SELECT dm_transform(attribute_name, attribute_subname,expression, reverse_expression)
     FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS (''mod1''))'
    BULK COLLECT INTO dmxf;
  dbms_data_mining.GET_TRANSFORM_LIST (
    xform_list       => v_xform_list,
    model_xforms     => dmxf);
  dbms_data_mining.CREATE_MODEL(
    model_name       => 'mod2',
    mining_function  => dbms_data_mining.classification,
    data_table_name  => 'mining_data',
    case_id_column_name => 'cust_id',
    target_column_name => 'cust_income_level',
    settings_table_name => 'setmod2',
    xform_list       => v_xform_list);
END;
/

-- Transformation expression embedded in mod1
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));

TO_CHAR(EXPRESSION)
-----
```

```

CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
.
.
.
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

```

**-- Transformation expression embedded in mod2**

```
SELECT TO_CHAR(expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));
```

```
TO_CHAR(EXPRESSION)
```

```

-----
CASE WHEN "CUST_YEAR_OF_BIRTH"<1915 THEN 0 WHEN "CUST_YEAR_OF_BIRTH"<=1915 THEN 0
WHEN "CUST_YEAR_OF_BIRTH"<=1920.5 THEN 1 WHEN "CUST_YEAR_OF_BIRTH"<=1924.5 THEN 2
.
.
.
.5 THEN 29 WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN 30 END

```

**-- Reverse transformation expression embedded in mod1**

```
SELECT TO_CHAR(reverse_expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod1'));
```

```
TO_CHAR(REVERSE_EXPRESSION)
```

```

-----
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915)', [1915; 1915]',1,'(1915; 1920.5]',2,'(1
920.5; 1924.5]',3,'(1924.5; 1928.5]',4,'(1928.5; 1932.5]',5,'(1932.5; 1936.5]',6
.
.
.
8,'(1987.5; 1988.5]',29,'(1988.5; 1989.5]',30,'(1989.5; )',NULL,'NULL')

```

**-- Reverse transformation expression embedded in mod2**

```
SELECT TO_CHAR(reverse_expression) FROM TABLE (dbms_data_mining.GET_MODEL_TRANSFORMATIONS('mod2'));
```

```
TO_CHAR(REVERSE_EXPRESSION)
```

```

-----
DECODE("CUST_YEAR_OF_BIRTH",0,'( ; 1915)', [1915; 1915]',1,'(1915; 1920.5]',2,'(1
920.5; 1924.5]',3,'(1924.5; 1928.5]',4,'(1928.5; 1932.5]',5,'(1932.5; 1936.5]',6
.
.
.
8,'(1987.5; 1988.5]',29,'(1988.5; 1989.5]',30,'(1989.5; )',NULL,'NULL')

```

## IMPORT\_MODEL Procedure

This procedure imports one or more data mining models. The procedure is overloaded. You can call it to import mining models from a dump file set, or you can call it to import a single mining model from a PMML document.

### Import from a dump file set

You can import mining models from a dump file set that was created by the [EXPORT\\_MODEL Procedure](#). `IMPORT_MODEL` and `EXPORT_MODEL` use Oracle Data Pump technology to export to and import from a dump file set.

When Oracle Data Pump is used directly to export/import an entire schema or database, the mining models in the schema or database are included. `EXPORT_MODEL` and `IMPORT_MODEL` export/import mining models only.

### Import from PMML

This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2) Data Mining.

You can import a mining model represented in Predictive Model Markup Language (PMML). The model must be of type `RegressionModel`, either linear regression or binary logistic regression.

PMML is an XML-based standard specified by the Data Mining Group (<http://www.dmg.org>). Applications that are PMML-compliant can deploy PMML-compliant models that were created by any vendor. Oracle Data Mining supports the core features of PMML 3.1 for regression models.

#### See Also:

*Oracle Data Mining Administrator's Guide* for more information about exporting and importing mining models

*Oracle Database Utilities* for information about Oracle Data Pump

<http://www.dmg.org/faq.html> for more information about PMML

## Syntax

Imports a mining model from a dump file set:

```
DBMS_DATA_MINING.IMPORT_MODEL (
    filename      IN  VARCHAR2,
    directory     IN  VARCHAR2,
    model_filter  IN  VARCHAR2 DEFAULT NULL,
    operation     IN  VARCHAR2 DEFAULT NULL,
    remote_link   IN  VARCHAR2 DEFAULT NULL,
    jobname       IN  VARCHAR2 DEFAULT NULL,
    schema_remap IN  VARCHAR2 DEFAULT NULL);
```

Imports a mining model from a PMML document:

```
DBMS_DATA_MINING.IMPORT_MODEL (
    model_name     IN  VARCHAR2,
    pmml_doc       IN  XMLTYPE);
```



## Parameters

**Table 44–78** *IMPORT\_MODEL Procedure Parameters*

Parameter	Description
filename	<p>Name of the dump file set from which the models should be imported. The dump file set must have been created by the EXPORT_MODEL procedure or the expdp export utility of Oracle Data Pump.</p> <p>The dump file set can contain one or more files. (Refer to "EXPORT_MODEL Procedure" on page 44-55 for details.) If the dump file set contains multiple files, you can specify 'filename%U' instead of listing them. For example, if your dump file set contains 3 files, archive01.dmp, archive02.dmp, and archive03.dmp, you can import them by specifying 'archive%U'.</p>
directory	<p>Name of a pre-defined directory object that specifies where the dump file set is located. Both the exporting and the importing user must have read/write access to the directory object and to the file system directory that it identifies.</p> <p>Note: The target database must have also have read/write access to the file system directory.</p>
model_filter	<p>Optional parameter that specifies one or more models to import. If you do not specify a value for model_filter, all models in the dump file set are imported. You can also specify NULL (the default) or 'ALL' to import all models.</p> <p>The value of model_filter can be one or more model names. The following are valid filters.</p> <pre>'mymodel1'</pre> <pre>'name IN ('mymodel2','mymodel3')'</pre> <p>The first causes IMPORT_MODEL to import a single model named mymodel1. The second causes IMPORT_MODEL to import two models, mymodel2 and mymodel3.</p>
operation	<p>Optional parameter that specifies whether to import the models or the SQL statements that create the models. By default, the models are imported.</p> <p>You can specify either of the following values for operation:</p> <ul style="list-style-type: none"> <li>▪ 'IMPORT' — Import the models (Default)</li> <li>▪ 'SQL_FILE' — Write the SQL DDL for creating the models to a text file. The text file is named <i>job_name.sql</i> and is located in the dump set directory.</li> </ul>
remote_link	Optional parameter not used in this release. Set to NULL
jobname	<p>Optional parameter that specifies the name of the import job. By default, the name has the form <i>username_imp_nnnn</i>, where <i>nnnn</i> is a number. For example, a job name in the SCOTT schema might be SCOTT_imp_134.</p> <p>If you specify a job name, it must be unique within the schema. The maximum length of the job name is 30 characters.</p> <p>A log file for the import job, named <i>jobname.log</i>, is created in the same directory as the dump file set.</p>
schema_remap	<p>Optional parameter for importing into a different schema. By default, models are exported and imported within the same schema.</p> <p>If the dump file set belongs to a different schema, you must specify a schema mapping in the form <i>export_user:import_user</i>. For example, you would specify 'SCOTT:MARY' to import a model exported by SCOTT into the MARY schema.</p> <p>Note: In some cases, you may need to have the IMP_FULL_DATABASE privilege or the SYS role to import a model from a different schema.</p>

**Table 44–78 (Cont.) IMPORT\_MODEL Procedure Parameters**

Parameter	Description
model_name	Name for the new model that will be created in the database as a result of an import from PMML. The name must be unique within the user's schema.
pmml.doc	The PMML document representing the model to be imported. The PMML document has an XMLTYPE object type. See <a href="#">Chapter 259, "XMLTYPE"</a> for details.

## Usage Notes

The following notes pertain to mining model import based on Oracle Data Pump.

Mining models are stored in the default tablespace of the mining model owner, or in a tablespace to which the owner has access. The tablespace must also exist in the target database, and the target user must have access to it. If the tablespace does not exist in the target database, you must create it before importing the models.

For example, if the models were created in schema DMUSER and the default tablespace for DMUSER is USERS, then the USERS tablespace must exist in the target database. You can create the USERS tablespace and grant access to a target user with appropriate tablespace quota as follows.

```
connect / as sysdba;
create tablespace USERS datafile 'data_file_name' size 200M autoextend on;
alter user target_user quota unlimited on USERS;
```

## Examples

1. This example shows a model being exported and imported within the schema dmuser2. Then the same model is imported into the dmuser3 schema. The dmuser3 user has the IMP\_FULL\_DATABASE privilege.

```
SQL> connect dmuser2
Enter password: dmuser2_password
Connected.
SQL> select model_name from user_mining_models;

MODEL_NAME
-----
NMF_SH_SAMPLE
SVMO_SH_CLAS_SAMPLE
SVMR_SH_REGR_SAMPLE

-- export the model called NMF_SH_SAMPLE to a dump file in same schema
SQL>EXECUTE DBMS_DATA_MINING.EXPORT_MODEL ('NMF_SH_SAMPLE_out',
'DATA_PUMP_DIR',
                                     'name = 'NMF_SH_SAMPLE''');
-- import the model back into the same schema
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL ('NMF_SH_SAMPLE_out01.dmp',
'DATA_PUMP_DIR', 'name = 'NMF_SH_SAMPLE''');

-- connect as different user
-- import same model into that schema
SQL> connect dmuser3
Enter password: dmuser3_password
Connected.
SQL>EXECUTE DBMS_DATA_MINING.IMPORT_MODEL ('NMF_SH_SAMPLE_out01.dmp',
'DATA_PUMP_DIR', 'name = 'NMF_SH_SAMPLE'',
'IMPORT', NULL, 'nmf_imp_job', 'dmuser2:dmuser3');
```

The following example shows user MARY importing all models from a dump file, `model_exp_001.dmp`, which was created by user SCOTT. The dump file is located in the file system directory mapped to a directory object called `DM_DUMP`. If user MARY does not have `IMP_FULL_DATABASE` privileges, `IMPORT_MODEL` will raise an error.

```
-- import all models
DECLARE
  file_name      VARCHAR2(40);
BEGIN
  file_name := 'model_exp_001.dmp';
  DBMS_DATA_MINING.IMPORT_MODEL(
    filename=>file_name,
    directory=>'DM_DUMP',
    schema_remap=>'SCOTT:MARY');
  DBMS_OUTPUT.PUT_LINE(
'DBMS_DATA_MINING.IMPORT_MODEL of all models from SCOTT done!');
END;
/
```

2. This example shows how a PMML document called `SamplePMML1.xml` could be imported from a location referenced by directory object `PMMLDIR` into the schema of the current user. The imported model will be called `PMMLMODEL1`.

```
BEGIN
  dbms_data_mining.import_model ('PMMLMODEL1',
    XMLType (bfilename ('PMMLDIR', 'SamplePMML1.xml'),
    nls_charset_id ('AL32UTF8')
  ));
END;
```

## RANK\_APPLY Procedure

This procedure ranks the results of an `APPLY` operation based on a top-N specification for predictive and descriptive model results. For classification models, you can provide a cost matrix as input, and obtain the ranked results with costs applied to the predictions.

### Syntax

```
DBMS_DATA_MINING.RANK_APPLY (
    apply_result_table_name      IN VARCHAR2,
    case_id_column_name         IN VARCHAR2,
    score_column_name           IN VARCHAR2,
    score_criterion_column_name IN VARCHAR2,
    ranked_apply_table_name     IN VARCHAR2,
    top_N                        IN INTEGER DEFAULT 1,
    cost_matrix_table_name      IN VARCHAR2 DEFAULT NULL,
    apply_result_schema_name    IN VARCHAR2 DEFAULT NULL,
    cost_matrix_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 44–79 RANK\_APPLY Procedure Parameters**

Parameter	Description
<code>apply_result_table_name</code>	Name of the table or view containing the results of an <code>APPLY</code> operation on the test dataset (see Usage Notes)
<code>case_id_column_name</code>	Name of the case identifier column. This must be the same as the one used for generating <code>APPLY</code> results.
<code>score_column_name</code>	Name of the prediction column in the apply results table
<code>score_criterion_column_name</code>	Name of the probability column in the apply results table
<code>ranked_apply_result_table_name</code>	Name of the table containing the ranked apply results
<code>top_N</code>	Top N predictions to be considered from the <code>APPLY</code> results for precision recall computation
<code>cost_matrix_table_name</code>	Name of the cost matrix table
<code>apply_result_schema_name</code>	Name of the schema hosting the <code>APPLY</code> results table
<code>cost_matrix_schema_name</code>	Name of the schema hosting the cost matrix table

### Usage Notes

You can use `RANK_APPLY` to generate ranked apply results, based on a top-N filter and also with application of cost for predictions, if the model was built with costs.

The behavior of `RANK_APPLY` is similar to that of `APPLY` with respect to other DDL-like operations such as `CREATE_MODEL`, `DROP_MODEL`, and `RENAME_MODEL`. The procedure does not depend on the model; the only input of relevance is the apply results generated in a fixed schema table from `APPLY`.

The main intended use of `RANK_APPLY` is for the generation of the final `APPLY` results against the scoring data in a production setting. You can apply the model against test

data using `APPLY`, compute various test metrics against various cost matrix tables, and use the candidate cost matrix for `RANK_APPLY`.

The schema for the apply results from each of the supported algorithms is listed in subsequent sections. The `case_id` column will be the same case identifier column as that of the apply results.

### Classification Models — NB, ABN, SVM

For numerical targets, the ranked results table will have the definition as shown:

```
(case_id      VARCHAR2/NUMBER,
prediction    NUMBER,
probability   NUMBER,
cost         NUMBER,
rank         INTEGER)
```

For categorical targets, the ranked results table will have the following definition:

```
(case_id      VARCHAR2/NUMBER,
prediction    VARCHAR2,
probability   NUMBER,
cost         NUMBER,
rank         INTEGER)
```

### Clustering using *k*-Means or O-Cluster

Clustering is an unsupervised mining function, and hence there are no targets. The results of an `APPLY` operation contains simply the cluster identifier corresponding to a case, and the associated probability. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the cluster ids ranked by top-N.

```
(case_id      VARCHAR2/NUMBER,
cluster_id    NUMBER,
probability   NUMBER,
rank         INTEGER)
```

### Feature Extraction using NMF

Feature extraction is also an unsupervised mining function, and hence there are no targets. The results of an `APPLY` operation contains simply the feature identifier corresponding to a case, and the associated match quality. Cost matrix is not considered here. The ranked results table will have the definition as shown, and contains the feature ids ranked by top-N.

```
(case_id      VARCHAR2/NUMBER,
feature_id    NUMBER,
match_quality NUMBER,
rank         INTEGER)
```

### Examples

```
BEGIN
/* build a model with name census_model.
 * (See example under CREATE_MODEL)
 */

/* if training data was pre-processed in any manner,
 * perform the same pre-processing steps on apply
 * data also.
 * (See examples in the section on DBMS_DATA_MINING_TRANSFORM)
```

```
*/

/* apply the model to data to be scored */
DBMS_DATA_MINING.RANK_APPLY(
  apply_result_table_name      => 'census_apply_result',
  case_id_column_name         => 'person_id',
  score_column_name           => 'prediction',
  score_criterion_column_name => 'probability',
  ranked_apply_result_tab_name => 'census_ranked_apply_result',
  top_N                       => 3,
  cost_matrix_table_name      => 'census_cost_matrix');
END;
/

-- View Ranked Apply Results
SELECT *
  FROM census_ranked_apply_result;
```

## REMOVE\_COST\_MATRIX Procedure

Removes the default scoring matrix from a classification model.

### See Also:

- ["ADD\\_COST\\_MATRIX Procedure"](#) on page 44-25
- ["REMOVE\\_COST\\_MATRIX Procedure"](#) on page 44-107

### Syntax

```
DBMS_DATA_MINING.REMOVE_COST_MATRIX (
    model_name IN VARCHAR2);
```

### Parameters

**Table 44–80 Remove\_Cost\_Matrix Procedure Parameters**

Parameter	Description
model_name	Name of the model in the form [schema_name.]model_name. If you do not specify a schema, your own schema is used.

### Usage Notes

If the model is not in your schema, then REMOVE\_COST\_MATRIX requires the ALTER ANY MINING MODEL system privilege or the ALTER object privilege for the mining model.

### Example

The Naive Bayes model NB\_SH\_CLAS\_SAMPLE has an associated cost matrix that can be used for scoring the model.

```
SQL>SELECT *
      FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
      ORDER BY predicted, actual;
```

```
ACTUAL      PREDICTED      COST
-----
0           0              0
1           0              .75
0           1              .25
1           1              0
```

You can remove the cost matrix with REMOVE\_COST\_MATRIX.

```
SQL>EXECUTE dbms_data_mining.remove_cost_matrix('nb_sh_clas_sample');
```

```
SQL>SELECT *
      FROM TABLE(dbms_data_mining.get_model_cost_matrix('nb_sh_clas_sample'))
      ORDER BY predicted, actual;
```

```
no rows selected
```

## RENAME\_MODEL Procedure

This procedure renames a mining model to a new name that you specify.

The model name is in the form `[schema_name.]model_name`. If you do not specify a schema, your own schema is used. For mining model naming restrictions, see ["Mining Model Naming Restrictions"](#) on page 44-5.

### Syntax

```
DBMS_DATA_MINING.RENAME_MODEL (
    model_name          IN VARCHAR2,
    new_model_name      IN VARCHAR2);
```

### Parameters

**Table 44–81** *RENAME\_MODEL Procedure Parameters*

Parameter	Description
<code>model_name</code>	Old name of the model
<code>new_model_name</code>	New name of the model.

### Usage Notes

If an APPLY operation is using a model, and you attempt to rename the model during that time, the RENAME will succeed and APPLY will return indeterminate results.

### Examples

Assume the existence of a model `census_model`. The following example shows how to rename this model.

```
BEGIN
    DBMS_DATA_MINING.RENAME_MODEL(
        model_name      => 'census_model',
        new_model_name  => 'census_new_model');
END;
/
```



---

---

## DBMS\_DATA\_MINING\_TRANSFORM

DBMS\_DATA\_MINING\_TRANSFORM implements a set of transformations that are commonly used in data mining and provides a framework that you can use for specifying your own transformations.

**See Also:**

- [Chapter 44, "DBMS\\_DATA\\_MINING"](#)
- *Oracle Data Mining Concepts*

This chapter contains the following topics:

- [Using DBMS\\_DATA\\_MINING\\_TRANSFORM](#)
  - Overview
  - Operational Notes
  - Security Model
  - Types
  - Constants
- [Summary of DBMS\\_DATA\\_MINING\\_TRANSFORM Subprograms](#)

---

## Using DBMS\_DATA\_MINING\_TRANSFORM

This section contains topics that relate to using the DBMS\_DATA\_MINING\_TRANSFORM package.

- [Overview](#)
- [Operational Notes](#)
- [Security Model](#)
- [Types](#)
- [Constants](#)

## Overview

A transformation is a SQL expression that modifies the data in one or more columns.

Data must typically undergo certain transformations before it can be used to build a mining model. Many data mining algorithms have specific transformation requirements.

Data that will be scored must be transformed in the same way as the data that was used to create (train) the model.

## External or Embedded Transformations

DBMS\_DATA\_MINING\_TRANSFORM offers two approaches to implementing transformations. For a given model, you can either:

- Create a list of transformation expressions and pass it to the [CREATE\\_MODEL Procedure](#)

*or*

- Create a view that implements the transformations and pass the name of the view to the [CREATE\\_MODEL Procedure](#)

If you create a transformation list and pass it to `CREATE_MODEL`, the transformation expressions are embedded in the model and automatically implemented whenever the model is applied.

If you create a view, the transformation expressions are external to the model. You will need to re-create the transformations whenever you apply the model.

---

---

**Note:** Embedded transformations significantly enhance the model's usability while simplifying the process of model management.

---

---

## Automatic Transformations

Oracle Data Mining supports an Automatic Data Preparation (ADP) mode. When ADP is enabled, most algorithm-specific transformations are *automatically* embedded. Any additional transformations must be explicitly provided in an embedded transformation list or in a view.

If ADP is enabled and you create a model with a transformation list, both sets of transformations are embedded. The model will execute the user-specified transformations from the transformation list before executing the automatic transformations specified by ADP.

Within a transformation list, you can selectively disable ADP for individual attributes.

**See Also:**

["Automatic Data Preparation"](#) on page 44-11

*Oracle Data Mining Concepts* for a list of the algorithm-specific transformations implemented by ADP

["About Transformation Lists"](#) on page 45-7

## Transformations in DBMS\_DATA\_MINING\_TRANSFORM

The transformations supported by `DBMS_DATA_MINING_TRANSFORM` are summarized in this section.

## Binning

Binning refers to the mapping of continuous or discrete values to discrete values of reduced cardinality.

- **Supervised Binning (Categorical and Numerical)**  
Binning is based on intrinsic relationships in the data as determined by a decision tree model.  
See ["INSERT\\_BIN\\_SUPER Procedure"](#) on page 45-51.
- **Top-N Frequency Categorical Binning**  
Binning is based on the number of cases in each category.  
See ["INSERT\\_BIN\\_CAT\\_FREQ Procedure"](#) on page 45-39
- **Equi-Width Numerical Binning**  
Binning is based on equal-range partitions.  
See ["INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure"](#) on page 45-44.
- **Quantile Numerical Binning**  
Binning is based on quantiles computed using the SQL `NTILE` function.  
See ["INSERT\\_BIN\\_NUM\\_QTILE Procedure"](#) on page 45-48.

## Linear Normalization

Normalization is the process of scaling continuous values down to a specific range, often between zero and one. Normalization transforms each numerical value by subtracting a number (the **shift**) and dividing the result by another number (the **scale**).

$$x_{\text{new}} = (x_{\text{old}} - \text{shift}) / \text{scale}$$

- **Min-Max Normalization**  
Normalization is based on the minimum and maximum with the following shift and scale:  

```
shift = min  
scale = max-min
```

  
See ["INSERT\\_NORM\\_LIN\\_MINMAX Procedure"](#) on page 45-67.
- **Scale Normalization**  
Normalization is based on the minimum and maximum with the following shift and scale:  

```
shift = 0  
scale = max{abs(max), abs(min)}
```

  
See ["INSERT\\_NORM\\_LIN\\_SCALE Procedure"](#) on page 45-70.
- **Z-Score Normalization**  
Normalization is based on the mean and standard deviation with the following shift and scale:  

```
shift = mean  
scale = standard_deviation
```

  
See ["INSERT\\_NORM\\_LIN\\_ZSCORE Procedure"](#) on page 45-73.

**Outlier Treatment**

An outlier is a numerical value that is located far from the rest of the data. Outliers can artificially skew the results of data mining.

- Winsorizing

Outliers are replaced with the nearest value that is not an outlier.

See "[INSERT\\_CLIP\\_WINSOR\\_TAIL Procedure](#)" on page 45-58

- Trimming

Outliers are set to NULL.

See "[INSERT\\_CLIP\\_TRIM\\_TAIL Procedure](#)" on page 45-55.

**Missing Value Treatment**

Missing data may indicate sparsity or it may indicate that some values are missing at random. DBMS\_DATA\_MINING\_TRANSFORM supports the following transformations for minimizing the effects of missing values:

- Missing numerical values are replaced with the mean.

See "[INSERT\\_MISS\\_NUM\\_MEAN Procedure](#)" on page 45-64.

- Missing categorical values are replaced with the mode.

See "[INSERT\\_MISS\\_CAT\\_MODE Procedure](#)" on page 45-61.

---

---

**Note:** Oracle Data Mining also has default mechanisms for handling missing data. See *Oracle Data Mining Application Developer's Guide* for details.

---

---

## Operational Notes

The `DBMS_DATA_MINING_TRANSFORM` package offers a flexible framework for specifying data transformations. If you choose to embed transformations in the model (the preferred method), you will create a **transformation list** object and pass it to the [CREATE\\_MODEL Procedure](#). If you choose to transform the data without embedding, you will create a view.

When specified in a transformation list, the transformation expressions are executed by the model. When specified in a view, the transformation expressions are executed by the view.

### Transformation Definitions

Transformation definitions are used to generate the SQL expressions that transform the data. For example, the transformation definitions for normalizing a numeric column are the shift and scale values for that data.

With the `DBMS_DATA_MINING_TRANSFORM` package, you can call procedures to compute the transformation definitions, or you can compute them yourself, or you can do both.

### Transformation Definition Tables

`DBMS_DATA_MINING_TRANSFORM` provides **INSERT** procedures that compute transformation definitions and insert them in transformation definition tables. You can modify the values in the transformation definition tables or populate them yourself.

**XFORM** routines use populated definition tables to transform data in external views.

**STACK** routines use populated definition tables to build transformation lists.

**To specify transformations based on definition tables, follow these steps:**

1. Use **CREATE** routines to create transformation definition tables.

The tables have columns to hold the transformation definitions for a given type of transformation. For example, the [CREATE\\_BIN\\_NUM Procedure](#) creates a definition table that has a column for storing data values and another column for storing the associated bin identifiers.

2. Use **INSERT** routines to compute and insert transformation definitions in the tables.

Each **INSERT** routine uses a specific technique for computing the transformation definitions. For example, the [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) computes bin boundaries by identifying the minimum and maximum values then setting the bin boundaries at equal intervals.

3. Use **STACK** or **XFORM** routines to generate transformation expressions based on the information in the definition tables:
  - Use **STACK** routines to add the transformation expressions to a transformation list. Pass the transformation list to the [CREATE\\_MODEL Procedure](#). The transformation expressions will be assembled into one long SQL query and embedded in the model.
  - Use **XFORM** routines to execute the transformation expressions within a view. The transformations will be external to the model and will need to be re-created whenever the model is applied to new data.

## Transformations Without Definition Tables

STACK routines are not the only method for adding transformation expressions to a transformation list. You can also build a transformation list without using definition tables.

To specify transformations without using definition tables, follow these steps:

1. Write a SQL expression for transforming an attribute.
2. Write a SQL expression for reversing the transformation. (See ["Reverse Transformations and Model Transparency"](#) on page 45-7.)
3. Determine whether or not to disable ADP for the attribute. By default ADP is enabled for the attribute if it is specified for the model. (See ["Disabling Automatic Data Preparation"](#) on page 45-8.)
4. Specify the SQL expressions and ADP instructions in a call to the [SET\\_TRANSFORM Procedure](#), which adds the information to a transformation list.
5. Repeat steps 1 through 4 for each attribute that you wish to transform.
6. Pass the transformation list to the [CREATE\\_MODEL Procedure](#). The transformation expressions will be assembled into one long SQL query and embedded in the model.

---

**Note:** SQL expressions that you specify with `SET_TRANSFORM` must fit within a `VARCHAR2`. To specify a longer expression, you can use the [SET\\_EXPRESSION Procedure](#). With `SET_EXPRESSION`, you can build an expression by appending rows to a `VARCHAR2` array.

---

## About Transformation Lists

The elements of a transformation list are **transformation records**. Each transformation record provides all the information needed by the model for managing the transformation of a single attribute.

Each transformation record includes the following fields:

- *attribute\_name* — Name of the column of data to be transformed
- *attribute\_subname* — Name of the nested attribute if *attribute\_name* is a nested column, otherwise NULL
- *expression* — SQL expression for transforming the attribute
- *reverse\_expression* — SQL expression for reversing the transformation
- *attribute\_spec* — When set to `NOPREP`, disables ADP for the attribute

**See Also:** [Table 45–1](#) for details about the `TRANSFORM_LIST` and `TRANSFORM_REC` object types

## Reverse Transformations and Model Transparency

An algorithm manipulates transformed attributes to train and score a model. The transformed attributes, however, may not be meaningful to an end user. For example, if attribute *x* has been transformed into bins 1 — 4, the bin names 1, 2, 3, and 4 are manipulated by the algorithm, but a user is probably not interested in the model details about bins 1 — 4 or in predicting the numbers 1 — 4.

To return original attribute values in model details and predictions, you can provide a reverse expression in the transformation record for the attribute. For example, if you specify the transformation expression '`log(10, y)`' for attribute *y*, you could specify the reverse transformation expression '`power(10, y)`'.

Reverse transformations enable **model transparency**. They make internal processing transparent to the user.

---

---

**Note:** `STACK` procedures automatically reverse normalization transformations, but they do not provide a mechanism for reversing binning, clipping, or missing value transformations.

You can use the `DBMS_DATA_MINING.ALTER_REVERSE_EXPRESSION` procedure to specify or update reverse transformations expressions for an existing model.

---

---

**See Also:**

[Example 45-1, "Stacking a Clipping Transformation"](#)

["ALTER\\_REVERSE\\_EXPRESSION Procedure"](#) on page 44-27

["Summary of DBMS\\_DATA\\_MINING Subprograms"](#) on page 44-23  
for links to the model details functions

**Disabling Automatic Data Preparation**

ADP is controlled by a model-specific setting (`PREP_AUTO`). The `PREP_AUTO` setting affects all model attributes unless you disable it for individual attributes.

If ADP is enabled and you set *attribute\_spec* to `NOPREP`, only the transformations that you specify for that attribute will be evaluated. If ADP is enabled and you do *not* set *attribute\_spec* to `NOPREP`, the automatic transformations will be evaluated *after* the transformations that you specify for the attribute.

If ADP is not enabled for the model, the *attribute\_spec* field of the transformation record is ignored.

**See Also:** ["Automatic Data Preparation"](#) on page 44-11 for information about the `PREP_AUTO` setting

**Adding Transformation Records to a Transformation List**

A transformation list is a stack of transformation records. When a new transformation record is added, it is appended to the top of the stack. (See ["About Stacking"](#) for details.)

When you use `SET_TRANSFORM` to add a transformation record to a transformation list, you can specify values for all the fields in the transformation record.

When you use `STACK` procedures to add transformation records to a transformation list, only the transformation expression field is populated. For normalization transformations, the reverse transformation expression field is also populated.

You can use both `STACK` procedures and `SET_TRANSFORM` to build one transformation list. Each `STACK` procedure call adds transformation records for all the attributes in a specified transformation definition table. Each `SET_TRANSFORM` call adds a transformation record for a single attribute.



## About Stacking

Transformation lists are built by stacking transformation records. Transformation lists are evaluated from bottom to top. Each transformation expression depends on the result of the transformation expression below it in the stack.

### Stack Procedures

STACK procedures create transformation records from the information in transformation definition tables. For example `STACK_BIN_NUM` builds a transformation record for each attribute specified in a definition table for numeric binning. STACK procedures stack the transformation records as follows:

- If an attribute is specified in the definition table but not in the transformation list, the STACK procedure creates a transformation record, computes the reverse transformation (if possible), inserts the transformation and reverse transformation in the transformation record, and appends the transformation record to the top of the transformation list.
- If an attribute is specified in the transformation list but not in the definition table, the STACK procedure takes no action.
- If an attribute is specified in the definition table *and* in the transformation list, the STACK procedure stacks the transformation expression from the definition table on top of the transformation expression in the transformation record and updates the reverse transformation. See [Example 45-1, "Stacking a Clipping Transformation"](#) and [Example 45-4, "Stacking a Nested Normalization Transformation"](#).

#### Example 45-1 Stacking a Clipping Transformation

This example shows how `STACK_CLIP Procedure` would add transformation records to a transformation list. Note that the clipping transformations are not reversed in COL1 and COL2 after stacking (as described in ["Reverse Transformations and Model Transparency"](#) on page 45-7).

#### Refer to:

- [CREATE\\_CLIP Procedure](#) — Creates the definition table
- [INSERT\\_CLIP\\_TRIM\\_TAIL Procedure](#) — Inserts definitions in the table
- [INSERT\\_CLIP\\_WINSOR\\_TAIL Procedure](#) — Inserts definitions in the table
- [Table 45-1](#) — Describes the structure of the transformation list (TRANSFORM\_LIST object)

Assume a clipping definition table populated as follows.

col	att	lcut	lval	rcut	rval
COL1	null	-1.5	-1.5	4.5	4.5
COL2	null	0	0	1	1

Assume the following transformation list before stacking.

```

-----
transformation record #1:
-----
    attribute_name = COL1
    attribute_subname = null
    expression = log(10, COL1)

```

```
reverse_expression = power(10, COL1)
-----
transformation record #2:
-----
attribute_name      = COL3
attribute_subname   = null
expression          = ln(COL3)
reverse_expression  = exp(COL3)
```

**After stacking, the transformation list is as follows.**

```
-----
transformation record #1:
-----
attribute_name      = COL1
attribute_subname   = null
expression          = CASE WHEN log(10, COL1) < -1.5 THEN -1.5
                    WHEN log(10, COL1) > 4.5 THEN 4.5
                    ELSE log(10, COL1)
                    END;
reverse_expression  = power(10, COL1)
-----
transformation record #2:
-----
attribute_name      = COL3
attribute_subname   = null
expression          = ln(COL3)
reverse_expression  = exp(COL3)
-----
transformation record #3:
-----
attribute_name      = COL2
attribute_subname   = null
expression          = CASE WHEN COL2 < 0 THEN 0
                    WHEN COL2 > 1 THEN 1
                    ELSE COL2
                    END;
reverse_expression  = null
```

## Nested Data Transformations

The CREATE routines create transformation definition tables that include two columns, `col` and `att`, for identifying attributes. The column `col` holds the name of a column in the data table. If the data column is not nested, then `att` is null, and the name of the attribute is `col`. If the data column is nested, then `att` holds the name of the nested attribute, and the name of the attribute is `col.att`.

The INSERT and XFORM routines ignore the `att` column in the definition tables. Neither the INSERT nor the XFORM routines support nested data.

Only the STACK procedures and SET\_TRANSFORM support nested data. Nested data transformations are always embedded in the model.

Nested columns in Oracle Data Mining can have either of the following types:

```
DM_NESTED_NUMERICALS
DM_NESTED_CATEGORICALS
```

**See Also:**

"Constants" on page 45-16

*Oracle Data Mining Application Developer's Guide* for details about nested attributes in Oracle Data Mining

**Specifying Nested Attributes in a Transformation Record**

A transformation record (TRANSFORM\_REC) includes two fields, `attribute_name` and `attribute_subname`, for identifying the attribute. The field `attribute_name` holds the name of a column in the data table. If the data column is not nested, then `attribute_subname` is null, and the name of the attribute is `attribute_name`. If the data column is nested, then `attribute_subname` holds the name of the nested attribute, and the name of the attribute is `attribute_name.attribute_subname`.

**Transforming Individual Nested Attributes**

You can specify different transformations for different attributes in a nested column, and you can specify a default transformation for all the remaining attributes in the column. To specify a default nested transformation, specify null in the `attribute_name` field and the name of the nested column in the `attribute_subname` field as shown in [Example 45-2](#). Note that the keyword `VALUE` is used to represent the value of a nested attribute in a transformation expression.

**Example 45-2 Transforming a Nested Column**

The following statement transforms two of the nested attributes in `COL_N1`. Attribute `ATTR1` is transformed with normalization; Attribute `ATTR2` is set to null, which causes attribute removal transformation (`ATTR2` is not used in training the model). All the remaining attributes in `COL_N1` are divided by 10.

```
DECLARE
  stk dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(
    stk,'COL_N1', 'ATTR1', '(VALUE - (-1.5))/20', 'VALUE *20 + (-1.5)');
  dbms_data_mining_transform.SET_TRANSFORM(
    stk,'COL_N1', 'ATTR2', NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM(
    stk, NULL, 'COL_N1', 'VALUE/10', 'VALUE*10');
END;
/
```

The following SQL is generated from this statement.

```
CAST(MULTISET(SELECT DM_NESTED_NUMERICAL(
  "ATTRIBUTE_NAME",
  DECODE("ATTRIBUTE_NAME",
    'ATTR1', ("VALUE" - (-1.5))/20,
    "VALUE"/10))
  FROM TABLE("COL_N1")
  WHERE "ATTRIBUTE_NAME" IS NOT IN ('ATTR2'))
  AS DM_NESTED_NUMERICALS)
```

If transformations are not specified for `COL_N1.ATTR1` and `COL_N1.ATTR2`, then the default transformation is used for all the attributes in `COL_N1`, and the resulting SQL does not include a `DECODE`.

```
CAST(MULTISET(SELECT DM_NESTED_NUMERICAL(
  "ATTRIBUTE_NAME",
  "VALUE"/10)
```

```

FROM TABLE("COL_N1")
AS DM_NESTED_NUMERICALS)

```

Since DECODE is limited to 256 arguments, multiple DECODE functions are nested to support an arbitrary number of individual nested attribute specifications.

### Adding a Nested Column

You can specify a transformation that adds a nested column to the data, as shown in [Example 45-3](#).

#### **Example 45-3 Adding a Nested Column to a Transformation List**

```

DECLARE
  v_xlst dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM(v_xlst,
    'YOB_CREDLIM', NULL,
    'dm_nested_numericals(
      dm_nested_numerical(
        'CUST_YEAR_OF_BIRTH', cust_year_of_birth),
      dm_nested_numerical(
        'CUST_CREDIT_LIMIT', cust_credit_limit))',
    NULL);
  dbms_data_mining_transform.SET_TRANSFORM(
    v_xlst, 'CUST_YEAR_OF_BIRTH', NULL, NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM(
    v_xlst, 'CUST_CREDIT_LIMIT', NULL, NULL, NULL);
  dbms_data_mining_transform.XFORM_STACK(
    v_xlst, 'mining_data', 'mining_data_v');
END;
/

set long 2000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_V';

TEXT
-----
SELECT "CUST_ID", "CUST_POSTAL_CODE", dm_nested_numericals(
  dm_nested_numerical(
    'CUST_YEAR_OF_BIRTH', cust_year_of_birth),
  dm_nested_numerical(
    'CUST_CREDIT_LIMIT', cust_credit_limit)) "YOB_CREDLIM" FROM mining_data

SELECT * FROM mining_data_v WHERE cust_id = 104500;

CUST_ID CUST_POSTAL_CODE YOB_CREDLIM(ATTRIBUTE_NAME, VALUE)
-----
104500 68524             DM_NESTED_NUMERICALS(DM_NESTED_NUMERICAL(
                        'CUST_YEAR_OF_BIRTH', 1962),
                        DM_NESTED_NUMERICAL('CUST_CREDIT_LIMIT', 15000))

```

### Stacking Nested Transformations

[Example 45-4](#) shows how the [STACK\\_NORM\\_LIN Procedure](#) would add transformation records for nested column COL\_N to a transformation list.

#### Refer to:

- [CREATE\\_NORM\\_LIN Procedure](#) — Creates the definition table
- [INSERT\\_NORM\\_LIN\\_MINMAX Procedure](#) — Inserts definitions in the table

- [INSERT\\_NORM\\_LIN\\_SCALE Procedure](#) — Inserts definitions in the table
- [INSERT\\_NORM\\_LIN\\_ZSCORE Procedure](#) — Inserts definitions in the table
- [Table 45-1](#) — Describes the structure of the transformation list

**Example 45-4 Stacking a Nested Normalization Transformation**

Assume a linear normalization definition table populated as follows.

col	att	shift	scale
COL_N	ATT2	0	20
null	COL_N	0	10

Assume the following transformation list before stacking.

```

-----
transformation record #1:
-----
      attribute_name      = COL_N
      attribute_subname   = ATT1
      expression          = log(10, VALUE)
      reverse_expression  = power(10, VALUE)
-----
transformation record #2:
-----
      attribute_name      = null
      attribute_subname   = COL_N
      expression          = ln(VALUE)
      reverse_expression  = exp(VALUE)

```

After stacking, the transformation list is as follows.

```

-----
transformation record #1:
-----
      attribute_name      = COL_N
      attribute_subname   = ATT1
      expression          = (log(10, VALUE) - 0)/10
      reverse_expression  = power(10, VALUE*10 + 0)
-----
transformation record #2:
-----
      attribute_name      = NULL
      attribute_subname   = COL_N
      expression          = (ln(VALUE)- 0)/10
      reverse_expression  = exp(VALUE *10 + 0)
-----
transformation record #3:
-----
      attribute_name      = COL_N
      attribute_subname   = ATT2
      expression          = (ln(VALUE) - 0)/20
      reverse_expression  = exp(VALUE * 20 + 0)

```

## Security Model

The `DBMS_DATA_MINING_TRANSFORM` package is owned by user `SYS` and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The `DBMS_DATA_MINING_TRANSFORM.INSERT_*` procedures have a `data_table_name` parameter that enables the user to provide the input data for transformation purposes. The value of `data_table_name` can be the name of a physical table or a view. The `data_table_name` parameter can also accept an inline query.

---

---

**Important:** Because an inline query can be used to specify the data for transformation, Oracle strongly recommends that the calling routine perform any necessary SQL injection checks on the input string.

---

---

**See Also:** ["Operational Notes"](#) on page 45-6 for a description of the `DBMS_DATA_MINING_TRANSFORM.INSERT_*` procedures

## Types

DBMS\_DATA\_MINING\_TRANSFORM defines the data types described in [Table 45–1](#).

**Table 45–1 Data Types in DBMS\_DATA\_MINING\_TRANSFORM**

List Type	List Elements	Description
<b>COLUMN_LIST</b>	VARRAY(1000) OF varchar2(32)	COLUMN_LIST stores quoted and non-quoted identifiers for column names.  COLUMN_LIST is the data type of the <i>exclude_list</i> parameter in the INSERT procedures. See <a href="#">"INSERT_AUTOBIN_NUM_EQWIDTH Procedure"</a> on page 45-35 for an example.  See <i>Oracle Database PL/SQL Language Reference</i> for information about populating VARRAY structures.
<b>DESCRIBE_LIST</b>	DBMS_SQL.DESC_TAB2  TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER  TYPE desc_rec2 IS RECORD ( col_type BINARY_INTEGER := 0, col_max_len BINARY_INTEGER := 0, col_name VARCHAR2(32767) := '', col_name_len BINARY_INTEGER := 0, col_schema_name VARCHAR2(32) := '', col_schema_name_len BINARY_INTEGER := 0, col_precision BINARY_INTEGER := 0, col_scale BINARY_INTEGER := 0, col_charsetid BINARY_INTEGER := 0, col_charsetform BINARY_INTEGER := 0, col_null_ok BOOLEAN := TRUE);	DESCRIBE_LIST describes the columns of the data table after the transformation list has been applied. A DESCRIBE_LIST is returned by the <a href="#">DESCRIBE_STACK Procedure</a> .  The DESC_TAB2 and DESC_REC2 types are defined in the DBMS_SQL package. See <a href="#">"DESC_REC2 Record Type"</a> on page 136-27.  The col_type field of DESC_REC2 identifies the data type of the column. The data type is expressed as a numeric constant that represents a built-in data type. For example, a 1 indicates a variable length character string. The codes for Oracle built-in data types are listed in <i>Oracle Database SQL Language Reference</i> . The codes for the Oracle Data Mining nested types are described in <a href="#">"Constants"</a> on page 45-16.  The col_name field of DESC_REC2 identifies the column name. It may be populated with a column name, an alias, or an expression. If the column name is a SELECT expression, it may be very long. If the expression is longer than 30 bytes, it cannot be used in a view unless it is given an alias.
<b>TRANSFORM_LIST</b>	TABLE OF transform_rec  TYPE transform_rec IS RECORD ( attribute_name VARCHAR2(30), attribute_subname VARCHAR2(4000), expression <b>EXPRESSION_REC</b> , reverse_expression <b>EXPRESSION_REC</b> , attribute_spec VARCHAR2(4000));  TYPE expression_rec IS RECORD ( lstmt DBMS_SQL.VARCHAR2A, lb BINARY_INTEGER DEFAULT 1, ub BINARY_INTEGER DEFAULT 0);  TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;	TRANSFORM_LIST is a list of transformations that can be embedded in a model. A TRANSFORM_LIST is accepted as an argument by the <a href="#">CREATE_MODEL Procedure</a> .  Each element in a TRANSFORM_LIST is a TRANSFORM_REC that specifies how to transform a single attribute. The attribute_name is a column name. The attribute_subname is the nested attribute name if the column is nested, otherwise attribute_subname is null.  The expression field holds a SQL expression for transforming the attribute. See <a href="#">"About Transformation Lists"</a> on page 45-7 for an explanation of reverse expressions.  The attribute_spec field can be used to disable ADP for this attribute. Specify the keyword NOPREP to disable ADP. By default attribute_spec is null.  The expressions in a TRANSFORM_REC have type EXPRESSION_REC. The lstmt field stores a VARCHAR2A, which is a table of VARCHAR2(32767). The VARCHAR2A data type allows transformation expressions to be very long, as they can be broken up across multiple rows of VARCHAR2. The VARCHAR2A type is defined in the DBMS_SQL package. See <a href="#">"VARCHAR2A Table Type"</a> on page 136-48.  The ub (upper bound) and lb (lower bound) fields indicate how many rows there are in the VARCHAR2A table. If ub < lb (default) the EXPRESSION_REC is empty; if lb=ub=1 there is one row; if lb=1 and ub=2 there are 2 rows, and so on.

## Constants

DBMS\_DATA\_MINING\_TRANSFORM defines the constants described in [Table 45-2](#).

**Table 45-2 Constants in DBMS\_DATA\_MINING\_TRANSFORM**

Constant	Value	Description
NEST_NUM_COL_TYPE	100001	<p>Indicates that an attribute in the transformation list comes from a row in a column of DM_NESTED_NUMERICALS.</p> <p>Nested numerical attributes are defined as follows:</p> <pre>attribute_name    VARCHAR2 (4000) value             NUMBER</pre>
NEST_CAT_COL_TYPE	100002	<p>Indicates that an attribute in the transformation list comes from a row in a column of DM_NESTED_CATAGORICALS.</p> <p>Nested categorical attributes are defined as follows:</p> <pre>attribute_name    VARCHAR2 (4000) value             VARCHAR2 (4000)</pre>

**See Also:** *Oracle Data Mining Application Developer's Guide* for information about nested data in Oracle Data Mining



---

## Summary of DBMS\_DATA\_MINING\_TRANSFORM Subprograms

**Table 45-3 DBMS\_DATA\_MINING\_TRANSFORM Package Subprograms**

Subprogram	Purpose
<a href="#">CREATE_BIN_CAT Procedure</a> on page 45-19	Creates a transformation definition table for categorical binning
<a href="#">CREATE_BIN_NUM Procedure</a> on page 45-21	Creates a transformation definition table for numerical binning
<a href="#">CREATE_CLIP Procedure</a> on page 45-23	Creates a transformation definition table for clipping
<a href="#">CREATE_COL_REM Procedure</a> on page 45-25	Creates a transformation definition table for column removal
<a href="#">CREATE_MISS_CAT Procedure</a> on page 45-26	Creates a transformation definition table for categorical missing value treatment
<a href="#">CREATE_MISS_NUM Procedure</a> on page 45-28	Creates a transformation definition table for numerical missing values treatment
<a href="#">CREATE_NORM_LIN Procedure</a> on page 45-30	Creates a transformation definition table for linear normalization
<a href="#">DESCRIBE_STACK Procedure</a> on page 45-32	Describes the transformation list
<a href="#">GET_EXPRESSION Function</a> on page 45-34	Returns a VARCHAR2 chunk from a transformation expression
<a href="#">INSERT_AUTOBIN_NUM_EQW IDTH Procedure</a> on page 45-34	Inserts numeric automatic equi-width binning definitions in a transformation definition table
<a href="#">INSERT_BIN_CAT_FREQ Procedure</a> on page 45-39	Inserts categorical frequency-based binning definitions in a transformation definition table
<a href="#">INSERT_BIN_NUM_EQWIDTH Procedure</a> on page 45-44	Inserts numeric equi-width binning definitions in a transformation definition table
<a href="#">INSERT_BIN_NUM_QTILE Procedure</a> on page 45-48	Inserts numeric quantile binning expressions in a transformation definition table
<a href="#">INSERT_BIN_SUPER Procedure</a> on page 45-51	Inserts supervised binning definitions in numerical and categorical transformation definition tables
<a href="#">INSERT_CLIP_TRIM_TAIL Procedure</a> on page 45-55	Inserts numerical trimming definitions in a transformation definition table
<a href="#">INSERT_CLIP_WINSOR_TAIL Procedure</a> on page 45-58	Inserts numerical winsorizing definitions in a transformation definition table
<a href="#">INSERT_MISS_CAT_MODE Procedure</a> on page 45-61	Inserts categorical missing value treatment definitions in a transformation definition table
<a href="#">INSERT_MISS_NUM_MEAN Procedure</a> on page 45-64	Inserts numerical missing value treatment definitions in a transformation definition table
<a href="#">INSERT_NORM_LIN_MINMAX Procedure</a> on page 45-67	Inserts linear min-max normalization definitions in a transformation definition table
<a href="#">INSERT_NORM_LIN_SCALE Procedure</a> on page 45-70	Inserts linear scale normalization definitions in a transformation definition table
<a href="#">INSERT_NORM_LIN_ZSCORE Procedure</a> on page 45-73	Inserts linear zscore normalization definitions in a transformation definition table

**Table 45-3 (Cont.) DBMS\_DATA\_MINING\_TRANSFORM Package Subprograms**

<b>Subprogram</b>	<b>Purpose</b>
<a href="#">SET_EXPRESSION Procedure</a> on page 45-76	Adds a VARCHAR2 chunk to an expression
<a href="#">SET_TRANSFORM Procedure</a> on page 45-78	Adds a transformation record to a transformation list
<a href="#">STACK_BIN_CAT Procedure</a> on page 45-80	Adds a categorical binning expression to a transformation list
<a href="#">STACK_BIN_NUM Procedure</a> on page 45-82	Adds a numerical binning expression to a transformation list
<a href="#">STACK_CLIP Procedure</a> on page 45-85	Adds a clipping expression to a transformation list
<a href="#">STACK_COL_REM Procedure</a> on page 45-87	Adds a column removal expression to a transformation list
<a href="#">STACK_MISS_CAT Procedure</a> on page 45-89	Adds a categorical missing value treatment expression to a transformation list
<a href="#">STACK_MISS_NUM Procedure</a> on page 45-91	Adds a numerical missing value treatment expression to a transformation list
<a href="#">STACK_NORM_LIN Procedure</a> on page 45-93	Adds a linear normalization expression to a transformation list
<a href="#">XFORM_BIN_CAT Procedure</a> on page 45-95	Creates a view of the data table with categorical binning transformations
<a href="#">XFORM_BIN_NUM Procedure</a> on page 45-97	Creates a view of the data table with numerical binning transformations
<a href="#">XFORM_CLIP Procedure</a> on page 45-100	Creates a view of the data table with clipping transformations
<a href="#">XFORM_COL_REM Procedure</a> on page 45-102	Creates a view of the data table with column removal transformations
<a href="#">XFORM_EXPR_NUM Procedure</a> on page 45-104	Creates a view of the data table with the specified numeric transformations
<a href="#">XFORM_EXPR_STR Procedure</a> on page 45-106	Creates a view of the data table with the specified categorical transformations
<a href="#">XFORM_MISS_CAT Procedure</a> on page 45-109	Creates a view of the data table with categorical missing value treatment
<a href="#">XFORM_MISS_NUM Procedure</a> on page 45-112	Creates a view of the data table with numerical missing value treatment
<a href="#">XFORM_NORM_LIN Procedure</a> on page 45-114	Creates a view of the data table with linear normalization transformations
<a href="#">XFORM_STACK Procedure</a> on page 45-117	Creates a view of the transformation list

## CREATE\_BIN\_CAT Procedure

This procedure creates a transformation definition table for categorical binning. The columns are described in [Table 45-4](#).

**Table 45-4 Columns in a Transformation Definition Table for Categorical Binning**

Name	Data Type	Description
col	VARCHAR2(30)	Name of a column of CHAR, VARCHAR2, or DM_NESTED_CATEGORICALS in the data to be mined.  If col is CHAR or VARCHAR2, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2(4000)	The attribute subname if col is a nested column of DM_NESTED_CATEGORICALS. If col is nested, the attribute name is col.att.  If col is not nested, att is null.
val	VARCHAR2(4000)	Values of the attribute
bin	VARCHAR2(4000)	Bin assignments for the values

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT (
    bin_table_name      IN VARCHAR2,
    bin_schema_name    IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45-5 CREATE\_BIN\_CAT Procedure Parameters**

Parameter	Description
bin_table_name	Name of the transformation definition table to be created
bin_schema_name	Schema of bin_table_name. If no schema is specified, the current schema is used.

### Usage Notes

You can use the following procedures to populate the transformation definition table:

- [INSERT\\_BIN\\_CAT\\_FREQ Procedure](#) — frequency-based binning
- [INSERT\\_BIN\\_SUPER Procedure](#) — supervised binning

**See Also:**

"Binning" on page 45-4

"Operational Notes" on page 45-6

### Examples

The following statement creates a table called bin\_cat\_xtbl in the current schema. The table has columns that can be populated with bin assignments for categorical attributes.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_CAT('bin_cat_xtbl');
END;
/
```

```
DESCRIBE bin_cat_xtbl
```

Name	Null?	Type
COL		VARCHAR2(30)
ATT		VARCHAR2(4000)
VAL		VARCHAR2(4000)
BIN		VARCHAR2(4000)

## CREATE\_BIN\_NUM Procedure

This procedure creates a transformation definition table for numerical binning. The columns are described in [Table 45–6](#).

**Table 45–6 Columns in a Transformation Definition Table for Numerical Binning**

Name	Data Type	Description
col	VARCHAR2 (30)	Name of a column of NUMBER, FLOAT, or DM_NESTED_NUMERICALS in the data to be mined.  If the column is NUMBER or FLOAT, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2 (4000)	The attribute subname if col is a nested column of DM_NESTED_NUMERICALS. If col is nested, the attribute name is col.att.  If col is not nested, att is null.
val	NUMBER	Values of the attribute
bin	VARCHAR2 (4000)	Bin assignments for the values

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM (
    bin_table_name    IN VARCHAR2,
    bin_schema_name  IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45–7 CREATE\_BIN\_NUM Procedure Parameters**

Parameter	Description
bin_table_name	Name of the transformation definition table to be created
bin_schema_name	Schema of bin_table_name. If no schema is specified, the current schema is used.

### Usage Notes

You can use the following procedures to populate the transformation definition table:

- [INSERT\\_AUTOBIN\\_NUM\\_EQWIDTH Procedure](#) — automatic equi-width binning
- [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) — user-specified equi-width binning
- [INSERT\\_BIN\\_NUM\\_QTILE Procedure](#) — quantile binning
- [INSERT\\_BIN\\_SUPER Procedure](#) — supervised binning

**See Also:**

"Binning" on page 45-4

"Operational Notes" on page 45-6

## Examples

The following statement creates a table called `bin_num_xtbl` in the current schema. The table has columns that can be populated with bin assignments for numerical attributes.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_BIN_NUM('bin_num_xtbl');
END;
/
```

```
DESCRIBE bin_num_xtbl
```

Name	Null?	Type
COL		VARCHAR2(30)
ATT		VARCHAR2(4000)
VAL		NUMBER
BIN		VARCHAR2(4000)

## CREATE\_CLIP Procedure

This procedure creates a transformation definition table for clipping or winsorizing to minimize the effect of outliers. The columns are described in [Table 45–8](#).

**Table 45–8 Columns in a Transformation Definition Table for Clipping or Winsorizing**

Name	Data Type	Description
col	VARCHAR2 (30)	Name of a column of NUMBER, FLOAT, or DM_NESTED_NUMERICALS in the data to be mined.  If <i>col</i> is NUMBER or FLOAT, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2 (4000)	The attribute subname if <i>col</i> is a nested column of DM_NESTED_NUMERICALS. If <i>col</i> is nested, the attribute name is <i>col.att</i> .  If <i>col</i> is not nested, <i>att</i> is null.
lcut	NUMBER	The lowest typical value for the attribute.  If the attribute values were plotted on an <i>xy</i> axis, <i>lcut</i> would be the left-most boundary of the range of values considered typical for this attribute.  Any values to the left of <i>lcut</i> are outliers.
lval	NUMBER	Value assigned to an outlier to the left of <i>lcut</i>
rcut	NUMBER	The highest typical value for the attribute  If the attribute values were plotted on an <i>xy</i> axis, <i>rcut</i> would be the right-most boundary of the range of values considered typical for this attribute.  Any values to the right of <i>rcut</i> are outliers.
rval	NUMBER	Value assigned to an outlier to the right of <i>rcut</i>

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP (
    clip_table_name    IN VARCHAR2,
    clip_schema_name  IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45–9 CREATE\_CLIP Procedure Parameters**

Parameter	Description
clip_table_name	Name of the transformation definition table to be created
clip_schema_name	Schema of <i>clip_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

You can use the following procedures to populate the transformation definition table:

- [INSERT\\_CLIP\\_TRIM\\_TAIL Procedure](#) — replaces outliers with nulls

- [INSERT\\_CLIP\\_WINSOR\\_TAIL Procedure](#) — replaces outliers with an average value

**See Also:**

["Outlier Treatment"](#) on page 45-5

["Operational Notes"](#) on page 45-6

## Examples

The following statement creates a table called `clip_xtbl` in the current schema. The table has columns that can be populated with clipping instructions for numerical attributes.

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_CLIP('clip_xtbl');
END;
/
```

```
DESCRIBE clip_xtbl
```

Name	Null?	Type
COL		VARCHAR2 (30)
ATT		VARCHAR2 (4000)
LCUT		NUMBER
LVAL		NUMBER
RCUT		NUMBER
RVAL		NUMBER



## CREATE\_COL\_REM Procedure

This procedure creates a transformation definition table for removing columns from the data table. The columns are described in [Table 45–10](#).

**Table 45–10 Columns in a Transformation Definition Table for Column Removal**

Name	Data Type	Description
col	VARCHAR2 (30)	Name of a column in the data to be mined. If <i>col</i> is not nested, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2 (4000)	The attribute subname if <i>col</i> is nested (DM_NESTED_NUMERICALS or DM_NESTED_CATEGORICALS). If <i>col</i> is nested, the attribute name is <i>col.att</i> .  If <i>col</i> is not nested, <i>att</i> is null.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM (
    rem_table_name      VARCHAR2,
    rem_schema_name    VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45–11 CREATE\_COL\_REM Procedure Parameters**

Parameter	Description
rem_table_name	Name of the transformation definition table to be created
rem_schema_name	Schema of <i>rem_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

See "[Operational Notes](#)" on page 45-6.

### Examples

The following statement creates a table called *rem\_att\_xtbl* in the current schema. The table has columns that can be populated with the names of attributes to exclude from the data to be mined.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM ('rem_att_xtbl');
END;
/
DESCRIBE rem_att_xtbl
```

Name	Null?	Type
COL		VARCHAR2 (30)
ATT		VARCHAR2 (4000)

## CREATE\_MISS\_CAT Procedure

This procedure creates a transformation definition table for replacing categorical missing values. The columns are described in [Table 45–12](#).

**Table 45–12 Columns in a Transformation Definition Table for Categorical Missing Value Treatment**

Name	Data Type	Description
col	VARCHAR2 (30)	Name of a column of CHAR, VARCHAR2, or DM_NESTED_CATEGORICALS in the data to be mined.  If <i>col</i> is CHAR or VARCHAR2, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2 (4000)	The attribute subname if <i>col</i> is a nested column of DM_NESTED_CATEGORICALS. If <i>col</i> is nested, the attribute name is <i>col.att</i> .  If <i>col</i> is not nested, <i>att</i> is null.
val	VARCHAR2 (4000)	Replacement for missing values in the attribute

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT (
    miss_table_name      IN VARCHAR2,
    miss_schema_name    IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45–13 CREATE\_MISS\_CAT Procedure Parameters**

Parameter	Description
miss_table_name	Name of the transformation definition table to be created
miss_schema_name	Schema of <i>miss_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

You can use the [INSERT\\_MISS\\_CAT\\_MODE Procedure](#) to populate the transformation definition table.

**See Also:**

["Missing Value Treatment"](#) on page 45-5

["Operational Notes"](#) on page 45-6

### Examples

The following statement creates a table called `miss_cat_xtbl` in the current schema. The table has columns that can be populated with values for missing data in categorical attributes.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('miss_cat_xtbl');
END;
```

/

DESCRIBE miss\_cat\_xtbl

Name	Null?	Type
COL		VARCHAR2(30)
ATT		VARCHAR2(4000)
VAL		VARCHAR2(4000)

## CREATE\_MISS\_NUM Procedure

This procedure creates a transformation definition table for replacing numerical missing values. The columns are described in [Table 45–14](#).

**Table 45–14 Columns in a Transformation Definition Table for Numerical Missing Value Treatment**

Name	Data Type	Description
col	VARCHAR2 (30)	Name of a column of NUMBER, FLOAT, or DM_NESTED_NUMERICALS in the data to be mined.  If the column is NUMBER or FLOAT, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2 (4000)	The attribute subname if col is a nested column of DM_NESTED_NUMERICALS. If col is nested, the attribute name is col.att.  If col is not nested, att is null.
val	NUMBER	Replacement for missing values in the attribute

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM (
    miss_table_name      IN VARCHAR2,
    miss_schema_name    IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45–15 CREATE\_MISS\_NUM Procedure Parameters**

Parameter	Description
miss_table_name	Name of the transformation definition table to be created
miss_schema_name	Schema of miss_table_name. If no schema is specified, the current schema is used.

### Usage Notes

You can use the [INSERT\\_MISS\\_NUM\\_MEAN Procedure](#) to populate the transformation definition table.

**See Also:**

["Missing Value Treatment"](#) on page 45-5

["Operational Notes"](#) on page 45-6

### Example

The following statement creates a table called miss\_num\_xtbl in the current schema. The table has columns that can be populated with values for missing data in numerical attributes.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('miss_num_xtbl');
END;
```

/

DESCRIBE miss\_num\_xtbl

Name	Null?	Type
COL		VARCHAR2 (30)
ATT		VARCHAR2 (4000)
VAL		NUMBER

## CREATE\_NORM\_LIN Procedure

This procedure creates a transformation definition table for linear normalization. The columns are described in [Table 45–16](#).

**Table 45–16 Columns in a Transformation Definition Table for Linear Normalization**

Name	Data Type	Description
col	VARCHAR2 (30)	Name of a column of NUMBER, FLOAT, or DM_NESTED_NUMERICALS in the data to be mined.  If the column is NUMBER or FLOAT, the column name is also the attribute name.  For information about attribute names, see <i>Oracle Data Mining Application Developer's Guide</i> . Refer especially to the section, "Scoping of Model Attribute Name".
att	VARCHAR2 (4000)	The attribute subname if col is a nested column of DM_NESTED_NUMERICALS. If col is nested, the attribute name is col.att.  If col is not nested, att is null.
shift	NUMBER	A constant to subtract from the attribute values
scale	NUMBER	A constant by which to divide the shifted values

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN (
    norm_table_name      IN VARCHAR2,
    norm_schema_name    IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 45–17 CREATE\_NORM\_LIN Procedure Parameters**

Parameter	Description
norm_table_name	Name of the transformation definition table to be created
norm_schema_name	Schema of norm_table_name. If no schema is specified, the current schema is used.

### Usage Notes

You can use the following procedures to populate the transformation definition table:

- [INSERT\\_NORM\\_LIN\\_MINMAX Procedure](#) — Uses linear min-max normalization
- [INSERT\\_NORM\\_LIN\\_SCALE Procedure](#) — Uses linear scale normalization
- [INSERT\\_NORM\\_LIN\\_ZSCORE Procedure](#) — Uses linear zscore normalization

**See Also:**

["Linear Normalization"](#) on page 45-4

["Operational Notes"](#) on page 45-6

## Examples

The following statement creates a table called `norm_xtbl` in the current schema. The table has columns that can be populated with shift and scale values for normalizing numerical attributes.

```
BEGIN
    DBMS_DATA_MINING_TRANSFORM.CREATE_NORM_LIN('norm_xtbl');
END;
/
```

```
DESCRIBE norm_xtbl
```

Name	Null?	Type
COL		VARCHAR2 (30)
ATT		VARCHAR2 (4000)
SHIFT		NUMBER
SCALE		NUMBER

## DESCRIBE\_STACK Procedure

This procedure describes the columns of the data table after a list of transformations has been applied. Only the columns that are specified in the transformation list are transformed. The remaining columns in the data table are included in the output without changes.

To create a view of the data table after the transformations have been applied, use the [XFORM\\_STACK Procedure](#).

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.DESCRIBE_STACK (
  xform_list          IN  TRANSFORM_LIST,
  data_table_name     IN  VARCHAR2,
  describe_list       OUT DESCRIBE_LIST,
  data_schema_name    IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–18 DESCRIBE\_STACK Procedure Parameters**

Parameter	Description
<code>xform_list</code>	A list of transformations. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>describe_list</code>	Descriptions of the columns in the data table after the transformations specified in <code>xform_list</code> have been applied. See <a href="#">Table 45–1</a> for a description of the <code>DESCRIBE_LIST</code> object type.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See "[Operational Notes](#)" on page 45-6 for information about transformation lists and embedded transformations.

### Examples

This example shows the column name and data type, the column name length, and the column maximum length for the view `dmuser.cust_info` after the transformation list has been applied. All the transformations are user-specified. The results of `DESCRIBE_STACK` do not include one of the columns in the original table, because the `SET_TRANSFORM` procedure sets that column to `NULL`.

```
CREATE OR REPLACE VIEW cust_info AS
  SELECT a.cust_id, c.country_id, c.cust_year_of_birth,
         CAST(COLLECT(DM_Nested_Numerical(
           b.prod_name, 1))
           AS DM_Nested_Numericals) custprods
  FROM sh.sales a, sh.products b, sh.customers c
  WHERE a.prod_id = b.prod_id AND
         a.cust_id=c.cust_id and
         a.cust_id between 100001 AND 105000
  GROUP BY a.cust_id, country_id, cust_year_of_birth;

describe cust_info
```



Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
COUNTRY_ID	NOT NULL	NUMBER
CUST_YEAR_OF_BIRTH	NOT NULL	NUMBER(4)
CUSTPRODS		SYS.DM_NESTED_NUMERICALS

```

DECLARE
  cust_stack  dbms_data_mining_transform.TRANSFORM_LIST;
  cust_cols   dbms_data_mining_transform.DESCRIBE_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'country_id', NULL, 'country_id/10', 'country_id*10');
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'cust_year_of_birth', NULL, NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'custprods', 'Mouse Pad', 'value*100', 'value/100');
  dbms_data_mining_transform.DESCRIBE_STACK(
    xform_list => cust_stack,
    data_table_name => 'cust_info',
    describe_list => cust_cols);
  dbms_output.put_line('====');
  for i in 1..cust_cols.COUNT loop
    dbms_output.put_line('COLUMN_NAME:      ' || cust_cols(i).col_name);
    dbms_output.put_line('COLUMN_TYPE:      ' || cust_cols(i).col_type);
    dbms_output.put_line('COLUMN_NAME_LEN:  ' || cust_cols(i).col_name_len);
    dbms_output.put_line('COLUMN_MAX_LEN:   ' || cust_cols(i).col_max_len);
    dbms_output.put_line('====');
  END loop;
END;
/
====
COLUMN_NAME:      CUST_ID
COLUMN_TYPE:      2
COLUMN_NAME_LEN:  7
COLUMN_MAX_LEN:   22
====
COLUMN_NAME:      COUNTRY_ID
COLUMN_TYPE:      2
COLUMN_NAME_LEN:  10
COLUMN_MAX_LEN:   22
====
COLUMN_NAME:      CUSTPRODS
COLUMN_TYPE:      100001
COLUMN_NAME_LEN:  9
COLUMN_MAX_LEN:   40
====

```

## GET\_EXPRESSION Function

This function returns a row from a VARCHAR2 array that stores a transformation expression. The array is built by calls to the [SET\\_EXPRESSION Procedure](#).

The array can be used for specifying SQL expressions that are too long to be used with the [SET\\_TRANSFORM Procedure](#).

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.GET_EXPRESSION (
    expression          IN EXPRESSION_REC,
    chunk_num          IN PLS_INTEGER DEFAULT NULL);
RETURN VARCHAR2;
```

### Parameters

**Table 45–19 GET\_EXPRESSION Function Parameters**

Parameter	Description
expression	An expression record (EXPRESSION_REC) that specifies a transformation expression or a reverse transformation expression for an attribute. Each expression record includes a VARCHAR2 array and index fields for specifying upper and lower boundaries within the array.  There are two EXPRESSION_REC fields within a transformation record (TRANSFORM_REC): one for the transformation expression; the other for the reverse transformation expression.  See <a href="#">Table 45–1</a> for a description of the EXPRESSION_REC type.
chunk	A VARCHAR2 chunk (row) to be appended to <i>expression</i> .

### Usage Notes

1. Chunk numbering starts with one. For chunks outside of the range, the return value is null. When a chunk number is null the whole expression is returned as a string. If the expression is too big, a VALUE\_ERROR is raised.
2. See "[About Transformation Lists](#)" on page 45-7.
3. See "[Operational Notes](#)" on page 45-6.

### Examples

See the example for the [SET\\_EXPRESSION Procedure](#).

## INSERT\_AUTOBIN\_NUM\_EQWIDTH Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure identifies the minimum and maximum values and computes the bin boundaries at equal intervals.

INSERT\_AUTOBIN\_NUM\_EQWIDTH computes the number of bins separately for each column. If you want to use equi-width binning with the same number of bins for each column, use the [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#).

INSERT\_AUTOBIN\_NUM\_EQWIDTH bins all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_AUTOBIN_NUM_EQWIDTH (
    bin_table_name      IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    bin_num              IN PLS_INTEGER DEFAULT 3,
    max_bin_num         IN PLS_INTEGER DEFAULT 100,
    exclude_list        IN COLUMN_LIST DEFAULT NULL,
    round_num           IN PLS_INTEGER DEFAULT 6,
    sample_size         IN PLS_INTEGER DEFAULT 50000,
    bin_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL,
    rem_table_name      IN VARCHAR2 DEFAULT NULL,
    rem_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–20** INSERT\_AUTOBIN\_NUM\_EQWIDTH Procedure Parameters

Parameter	Description
bin_table_name	Name of the transformation definition table for numerical binning. You can use the <a href="#">CREATE_BIN_NUM Procedure</a> to create the definition table. The following columns are required: COL        VARCHAR2(30) VAL        NUMBER BIN        VARCHAR2(4000) CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_AUTOBIN_NUM_EQWIDTH.
data_table_name	Name of the table containing the data to be transformed
bin_num	Minimum number of bins. If <i>bin_num</i> is 0 or NULL, it is ignored. The default value of <i>bin_num</i> is 3.
max_bin_num	Maximum number of bins. If <i>max_bin_num</i> is 0 or NULL, it is ignored. The default value of <i>max_bin_num</i> is 100.
exclude_list	List of numeric columns to be excluded from the binning process. If you do not specify <i>exclude_list</i> , all numeric columns in the data source are binned. The format of <i>exclude_list</i> is: dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')

**Table 45–20 (Cont.) INSERT\_AUTOBIN\_NUM\_EQWIDTH Procedure Parameters**

Parameter	Description
<code>round_num</code>	<p>Specifies how to round the number in the <code>VAL</code> column of the transformation definition table.</p> <p>When <code>round_num</code> is positive, it specifies the most significant digits to retain. When <code>round_num</code> is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. See the Usage Notes for an example.</p> <p>The default value of <code>round_num</code> is 6.</p>
<code>sample_size</code>	<p>Size of the data sample. If <code>sample_size</code> is less than the total number of non-NULL values in the column, then <code>sample_size</code> is used instead of the SQL <code>COUNT</code> function in computing the number of bins. If <code>sample_size</code> is 0 or NULL, it is ignored. See the Usage Notes.</p> <p>The default value of <code>sample_size</code> is 50,000.</p>
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>rem_table_name</code>	<p>Name of a transformation definition table for column removal. The table must have the columns described in "<a href="#">CREATE_COL_REM Procedure</a>" on page 45-25.</p> <p>INSERT_AUTOBIN_NUM_EQWIDTH ignores columns with all nulls or only one unique value. If you specify a value for <code>rem_table_name</code>, these columns are removed from the mining data. If you do not specify a value for <code>rem_table_name</code>, these unbinned columns remain in the data.</p>
<code>rem_schema_name</code>	Schema of <code>rem_table_name</code> . If no schema is specified, the current schema is used.

## Usage Notes

- INSERT\_AUTOBIN\_NUM\_EQWIDTH computes the number of bins for a column based on the number of non-null values (`COUNT`), the maximum (`MAX`), the minimum (`MIN`), the standard deviation (`STDDEV`), and the constant  $C=3.49/0.9$ :

$$N = \text{floor}(\text{power}(\text{COUNT}, 1/3) * (\text{max} - \text{min}) / (c * \text{dev}))$$

If the `sample_size` parameter is specified, it is used instead of `COUNT`.

See *Oracle Database SQL Language Reference* for information about the `COUNT`, `MAX`, `MIN`, `STDDEV`, `FLOOR`, and `POWER` functions.
- INSERT\_AUTOBIN\_NUM\_EQWIDTH uses absolute values to compute the number of bins. The sign of the parameters `bin_num`, `max_bin_num`, and `sample_size` has no effect on the result.
- In computing the number of bins, INSERT\_AUTOBIN\_NUM\_EQWIDTH evaluates the following criteria in the following order:

  - The minimum number of bins (`bin_num`)
  - The maximum number of bins (`max_bin_num`)
  - The maximum number of bins for integer columns, calculated as the number of distinct values in the range `max-min+1`.
- The `round_num` parameter controls the rounding of column values in the transformation definition table, as follows:

**For a value of 308.162:**

```

when round_num = 1      result is 300
when round_num = 2      result is 310
when round_num = 3      result is 308
when round_num = 0      result is 308.162
when round_num = -1     result is 308.16
when round_num = -2     result is 308.2

```

**Examples**

In this example, `INSERT_AUTOBIN_NUM_EQWIDTH` computes the bin boundaries for the `cust_year_of_birth` column in `sh.customers` and inserts the transformations in a transformation definition table. The [STACK\\_BIN\\_NUM Procedure](#) creates a transformation list from the contents of the definition table. The [CREATE\\_MODEL Procedure](#) embeds the transformation list in a new model called `nb_model`.

The transformation and reverse transformation expressions embedded in `nb_model` are returned by the [GET\\_MODEL\\_TRANSFORMATIONS Function](#).

```

CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_postal_code
  FROM sh.customers;

```

```

DESCRIBE mining_data

```

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_YEAR_OF_BIRTH	NOT NULL	NUMBER(4)
CUST_POSTAL_CODE	NOT NULL	VARCHAR2(10)

```

BEGIN
  dbms_data_mining_transform.CREATE_BIN_NUM(
    bin_table_name => 'bin_tbl');
  dbms_data_mining_transform.INSERT_AUTOBIN_NUM_EQWIDTH (
    bin_table_name => 'bin_tbl',
    data_table_name => 'mining_data',
    bin_num         => 3,
    max_bin_num    => 5,
    exclude_list   => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/

```

```

set numwidth 4
column val off
SELECT col, val, bin FROM bin_tbl
  ORDER BY val ASC;

```

COL	VAL	BIN
CUST_YEAR_OF_BIRTH	1913	
CUST_YEAR_OF_BIRTH	1928	1
CUST_YEAR_OF_BIRTH	1944	2
CUST_YEAR_OF_BIRTH	1959	3
CUST_YEAR_OF_BIRTH	1975	4
CUST_YEAR_OF_BIRTH	1990	5

```

DECLARE
  year_birth_xform dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_NUM (

```

```

        bin_table_name      => 'bin_tbl',
        xform_list          => year_birth_xform);
dbms_data_mining.CREATE_MODEL(
    model_name              => 'nb_model',
    mining_function         => dbms_data_mining.classification,
    data_table_name        => 'mining_data',
    case_id_column_name    => 'cust_id',
    target_column_name     => 'cust_postal_code',
    settings_table_name    => null,
    data_schema_name       => null,
    settings_schema_name   => null,
    xform_list             => year_birth_xform);
END;
/

SELECT attribute_name
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

ATTRIBUTE_NAME
-----
CUST_YEAR_OF_BIRTH

SELECT expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

EXPRESSION
-----
CASE WHEN "CUST_YEAR_OF_BIRTH"<1913 THEN NULL WHEN "CUST_YEAR_OF_BIRTH"<=1928.4
 THEN '1' WHEN "CUST_YEAR_OF_BIRTH"<=1943.8 THEN '2' WHEN "CUST_YEAR_OF_BIRTH"
<=1959.2 THEN '3' WHEN "CUST_YEAR_OF_BIRTH"<=1974.6 THEN '4' WHEN
"CUST_YEAR_OF_BIRTH" <=1990 THEN '5' END

SELECT reverse_expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

REVERSE_EXPRESSION
-----
DECODE("CUST_YEAR_OF_BIRTH", '5', '(1974.6; 1990]', '1', '[1913; 1928.4]', '2', '(1928
.4; 1943.8]', '3', '(1943.8; 1959.2]', '4', '(1959.2; 1974.6]', NULL, '( ; 1913), (199
0; ), NULL')

```

## INSERT\_BIN\_CAT\_FREQ Procedure

This procedure performs categorical binning and inserts the transformation definitions in a transformation definition table. The procedure computes the bin boundaries based on frequency.

INSERT\_BIN\_CAT\_FREQ bins all the CHAR and VARCHAR2 columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_CAT_FREQ (
    bin_table_name      IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    bin_num             IN PLS_INTEGER DEFAULT 9,
    exclude_list        IN COLUMN_LIST DEFAULT NULL,
    default_num         IN PLS_INTEGER DEFAULT 2,
    bin_support          IN NUMBER DEFAULT NULL,
    bin_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–21** INSERT\_BIN\_CAT\_FREQ Procedure Parameters

Parameter	Description
bin_table_name	Name of the transformation definition table for categorical binning. You can use the <a href="#">CREATE_BIN_CAT Procedure</a> to create the definition table. The following columns are required: COL        VARCHAR2 (30) VAL        VARCHAR2 (4000) BIN        VARCHAR2 (4000) CREATE_BIN_CAT creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_CAT_FREQ.
data_table_name	Name of the table containing the data to be transformed
bin_num	The number of bins to fill using frequency-based binning. The total number of bins will be <i>bin_num</i> +1. The additional bin is the default bin. Classes that are not assigned to a frequency-based bin will be assigned to the default bin.  The default binning order is from highest to lowest: the most frequently occurring class is assigned to the first bin, the second most frequently occurring class is assigned to the second bin, and so on. You can reverse the binning order by specifying a negative number for <i>bin_num</i> . The negative sign causes the binning order to be from lowest to highest.  If the total number of distinct values (classes) in the column is less than <i>bin_num</i> , then a separate bin will be created for each value and the default bin will be empty.  If you specify NULL or 0 for <i>bin_num</i> , no binning is performed.  The default value of <i>bin_num</i> is 9.

**Table 45–21 (Cont.) INSERT\_BIN\_CAT\_FREQ Procedure Parameters**

Parameter	Description
<code>exclude_list</code>	<p>List of CHAR and VARCHAR2 columns to be excluded from the binning process. If you do not specify <code>exclude_list</code>, all CHAR and VARCHAR2 columns in the data source are binned.</p> <p>The format of <code>exclude_list</code> is:</p> <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ... 'coln')</pre>
<code>default_num</code>	<p>The number of class occurrences (rows of the same class) required for assignment to the default bin</p> <p>By default, <code>default_num</code> is the minimum number of occurrences required for assignment to the default bin. For example, if <code>default_num</code> is 3 and a given class occurs only once, it will not be assigned to the default bin. You can change the occurrence requirement from minimum to maximum by specifying a negative number for <code>default_num</code>. For example, if <code>default_num</code> is -3 and a given class occurs only once, it <i>will</i> be assigned to the default bin, but a class that occurs four or more times will not be included.</p> <p>If you specify NULL or 0 for <code>default_bin</code>, there are no requirements for assignment to the default bin.</p> <p>The default value of <code>default_num</code> is 2.</p>
<code>bin_support</code>	<p>The number of class occurrences (rows of the same class) required for assignment to a frequency-based bin. <code>bin_support</code> is expressed as a fraction of the total number of rows.</p> <p>By default, <code>bin_support</code> is the minimum percentage required for assignment to a frequency-based bin. For example, if there are twenty rows of data and you specify .2 for <code>bin_support</code>, then there must be four or more occurrences of a class (.2*20) in order for it to be assigned to a frequency-based bin. You can change <code>bin_support</code> from a minimum percentage to a maximum percentage by specifying a negative number for <code>bin_support</code>. For example, if there are twenty rows of data and you specify -.2 for <code>bin_support</code>, then there must be four or less occurrences of a class in order for it to be assigned to a frequency-based bin.</p> <p>Classes that occur less than a positive <code>bin_support</code> or more than a negative <code>bin_support</code> will be assigned to the default bin.</p> <p>If you specify NULL or 0 for <code>bin_support</code>, then there is no support requirement for frequency-based binning.</p> <p>The default value of <code>bin_support</code> is NULL.</p>
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Usage Notes

If values occur with the same frequency, INSERT\_BIN\_CAT\_FREQ assigns them in descending order when binning is from most to least frequent, or in ascending order when binning is from least to most frequent.

## Examples

1. In this example, INSERT\_BIN\_CAT\_FREQ computes the bin boundaries for the `cust_postal_code` and `cust_city` columns in `sh.customers` and inserts the transformations in a transformation definition table. The [STACK\\_BIN\\_CAT](#)



**Procedure** creates a transformation list from the contents of the definition table, and the **CREATE\_MODEL Procedure** embeds the transformation list in a new model called `nb_model`.

The transformation and reverse transformation expressions embedded in `nb_model` are returned by the **GET\_MODEL\_TRANSFORMATIONS Function**.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_postal_code, cust_city
    FROM sh.customers;
```

```
DESCRIBE mining_data
```

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_YEAR_OF_BIRTH	NOT NULL	NUMBER(4)
CUST_POSTAL_CODE	NOT NULL	VARCHAR2(10)
CUST_CITY	NOT NULL	VARCHAR2(30)

```
BEGIN
```

```
    dbms_data_mining_transform.CREATE_BIN_CAT(
        bin_table_name => 'bin_tbl_1');
    dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (
        bin_table_name => 'bin_tbl_1',
        data_table_name => 'mining_data',
        bin_num         => 4);
```

```
END;
```

```
/
```

```
column col format a18
column val format a15
column bin format a10
SELECT col, val, bin
    FROM bin_tbl_1
    ORDER BY col ASC, bin ASC;
```

COL	VAL	BIN
CUST_CITY	Los Angeles	1
CUST_CITY	Greenwich	2
CUST_CITY	Killarney	3
CUST_CITY	Montara	4
CUST_CITY		5
CUST_POSTAL_CODE	38082	1
CUST_POSTAL_CODE	63736	2
CUST_POSTAL_CODE	55787	3
CUST_POSTAL_CODE	78558	4
CUST_POSTAL_CODE		5

```
DECLARE
```

```
    city_xform dbms_data_mining_transform.TRANSFORM_LIST;
```

```
BEGIN
```

```
    dbms_data_mining_transform.STACK_BIN_CAT (
        bin_table_name => 'bin_tbl_1',
        xform_list     => city_xform);
    dbms_data_mining.CREATE_MODEL(
        model_name      => 'nb_model',
        mining_function => dbms_data_mining.classification,
        data_table_name => 'mining_data',
        case_id_column_name => 'cust_id',
        target_column_name => 'cust_city',
```

```

        settings_table_name => null,
        data_schema_name   => null,
        settings_schema_name => null,
        xform_list         => city_xform);
END;
/

SELECT attribute_name
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

ATTRIBUTE_NAME
-----
CUST_CITY
CUST_POSTAL_CODE

SELECT expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

EXPRESSION
-----
DECODE("CUST_CITY", 'Greenwich', '2', 'Killarney', '3', 'Los Angeles', '1',
'Montara', '4', NULL, NULL, '5')
DECODE("CUST_POSTAL_CODE", '38082', '1', '55787', '3', '63736', '2', '78558', '4', NULL,
NULL, '5')

SELECT reverse_expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('nb_model'));

REVERSE_EXPRESSION
-----
DECODE("CUST_CITY", '2', ''Greenwich'', '3', ''Killarney'', '1',
''Los Angeles'', '4', ''Montara'', NULL, 'NULL', '5', 'DEFAULT')
DECODE("CUST_POSTAL_CODE", '1', ''38082'', '3', ''55787'', '2', ''63736'',
'4', ''78558'', NULL, 'NULL', '5', 'DEFAULT')

```

- The binning order in example 1 is from most frequent to least frequent. The following example shows reverse order binning (least frequent to most frequent). The binning order is reversed by setting *bin\_num* to -4 instead of 4.

```

BEGIN
  dbms_data_mining_transform.CREATE_BIN_CAT(
    bin_table_name => 'bin_tbl_reverse');
  dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (
    bin_table_name => 'bin_tbl_reverse',
    data_table_name => 'mining_data',
    bin_num         => -4);
END;
/

column col format a20
SELECT col, val, bin
       FROM bin_tbl_reverse
       ORDER BY col ASC, bin ASC;

```

COL	VAL	BIN
CUST_CITY	Tokyo	1
CUST_CITY	Slidrecht	2
CUST_CITY	Haarlem	3
CUST_CITY	Diemen	4

CUST_CITY		5
CUST_POSTAL_CODE	49358	1
CUST_POSTAL_CODE	80563	2
CUST_POSTAL_CODE	74903	3
CUST_POSTAL_CODE	71349	4
CUST_POSTAL_CODE		5

## INSERT\_BIN\_NUM\_EQWIDTH Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure identifies the minimum and maximum values and computes the bin boundaries at equal intervals.

INSERT\_BIN\_NUM\_EQWIDTH computes a specified number of bins ( $n$ ) and assigns  $(max-min)/n$  values to each bin. The number of bins is the same for each column. If you want to use equi-width binning, but you want the number of bins to be calculated on a per-column basis, use the [INSERT\\_AUTOBIN\\_NUM\\_EQWIDTH Procedure](#).

INSERT\_BIN\_NUM\_EQWIDTH bins all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_EQWIDTH (
    bin_table_name      IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    bin_num             IN PLS_INTEGER DEFAULT 10,
    exclude_list       IN COLUMN_LIST DEFAULT NULL,
    round_num          IN PLS_INTEGER DEFAULT 6,
    bin_schema_name    IN VARCHAR2 DEFAULT NULL,
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–22** INSERT\_BIN\_NUM\_EQWIDTH Procedure Parameters

Parameter	Description
bin_table_name	Name of the transformation definition table for numerical binning. You can use the <a href="#">CREATE_BIN_NUM Procedure</a> to create the definition table. The following columns are required: <pre>COL      VARCHAR2(30) VAL      NUMBER BIN      VARCHAR2(4000)</pre> CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_NUM_EQWIDTH.
data_table_name	Name of the table containing the data to be transformed
bin_num	Number of bins. No binning occurs if <i>bin_num</i> is 0 or NULL. The default number of bins is 10.
exclude_list	List of numeric columns to be excluded from the binning process. If you do not specify <i>exclude_list</i> , all numeric columns in the data source are binned. The format of <i>exclude_list</i> is: <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ... 'coln')</pre>
round_num	Specifies how to round the number in the VAL column of the transformation definition table. When <i>round_num</i> is positive, it specifies the most significant digits to retain. When <i>round_num</i> is negative, it specifies the least significant digits to remove. In both cases, the result is rounded to the specified number of digits. See the Usage Notes for an example. The default value of <i>round_num</i> is 6.

**Table 45–22 (Cont.) INSERT\_BIN\_NUM\_EQWIDTH Procedure Parameters**

Parameter	Description
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Usage Notes

- The `round_num` parameter controls the rounding of column values in the transformation definition table, as follows:

**For a value of 308.162:**

```

when round_num = 1      result is 300
when round_num = 2      result is 310
when round_num = 3      result is 308
when round_num = 0      result is 308.162
when round_num = -1     result is 308.16
when round_num = -2     result is 308.2

```

- INSERT\_BIN\_NUM\_EQWIDTH ignores columns with all NULL values or only one unique value.

## Examples

In this example, INSERT\_BIN\_NUM\_EQWIDTH computes the bin boundaries for the `affinity_card` column in `mining_data_build` and inserts the transformations in a transformation definition table. The [STACK\\_BIN\\_NUM Procedure](#) creates a transformation list from the contents of the definition table. The [CREATE\\_MODEL Procedure](#) embeds the transformation list in a new model called `glm_model`.

The transformation and reverse transformation expressions embedded in `glm_model` are returned by the [GET\\_MODEL\\_TRANSFORMATIONS Function](#).

```

CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_income_level, cust_gender, affinity_card
  FROM mining_data_build;

```

```

DESCRIBE mining_data

```

```

Name                               Null?    Type
-----
CUST_ID                             NOT NULL NUMBER
CUST_INCOME_LEVEL                   VARCHAR2(30)
CUST_GENDER                          VARCHAR2(1)
AFFINITY_CARD                        NUMBER(10)

```

```

BEGIN

```

```

  dbms_data_mining_transform.CREATE_BIN_NUM(
    bin_table_name => 'bin_tbl');
  dbms_data_mining_transform.INSERT_BIN_NUM_EQWIDTH (
    bin_table_name => 'bin_tbl',
    data_table_name => 'mining_data',
    bin_num         => 4,
    exclude_list   => dbms_data_mining_transform.COLUMN_LIST('cust_id'));

```

```

END;

```

```

/

```

```

set numwidth 10

```

```
column val off
column col format a20
column bin format a10
SELECT col, val, bin FROM bin_tbl
      ORDER BY val ASC;
```

```
COL                                VAL  BIN
-----
AFFINITY_CARD                      0
AFFINITY_CARD                      .25  1
AFFINITY_CARD                      .5   2
AFFINITY_CARD                      .75  3
AFFINITY_CARD                      1    4
```

```
CREATE TABLE glmsettings(
      setting_name VARCHAR2(30),
      setting_value VARCHAR2(30));
```

```
BEGIN
      INSERT INTO glmsettings (setting_name, setting_value) VALUES
            (dbms_data_mining.algo_name,
             dbms_data_mining.algo_generalized_linear_model);
      COMMIT;
END;
/
```

```
DECLARE
      xforms  dbms_data_mining_transform.TRANSFORM_LIST;
```

```
BEGIN
      dbms_data_mining_transform.STACK_BIN_NUM (
            bin_table_name      => 'bin_tbl',
            xform_list          => xforms,
            literal_flag        => TRUE);
      dbms_data_mining.CREATE_MODEL(
            model_name          => 'glm_model',
            mining_function     => dbms_data_mining.regression,
            data_table_name     => 'mining_data',
            case_id_column_name => 'cust_id',
            target_column_name  => 'affinity_card',
            settings_table_name => 'glmsettings',
            data_schema_name    => null,
            settings_schema_name => null,
            xform_list          => xforms);
```

```
END;
/
```

```
SELECT attribute_name
      FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));
```

```
ATTRIBUTE_NAME
-----
AFFINITY_CARD
```

```
SELECT expression
      FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));
```

```
EXPRESSION
-----
CASE WHEN "AFFINITY_CARD"<0 THEN NULL WHEN "AFFINITY_CARD"<=.25 THEN 1 WHEN
"AFFINITY_CARD"<=.5 THEN 2 WHEN "AFFINITY_CARD"<=.75 THEN 3 WHEN
```

```
"AFFINITY_CARD"<=1 THEN 4 END

SELECT reverse_expression
       FROM TABLE(dbms_data_mining.GET_MODEL_TRANSFORMATIONS('glm_model'));

REVERSE_EXPRESSION
-----
DECODE("AFFINITY_CARD",4, '( .75; 1]',1, '[0; .25]',2, '( .25; .5]',3, '( .5; .75]',
NULL, '( ; 0), (1; ), NULL')
```

## INSERT\_BIN\_NUM\_QTILE Procedure

This procedure performs numerical binning and inserts the transformation definitions in a transformation definition table. The procedure calls the SQL `NTILE` function to order the data and divide it equally into the specified number of bins (quantiles).

`INSERT_BIN_NUM_QTILE` bins all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_NUM_QTILE (
  bin_table_name      IN VARCHAR2,
  data_table_name     IN VARCHAR2,
  bin_num             IN PLS_INTEGER DEFAULT 10,
  exclude_list        IN COLUMN_LIST DEFAULT NULL,
  bin_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–23** *INSERT\_BIN\_NUM\_QTILE Procedure Parameters*

Parameter	Description						
<code>bin_table_name</code>	Name of the transformation definition table for numerical binning. You can use the <a href="#">CREATE_BIN_NUM Procedure</a> to create the definition table. The following columns are required: <table border="0"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>VAL</td> <td>NUMBER</td> </tr> <tr> <td>BIN</td> <td>VARCHAR2(4000)</td> </tr> </table> <code>CREATE_BIN_NUM</code> creates an additional column, <code>ATT</code> , which may be used for specifying nested attributes. This column is not used by <code>INSERT_BIN_NUM_QTILE</code> .	COL	VARCHAR2(30)	VAL	NUMBER	BIN	VARCHAR2(4000)
COL	VARCHAR2(30)						
VAL	NUMBER						
BIN	VARCHAR2(4000)						
<code>data_table_name</code>	Name of the table containing the data to be transformed						
<code>bin_num</code>	Number of bins. No binning occurs if <code>bin_num</code> is 0 or <code>NULL</code> . The default number of bins is 10.						
<code>exclude_list</code>	List of numeric columns to be excluded from the binning process. If you do not specify <code>exclude_list</code> , all numeric columns in the data source are binned. The format of <code>exclude_list</code> is: <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')</code>						
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.						
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.						

### Usage Notes

- After dividing the data into quantiles, the `NTILE` function distributes any remainder values one for each quantile, starting with the first. See *Oracle Database SQL Language Reference* for details.
- Columns with all `NULL` values are ignored by `INSERT_BIN_NUM_QTILE`.



## Examples

In this example, `INSERT_BIN_NUM_QTILE` computes the bin boundaries for the `cust_year_of_birth` and `cust_credit_limit` columns in `sh.customers` and inserts the transformations in a transformation definition table. The [STACK\\_BIN\\_NUM Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in `STACK_VIEW`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
    FROM sh.customers;
```

```
DESCRIBE mining_data
```

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_YEAR_OF_BIRTH	NOT NULL	NUMBER(4)
CUST_CREDIT_LIMIT		NUMBER
CUST_CITY	NOT NULL	VARCHAR2(30)

```
BEGIN
    dbms_data_mining_transform.CREATE_BIN_NUM(
        bin_table_name => 'bin_tbl');
    dbms_data_mining_transform.INSERT_BIN_NUM_QTILE (
        bin_table_name => 'bin_tbl',
        data_table_name => 'mining_data',
        bin_num => 3,
        exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
```

```
/
```

```
set numwidth 8
column val off
column col format a20
column bin format a10
SELECT col, val, bin
    FROM bin_tbl
    ORDER BY col ASC, val ASC;
```

COL	VAL	BIN
CUST_CREDIT_LIMIT	1500	
CUST_CREDIT_LIMIT	3000	1
CUST_CREDIT_LIMIT	9000	2
CUST_CREDIT_LIMIT	15000	3
CUST_YEAR_OF_BIRTH	1913	
CUST_YEAR_OF_BIRTH	1949	1
CUST_YEAR_OF_BIRTH	1965	2
CUST_YEAR_OF_BIRTH	1990	3

```
DECLARE
    xforms dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_BIN_NUM (
        bin_table_name => 'bin_tbl',
        xform_list => xforms);
    dbms_data_mining_transform.XFORM_STACK (
```

```

        xform_list          => xforms,
        data_table_name     => 'mining_data',
        xform_view_name     => 'stack_view');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name in 'STACK_VIEW';

TEXT
-----
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH"<1913 THEN NULL WHEN "CUST_YEAR_O
F_BIRTH"<=1949 THEN '1' WHEN "CUST_YEAR_OF_BIRTH"<=1965 THEN '2' WHEN "CUST_YEAR
_OF_BIRTH"<=1990 THEN '3' END "CUST_YEAR_OF_BIRTH",CASE WHEN "CUST_CREDIT_LIMIT"
<1500 THEN NULL WHEN "CUST_CREDIT_LIMIT"<=3000 THEN '1' WHEN "CUST_CREDIT_LIMIT"
<=9000 THEN '2' WHEN "CUST_CREDIT_LIMIT"<=15000 THEN '3' END "CUST_CREDIT_LIMIT"
,"CUST_CITY" FROM mining_data

```

## INSERT\_BIN\_SUPER Procedure

This procedure performs numerical and categorical binning and inserts the transformation definitions in transformation definition tables. The procedure computes bin boundaries based on intrinsic relationships between predictors and a target.

INSERT\_BIN\_SUPER uses an intelligent binning technique known as **supervised binning**. It builds a single-predictor decision tree and derives the bin boundaries from splits within the tree.

INSERT\_BIN\_SUPER bins all the VARCHAR2, CHAR, NUMBER, and FLOAT columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_BIN_SUPER (
  num_table_name      IN VARCHAR2,
  cat_table_name      IN VARCHAR2,
  data_table_name     IN VARCHAR2,
  target_column_name  IN VARCHAR2,
  max_bin_num         IN PLS_INTEGER DEFAULT 1000,
  exclude_list        IN COLUMN_LIST  DEFAULT NULL,
  num_schema_name     IN VARCHAR2     DEFAULT NULL,
  cat_schema_name     IN VARCHAR2     DEFAULT NULL,
  data_schema_name    IN VARCHAR2     DEFAULT NULL,
  rem_table_name      IN VARCHAR2     DEFAULT NULL,
  rem_schema_name     IN VARCHAR2     DEFAULT NULL);
```

### Parameters

**Table 45–24** INSERT\_BIN\_SUPER Procedure Parameters

Parameter	Description						
num_table_name	Name of the transformation definition table for numerical binning. You can use the <a href="#">CREATE_BIN_NUM Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2 (30)</td> </tr> <tr> <td>VAL</td> <td>VNUMBER</td> </tr> <tr> <td>BIN</td> <td>VARCHAR2 (4000)</td> </tr> </table> CREATE_BIN_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_BIN_SUPER.	COL	VARCHAR2 (30)	VAL	VNUMBER	BIN	VARCHAR2 (4000)
COL	VARCHAR2 (30)						
VAL	VNUMBER						
BIN	VARCHAR2 (4000)						
cat_table_name	Name of the transformation definition table for categorical binning. You can use the <a href="#">CREATE_BIN_CAT Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2 (30)</td> </tr> <tr> <td>VAL</td> <td>VARCHAR2 (4000)</td> </tr> <tr> <td>BIN</td> <td>VARCHAR2 (4000)</td> </tr> </table> CREATE_BIN_CAT creates an additional column, ATT, which is used for specifying nested attributes. This column is not used by INSERT_BIN_SUPER.	COL	VARCHAR2 (30)	VAL	VARCHAR2 (4000)	BIN	VARCHAR2 (4000)
COL	VARCHAR2 (30)						
VAL	VARCHAR2 (4000)						
BIN	VARCHAR2 (4000)						
data_table_name	Name of the table containing the data to be transformed						
target_column_name	Name of a column to be used as the target for the decision tree models						
max_bin_num	The maximum number of bins. The default is 1000.						

**Table 45–24 (Cont.) INSERT\_BIN\_SUPER Procedure Parameters**

Parameter	Description
<code>exclude_list</code>	List of columns to be excluded from the binning process. If you do not specify <code>exclude_list</code> , all CHAR, VARCHAR2, NUMBER, and FLOAT columns in the data source are binned.  The format of <code>exclude_list</code> is:  <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')</code>
<code>num_schema_name</code>	Schema of <code>num_table_name</code> . If no schema is specified, the current schema is used.
<code>cat_schema_name</code>	Schema of <code>cat_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>rem_table_name</code>	Name of a column removal definition table. The table must have the columns described in " <a href="#">CREATE_COL_REM Procedure</a> " on page 45-25. You can use <code>CREATE_COL_REM</code> to create the table. See Usage Notes.
<code>rem_schema_name</code>	Schema of <code>rem_table_name</code> . If no schema is specified, the current schema is used.

## Usage Notes

- Columns that have no significant splits are not binned. You can remove the unbinned columns from the mining data by specifying a column removal definition table. If you do not specify a column removal definition table, the unbinned columns remain in the mining data.
- See *Oracle Data Mining Concepts* to learn more about decision trees in Oracle Data Mining

## Examples

In this example, `INSERT_BIN_SUPER` computes the bin boundaries for predictors of `cust_credit_limit` and inserts the transformations in transformation definition tables. One predictor is numerical, the other is categorical. (`INSERT_BIN_SUPER` determines that the `cust_postal_code` column is not a significant predictor.) `STACK` procedures create transformation lists from the contents of the definition tables.

The SQL expressions that compute the transformations are shown in the views `MINING_DATA_STACK_NUM` and `MINING_DATA_STACK_CAT`. The views are for display purposes only; they cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_marital_status,
         cust_postal_code, cust_credit_limit
  FROM sh.customers;
```

```
DESCRIBE mining_data
```

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_YEAR_OF_BIRTH	NOT NULL	NUMBER(4)
CUST_MARITAL_STATUS		VARCHAR2(20)
CUST_POSTAL_CODE	NOT NULL	VARCHAR2(10)
CUST_CREDIT_LIMIT		NUMBER

```

BEGIN
  dbms_data_mining_transform.CREATE_BIN_NUM(
    bin_table_name => 'bin_num_tbl');
  dbms_data_mining_transform.CREATE_BIN_CAT(
    bin_table_name => 'bin_cat_tbl');
  dbms_data_mining_transform.CREATE_COL_REM(
    rem_table_name => 'rem_tbl');
END;
/

BEGIN
  COMMIT;
  dbms_data_mining_transform.INSERT_BIN_SUPER (
    num_table_name => 'bin_num_tbl',
    cat_table_name => 'bin_cat_tbl',
    data_table_name => 'mining_data',
    target_column_name => 'cust_credit_limit',
    max_bin_num => 4,
    exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'),
    num_schema_name => 'dmuser',
    cat_schema_name => 'dmuser',
    data_schema_name => 'dmuser',
    rem_table_name => 'rem_tbl',
    rem_schema_name => 'dmuser');
  COMMIT;
END;
/

```

```

set numwidth 8
column val off
SELECT col, val, bin FROM bin_num_tbl
      ORDER BY bin ASC;

```

COL	VAL	BIN
-----	-----	-----
CUST_YEAR_OF_BIRTH	1923.5	1
CUST_YEAR_OF_BIRTH	1923.5	1
CUST_YEAR_OF_BIRTH	1945.5	2
CUST_YEAR_OF_BIRTH	1980.5	3
CUST_YEAR_OF_BIRTH		4

```

column val on
column val format a20
SELECT col, val, bin FROM bin_cat_tbl
      ORDER BY bin ASC;

```

COL	VAL	BIN
-----	-----	-----
CUST_MARITAL_STATUS	married	1
CUST_MARITAL_STATUS	single	2
CUST_MARITAL_STATUS	Mar-AF	3
CUST_MARITAL_STATUS	Mabsent	3
CUST_MARITAL_STATUS	Divorc.	3
CUST_MARITAL_STATUS	Married	3
CUST_MARITAL_STATUS	Widowed	3
CUST_MARITAL_STATUS	NeverM	3
CUST_MARITAL_STATUS	Separ.	3
CUST_MARITAL_STATUS	divorced	4
CUST_MARITAL_STATUS	widow	4

```

SELECT col from rem_tbl;

COL
-----
CUST_POSTAL_CODE

DECLARE
  xforms_num      dbms_data_mining_transform.TRANSFORM_LIST;
  xforms_cat      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_NUM (
    bin_table_name => 'bin_num_tbl',
    xform_list     => xforms_num);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list     => xforms_num,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_num');
  dbms_data_mining_transform.STACK_BIN_CAT (
    bin_table_name => 'bin_cat_tbl',
    xform_list     => xforms_cat);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list     => xforms_cat,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_cat');

  END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK_NUM';

TEXT
-----
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH"<1923.5 THEN '1' WHEN "CUST_YEAR_
OF_BIRTH"<=1923.5 THEN '1' WHEN "CUST_YEAR_OF_BIRTH"<=1945.5 THEN '2' WHEN "CUST
_YEAR_OF_BIRTH"<=1980.5 THEN '3' WHEN "CUST_YEAR_OF_BIRTH" IS NOT NULL THEN '4'
END "CUST_YEAR_OF_BIRTH", "CUST_MARITAL_STATUS", "CUST_POSTAL_CODE", "CUST_CREDIT_L
IMIT" FROM mining_data

SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK_CAT';

TEXT
-----
SELECT "CUST_ID", "CUST_YEAR_OF_BIRTH",DECODE("CUST_MARITAL_STATUS", 'Divorc.', '3'
, 'Mabsent', '3', 'Mar-AF', '3', 'Married', '3', 'NeverM', '3', 'Separ.', '3', 'Widowed', '3
', 'divorced', '4', 'married', '1', 'single', '2', 'widow', '4') "CUST_MARITAL_STATUS", "
CUST_POSTAL_CODE", "CUST_CREDIT_LIMIT" FROM mining_data

```

## INSERT\_CLIP\_TRIM\_TAIL Procedure

This procedure replaces numeric outliers with nulls and inserts the transformation definitions in a transformation definition table.

INSERT\_CLIP\_TRIM\_TAIL computes the boundaries of the data based on a specified percentage. It removes the values that fall outside the boundaries (tail values) from the data. If you wish to replace the tail values instead of removing them, use the [INSERT\\_CLIP\\_WINSOR\\_TAIL Procedure](#).

INSERT\_CLIP\_TRIM\_TAIL clips all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_TRIM_TAIL (
  clip_table_name      IN VARCHAR2,
  data_table_name      IN VARCHAR2,
  tail_frac            IN NUMBER DEFAULT 0.025,
  exclude_list         IN COLUMN_LIST DEFAULT NULL,
  clip_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–25** INSERT\_CLIP\_TRIM\_TAIL Procedure Parameters

Parameter	Description										
clip_table_name	Name of the transformation definition table for numerical clipping. You can use the <a href="#">CREATE_CLIP Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>LCUT</td> <td>NUMBER</td> </tr> <tr> <td>LVAL</td> <td>NUMBER</td> </tr> <tr> <td>RCUT</td> <td>NUMBER</td> </tr> <tr> <td>RVAL</td> <td>NUMBER</td> </tr> </table> CREATE_CLIP creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_CLIP_TRIM_TAIL.	COL	VARCHAR2(30)	LCUT	NUMBER	LVAL	NUMBER	RCUT	NUMBER	RVAL	NUMBER
COL	VARCHAR2(30)										
LCUT	NUMBER										
LVAL	NUMBER										
RCUT	NUMBER										
RVAL	NUMBER										
data_table_name	Name of the table containing the data to be transformed										
tail_frac	The percentage of non-null values to be designated as outliers at each end of the data. For example, if <i>tail_frac</i> is .01, then 1% of the data at the low end and 1% of the data at the high end will be treated as outliers.  If <i>tail_frac</i> is greater than or equal to .5, no clipping occurs. The default value of <i>tail_frac</i> is 0.025.										
exclude_list	List of NUMBER columns to be excluded from the clipping process. If you do not specify <i>exclude_list</i> , all NUMBER columns in the data are clipped.  The format of <i>exclude_list</i> is:  dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')										
clip_schema_name	Schema of <i>clip_table_name</i> . If no schema is specified, the current schema is used.										

**Table 45–25 (Cont.) INSERT\_CLIP\_TRIM\_TAIL Procedure Parameters**

Parameter	Description
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Usage Notes

The `DBMS_DATA_MINING_TRANSFORM` package provides two clipping procedures: `INSERT_CLIP_TRIM_TAIL` and `INSERT_CLIP_WINSOR_TAIL`. Both procedures compute the boundaries as follows:

- Count the number of non-null values, ***n***, and sort them in ascending order
- Calculate the number of outliers, ***t***, as ***n\*tail\_frac***
- Define the lower boundary ***lcut*** as the value at position ***1+floor(t)***
- Define the upper boundary ***rcut*** as the value at position ***n-floor(t)***  
(The SQL `FLOOR` function returns the largest integer less than or equal to ***t***.)
- All values that are  $\leq$  ***lcut*** or  $\geq$  ***rcut*** are designated as outliers.

`INSERT_CLIP_TRIM_TAIL` replaces the outliers with nulls, effectively removing them from the data.

`INSERT_CLIP_WINSOR_TAIL` assigns ***lcut*** to the low outliers and ***rcut*** to the high outliers.

## Examples

In this example, `INSERT_CLIP_TRIM_TAIL` trims 10% of the data in two columns (5% from the high end and 5% from the low end) and inserts the transformations in a transformation definition table. The [STACK\\_CLIP Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the trimming is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
    FROM sh.customers;

DESCRIBE mining_data
Name                               Null?    Type
-----
CUST_ID                            NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                 NOT NULL NUMBER(4)
CUST_CREDIT_LIMIT                   NUMBER
CUST_CITY                           NOT NULL VARCHAR2(30)

BEGIN
    dbms_data_mining_transform.CREATE_CLIP(
        clip_table_name => 'clip_tbl');
    dbms_data_mining_transform.INSERT_CLIP_TRIM_TAIL(
        clip_table_name => 'clip_tbl',
        data_table_name => 'mining_data',
        tail_frac       => 0.05,
        exclude_list    => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/
```



```

SELECT col, lcut, lval, rcut, rval
      FROM clip_tbl
      ORDER BY col ASC;

```

COL	LCUT	LVAL	RCUT	RVAL
CUST_CREDIT_LIMIT	1500		11000	
CUST_YEAR_OF_BIRTH	1934		1982	

```

DECLARE
  xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_CLIP (
    clip_table_name => 'clip_tbl',
    xform_list      => xforms);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => xforms,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack');
END;
/

```

```

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

```

```

TEXT

```

```

-----
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH" < 1934 THEN NULL WHEN "CUST_YEAR
_OF_BIRTH" > 1982 THEN NULL ELSE "CUST_YEAR_OF_BIRTH" END "CUST_YEAR_OF_BIRTH",C
ASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN NULL WHEN "CUST_CREDIT_LIMIT" > 11000 T
HEN NULL ELSE "CUST_CREDIT_LIMIT" END "CUST_CREDIT_LIMIT","CUST_CITY" FROM minin
g_data

```

## INSERT\_CLIP\_WINSOR\_TAIL Procedure

This procedure replaces numeric outliers with the upper or lower boundary values. It inserts the transformation definitions in a transformation definition table.

INSERT\_CLIP\_WINSOR\_TAIL computes the boundaries of the data based on a specified percentage. It replaces the values that fall outside the boundaries (tail values) with the related boundary value. If you wish to set tail values to null, use the [INSERT\\_CLIP\\_TRIM\\_TAIL Procedure](#).

INSERT\_CLIP\_WINSOR\_TAIL clips all the NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_CLIP_WINSOR_TAIL (
  clip_table_name      IN VARCHAR2,
  data_table_name      IN VARCHAR2,
  tail_frac            IN NUMBER DEFAULT 0.025,
  exclude_list         IN COLUMN_LIST DEFAULT NULL,
  clip_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–26** INSERT\_CLIP\_WINSOR\_TAIL Procedure Parameters

Parameter	Description										
clip_table_name	Name of the transformation definition table for numerical clipping. You can use the <a href="#">CREATE_CLIP Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>LCUT</td> <td>NUMBER</td> </tr> <tr> <td>LVAL</td> <td>NUMBER</td> </tr> <tr> <td>RCUT</td> <td>NUMBER</td> </tr> <tr> <td>RVAL</td> <td>NUMBER</td> </tr> </table> CREATE_CLIP creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_CLIP_WINSOR_TAIL.	COL	VARCHAR2(30)	LCUT	NUMBER	LVAL	NUMBER	RCUT	NUMBER	RVAL	NUMBER
COL	VARCHAR2(30)										
LCUT	NUMBER										
LVAL	NUMBER										
RCUT	NUMBER										
RVAL	NUMBER										
data_table_name	Name of the table containing the data to be transformed										
tail_frac	The percentage of non-null values to be designated as outliers at each end of the data. For example, if <i>tail_frac</i> is .01, then 1% of the data at the low end and 1% of the data at the high end will be treated as outliers.  If <i>tail_frac</i> is greater than or equal to .5, no clipping occurs. The default value of <i>tail_frac</i> is 0.025.										
exclude_list	List of NUMBER columns to be excluded from the clipping process. If you do not specify <i>exclude_list</i> , all NUMBER columns in the data are clipped.  The format of <i>exclude_list</i> is: <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ... 'coln')</pre>										
clip_schema_name	Schema of <i>clip_table_name</i> . If no schema is specified, the current schema is used.										

**Table 45–26 (Cont.) INSERT\_CLIP\_WINSOR\_TAIL Procedure Parameters**

Parameter	Description
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Usage Notes

The DBMS\_DATA\_MINING\_TRANSFORM package provides two clipping procedures: INSERT\_CLIP\_WINSOR\_TAIL and INSERT\_CLIP\_TRIM\_TAIL. Both procedures compute the boundaries as follows:

- Count the number of non-null values, ***n***, and sort them in ascending order
- Calculate the number of outliers, ***t***, as ***n\*tail\_frac***
- Define the lower boundary ***lcut*** as the value at position ***1+floor(t)***
- Define the upper boundary ***rcut*** as the value at position ***n-floor(t)***  
(The SQL FLOOR function returns the largest integer less than or equal to ***t***.)
- All values that are ***<= lcut*** or ***=> rcut*** are designated as outliers.

INSERT\_CLIP\_WINSOR\_TAIL assigns ***lcut*** to the low outliers and ***rcut*** to the high outliers.

INSERT\_CLIP\_TRIM\_TAIL replaces the outliers with nulls, effectively removing them from the data.

## Examples

In this example, INSERT\_CLIP\_WINSOR\_TAIL winsorizes 10% of the data in two columns (5% from the high end, and 5% from the low end) and inserts the transformations in a transformation definition table. The [STACK\\_CLIP Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view MINING\_DATA\_STACK. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_credit_limit, cust_city
    FROM sh.customers;

describe mining_data
Name                                         Null?    Type
-----
CUST_ID                                     NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                         NOT NULL NUMBER(4)
CUST_CREDIT_LIMIT                           NUMBER
CUST_CITY                                   NOT NULL VARCHAR2(30)

BEGIN
  dbms_data_mining_transform.CREATE_CLIP(
    clip_table_name => 'clip_tbl');
  dbms_data_mining_transform.INSERT_CLIP_WINSOR_TAIL(
    clip_table_name => 'clip_tbl',
    data_table_name => 'mining_data',
    tail_frac       => 0.05,
    exclude_list    => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/
```

```

SELECT col, lcut, lval, rcut, rval FROM clip_tbl
      ORDER BY col ASC;
COL                                LCUT      LVAL      RCUT      RVAL
-----
CUST_CREDIT_LIMIT                  1500      1500      11000     11000
CUST_YEAR_OF_BIRTH                 1934      1934      1982      1982

DECLARE
  xforms          dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_CLIP (
    clip_table_name => 'clip_tbl',
    xform_list      => xforms);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => xforms,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack');
END;
/

set long 3000
SQL> SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
-----
SELECT "CUST_ID",CASE WHEN "CUST_YEAR_OF_BIRTH" < 1934 THEN 1934 WHEN "CUST_YEAR
_OF_BIRTH" > 1982 THEN 1982 ELSE "CUST_YEAR_OF_BIRTH" END "CUST_YEAR_OF_BIRTH",C
ASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN 1500 WHEN "CUST_CREDIT_LIMIT" > 11000 T
HEN 11000 ELSE "CUST_CREDIT_LIMIT" END "CUST_CREDIT_LIMIT","CUST_CITY" FROM mini
ng_data

```

## INSERT\_MISS\_CAT\_MODE Procedure

This procedure replaces missing categorical values with the value that occurs most frequently in the column (the mode). It inserts the transformation definitions in a transformation definition table.

INSERT\_MISS\_CAT\_MODE replaces missing values in all VARCHAR2 and CHAR columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE (
    miss_table_name    IN VARCHAR2,
    data_table_name    IN VARCHAR2,
    exclude_list       IN COLUMN_LIST DEFAULT NULL,
    miss_schema_name   IN VARCHAR2 DEFAULT NULL,
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–27** INSERT\_MISS\_CAT\_MODE Procedure Parameters

Parameter	Description
miss_table_name	Name of the transformation definition table for categorical missing value treatment. You can use the <a href="#">CREATE_MISS_CAT Procedure</a> to create the definition table. The following columns are required: COL            VARCHAR2 (30) VAL            VARCHAR2 (4000) CREATE_MISS_CAT creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_MISS_CAT_MODE.
data_table_name	Name of the table containing the data to be transformed
exclude_list	List of VARCHAR2 and CHAR columns to be excluded from missing value treatment. If you do not specify <i>exclude_list</i> , all VARCHAR2 and CHAR columns are transformed. The format of <i>exclude_list</i> is: dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')
miss_schema_name	Schema of <i>miss_table_name</i> . If no schema is specified, the current schema is used.
data_schema_name	Schema of <i>data_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

If you wish to replace categorical missing values with a value other than the mode, you can edit the transformation definition table.

**See Also:** *Oracle Data Mining Application Developer's Guide* for information about default missing value treatment in Oracle Data Mining

## Example

In this example, `INSERT_MISS_CAT_MODE` computes missing value treatment for `cust_city` and inserts the transformation in a transformation definition table. The [STACK\\_MISS\\_CAT Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_city
    FROM sh.customers;

describe mining_data
Name                               Null?    Type
-----
CUST_ID                             NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                   NOT NULL NUMBER(4)
CUST_CITY                             NOT NULL VARCHAR2(30)

BEGIN
    dbms_data_mining_transform.create_miss_cat(
        miss_table_name => 'missc_tbl');
    dbms_data_mining_transform.insert_miss_cat_mode(
        miss_table_name => 'missc_tbl',
        data_table_name => 'mining_data');
END;
/

SELECT stats_mode(cust_city) FROM mining_data;

STATS_MODE(CUST_CITY)
-----
Los Angeles

SELECT col, val
    from missc_tbl;

COL                                VAL
-----
CUST_CITY                           Los Angeles

DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_MISS_CAT (
        miss_table_name => 'missc_tbl',
        xform_list      => xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => xforms,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';

TEXT
```

```
-----  
SELECT "CUST_ID", "CUST_YEAR_OF_BIRTH", NVL("CUST_CITY", 'Los Angeles') "CUST_CITY"  
FROM mining_data
```

## INSERT\_MISS\_NUM\_MEAN Procedure

This procedure replaces missing numerical values with the average (the mean) and inserts the transformation definitions in a transformation definition table.

INSERT\_MISS\_NUM\_MEAN replaces missing values in all NUMBER and FLOAT columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (
    miss_table_name    IN VARCHAR2,
    data_table_name    IN VARCHAR2,
    exclude_list       IN COLUMN_LIST DEFAULT NULL,
    round_num          IN PLS_INTEGER DEFAULT 6,
    miss_schema_name   IN VARCHAR2 DEFAULT NULL,
    data_schema_name   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–28** INSERT\_MISS\_NUM\_MEAN Procedure Parameters

Parameter	Description				
miss_table_name	Name of the transformation definition table for numerical missing value treatment. You can use the <a href="#">CREATE_MISS_NUM Procedure</a> to create the definition table.  The following columns are required by INSERT_MISS_NUM_MEAN:  <table border="1"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>VAL</td> <td>NUMBER</td> </tr> </table> CREATE_MISS_NUM creates an additional column, ATT, which may be used for specifying nested attributes. This column is not used by INSERT_MISS_NUM_MEAN.	COL	VARCHAR2(30)	VAL	NUMBER
COL	VARCHAR2(30)				
VAL	NUMBER				
data_table_name	Name of the table containing the data to be transformed				
exclude_list	List of NUMBER columns to be excluded from missing value treatment. If you do not specify <i>exclude_list</i> , all NUMBER columns are transformed.  The format of <i>exclude_list</i> is:  <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ... 'coln')</pre>				
round_num	The number of significant digits to use for the mean.  The default number is 6.				
miss_schema_name	Schema of <i>miss_table_name</i> . If no schema is specified, the current schema is used.				
data_schema_name	Schema of <i>data_table_name</i> . If no schema is specified, the current schema is used.				

### Usage Notes

If you wish to replace numerical missing values with a value other than the mean, you can edit the transformation definition table.

**See Also:** *Oracle Data Mining Application Developer's Guide* for information about default missing value treatment in Oracle Data Mining



## Example

In this example, `INSERT_MISS_NUM_MEAN` computes missing value treatment for `cust_year_of_birth` and inserts the transformation in a transformation definition table. The [STACK\\_MISS\\_NUM Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_city
  FROM sh.customers;

DESCRIBE mining_data
Name                                                    Null?    Type
-----
CUST_ID                                                NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                                    NOT NULL NUMBER(4)
CUST_CITY                                              NOT NULL VARCHAR2(30)

BEGIN
  dbms_data_mining_transform.create_miss_num(
    miss_table_name => 'missn_tbl');
  dbms_data_mining_transform.insert_miss_num_mean(
    miss_table_name => 'missn_tbl',
    data_table_name => 'mining_data',
    exclude_list   => DBMS_DATA_MINING_TRANSFORM.COLUMN_LIST('cust_id'));
END;
/

set numwidth 4
column val off
SELECT col, val
  FROM missn_tbl;

COL                VAL
-----
CUST_YEAR_OF_BIRTH 1957

SELECT avg(cust_year_of_birth) FROM mining_data;

AVG(CUST_YEAR_OF_BIRTH)
-----
                        1957

DECLARE
  xforms          dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_MISS_NUM (
    miss_table_name => 'missn_tbl',
    xform_list      => xforms);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => xforms,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack');
END;
/

set long 3000
```

```
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';
```

TEXT

```
-----  
SELECT "CUST_ID", NVL("CUST_YEAR_OF_BIRTH", 1957.4) "CUST_YEAR_OF_BIRTH", "CUST_CIT  
Y" FROM mining_data
```

## INSERT\_NORM\_LIN\_MINMAX Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table. `INSERT_NORM_LIN_MINMAX` computes the minimum and maximum values from the data and sets the value of *shift* and *scale* as follows:

```
shift = min
scale = max - min
```

Normalization is computed as:

```
x_new = (x_old - shift)/scale
```

`INSERT_NORM_LIN_MINMAX` rounds the value of *scale* to a specified number of significant digits before storing it in the transformation definition table.

`INSERT_NORM_LIN_MINMAX` normalizes all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_MINMAX (
    norm_table_name      IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    exclude_list         IN COLUMN_LIST DEFAULT NULL,
    round_num            IN PLS_INTEGER DEFAULT 6,
    norm_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–29** INSERT\_NORM\_LIN\_MINMAX Procedure Parameters

Parameter	Description						
<code>norm_table_name</code>	Name of the transformation definition table for linear normalization. You can use the <a href="#">CREATE_NORM_LIN Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>SHIFT</td> <td>NUMBER</td> </tr> <tr> <td>SCALE</td> <td>NUMBER</td> </tr> </table> <code>CREATE_NORM_LIN</code> creates an additional column, <code>ATT</code> , which may be used for specifying nested attributes. This column is not used by <code>INSERT_NORM_LIN_MINMAX</code> .	COL	VARCHAR2(30)	SHIFT	NUMBER	SCALE	NUMBER
COL	VARCHAR2(30)						
SHIFT	NUMBER						
SCALE	NUMBER						
<code>data_table_name</code>	Name of the table containing the data to be transformed						
<code>exclude_list</code>	List of <code>NUMBER</code> columns to be excluded from normalization. If you do not specify <i>exclude_list</i> , all <code>NUMBER</code> columns are transformed. <p>The format of <i>exclude_list</i> is:</p> <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ...'coln')</pre>						
<code>round_num</code>	The number of significant digits to use for the minimum and maximum. The default number is 6.						
<code>norm_schema_name</code>	Schema of <i>norm_table_name</i> . If no schema is specified, the current schema is used.						

**Table 45–29 (Cont.) INSERT\_NORM\_LIN\_MINMAX Procedure Parameters**

Parameter	Description
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Examples

In this example, `INSERT_NORM_LIN_MINMAX` normalizes the `cust_year_of_birth` column and inserts the transformation in a transformation definition table. The [STACK\\_NORM\\_LIN Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_gender, cust_year_of_birth
    FROM sh.customers;

describe mining_data
Name                                     Null?    Type
-----
CUST_ID                                 NOT NULL NUMBER
CUST_GENDER                             NOT NULL CHAR(1)
CUST_YEAR_OF_BIRTH                       NOT NULL NUMBER(4)

BEGIN
    dbms_data_mining_transform.CREATE_NORM_LIN(
        norm_table_name => 'norm_tbl');
    dbms_data_mining_transform.INSERT_NORM_LIN_MINMAX(
        norm_table_name => 'norm_tbl',
        data_table_name => 'mining_data',
        exclude_list   => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
        round_num      => 3);
END;
/

SELECT col, shift, scale FROM norm_tbl;

COL                                     SHIFT    SCALE
-----
CUST_YEAR_OF_BIRTH                       1910     77

DECLARE
    xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_NORM_LIN (
        norm_table_name => 'norm_tbl',
        xform_list      => xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => xforms,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';
```

TEXT

-----  
SELECT "CUST\_ID", "CUST\_GENDER", ("CUST\_YEAR\_OF\_BIRTH"-1910)/77 "CUST\_YEAR\_OF\_BIRTH"  
FROM mining\_data

## INSERT\_NORM\_LIN\_SCALE Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table. `INSERT_NORM_LIN_SCALE` computes the minimum and maximum values from the data and sets the value of *shift* and *scale* as follows:

```
shift = 0
scale = max(abs(max), abs(min))
```

Normalization is computed as:

```
x_new = (x_old) / scale
```

`INSERT_NORM_LIN_SCALE` rounds the value of *scale* to a specified number of significant digits before storing it in the transformation definition table.

`INSERT_NORM_LIN_SCALE` normalizes all the `NUMBER` and `FLOAT` columns in the data source unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_SCALE (
    norm_table_name      IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    exclude_list         IN COLUMN_LIST DEFAULT NULL,
    round_num            IN PLS_INTEGER DEFAULT 6,
    norm_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–30** INSERT\_NORM\_LIN\_SCALE Procedure Parameters

Parameter	Description						
<code>norm_table_name</code>	Name of the transformation definition table for linear normalization. You can use the <a href="#">CREATE_NORM_LIN Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>SHIFT</td> <td>NUMBER</td> </tr> <tr> <td>SCALE</td> <td>NUMBER</td> </tr> </table> <code>CREATE_NORM_LIN</code> creates an additional column, <code>ATT</code> , which may be used for specifying nested attributes. This column is not used by <code>INSERT_NORM_LIN_SCALE</code> .	COL	VARCHAR2(30)	SHIFT	NUMBER	SCALE	NUMBER
COL	VARCHAR2(30)						
SHIFT	NUMBER						
SCALE	NUMBER						
<code>data_table_name</code>	Name of the table containing the data to be transformed						
<code>exclude_list</code>	List of <code>NUMBER</code> columns to be excluded from normalization. If you do not specify <i>exclude_list</i> , all <code>NUMBER</code> columns are transformed. <p>The format of <i>exclude_list</i> is:</p> <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ... 'coln')</pre>						
<code>round_num</code>	The number of significant digits to use for <i>scale</i> . The default number is 6.						
<code>norm_schema_name</code>	Schema of <i>norm_table_name</i> . If no schema is specified, the current schema is used.						

**Table 45–30 (Cont.) INSERT\_NORM\_LIN\_SCALE Procedure Parameters**

Parameter	Description
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Examples

In this example, `INSERT_NORM_LIN_SCALE` normalizes the `cust_year_of_birth` column and inserts the transformation in a transformation definition table. The [STACK\\_NORM\\_LIN Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_gender, cust_year_of_birth
    FROM sh.customers;

DESCRIBE mining_data
Name                                     Null?      Type
-----
CUST_ID                                 NOT NULL  NUMBER
CUST_GENDER                             NOT NULL  CHAR(1)
CUST_YEAR_OF_BIRTH                       NOT NULL  NUMBER(4)

BEGIN
    dbms_data_mining_transform.CREATE_NORM_LIN(
        norm_table_name => 'norm_tbl');
    dbms_data_mining_transform.INSERT_NORM_LIN_SCALE(
        norm_table_name => 'norm_tbl',
        data_table_name => 'mining_data',
        exclude_list   => dbms_data_mining_transform.COLUMN_LIST('cust_id'),
        round_num      => 3);
END;
/

SELECT col, shift, scale FROM norm_tbl;

COL                SHIFT SCALE
-----
CUST_YEAR_OF_BIRTH      0  1990

DECLARE
    xforms          dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.STACK_NORM_LIN (
        norm_table_name => 'norm_tbl',
        xform_list      => xforms);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => xforms,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack');
END;
/

set long 3000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';
```

TEXT

-----  
SELECT "CUST\_ID", "CUST\_GENDER", ("CUST\_YEAR\_OF\_BIRTH"-0)/1990 "CUST\_YEAR\_OF\_BIRTH"  
" FROM mining\_data



## INSERT\_NORM\_LIN\_ZSCORE Procedure

This procedure performs linear normalization and inserts the transformation definitions in a transformation definition table. `INSERT_NORM_LIN_ZSCORE` computes the mean and the standard deviation from the data and sets the value of *shift* and *scale* as follows:

```
shift = mean
scale = stddev
```

Normalization is computed as:

$$x_{new} = (x_{old} - shift) / scale$$

`INSERT_NORM_LIN_ZSCORE` rounds the value of *scale* to a specified number of significant digits before storing it in the transformation definition table.

`INSERT_NORM_LIN_ZSCORE` normalizes all the `NUMBER` and `FLOAT` columns in the data unless you specify a list of columns to ignore.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.INSERT_NORM_LIN_ZSCORE (
    norm_table_name      IN VARCHAR2,
    data_table_name      IN VARCHAR2,
    exclude_list         IN COLUMN_LIST DEFAULT NULL,
    round_num            IN PLS_INTEGER DEFAULT 6,
    norm_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–31** INSERT\_NORM\_LIN\_ZSCORE Procedure Parameters

Parameter	Description						
<code>norm_table_name</code>	Name of the transformation definition table for linear normalization. You can use the <a href="#">CREATE_NORM_LIN Procedure</a> to create the definition table. The following columns are required: <table border="0" style="margin-left: 20px;"> <tr> <td>COL</td> <td>VARCHAR2(30)</td> </tr> <tr> <td>SHIFT</td> <td>NUMBER</td> </tr> <tr> <td>SCALE</td> <td>NUMBER</td> </tr> </table> <code>CREATE_NORM_LIN</code> creates an additional column, <code>ATT</code> , which may be used for specifying nested attributes. This column is not used by <code>INSERT_NORM_LIN_ZSCORE</code> .	COL	VARCHAR2(30)	SHIFT	NUMBER	SCALE	NUMBER
COL	VARCHAR2(30)						
SHIFT	NUMBER						
SCALE	NUMBER						
<code>data_table_name</code>	Name of the table containing the data to be transformed						
<code>exclude_list</code>	List of <code>NUMBER</code> columns to be excluded from normalization. If you do not specify <i>exclude_list</i> , all <code>NUMBER</code> columns are transformed. <p>The format of <i>exclude_list</i> is:</p> <pre>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2',  ...'coln')</pre>						
<code>round_num</code>	The number of significant digits to use for <i>scale</i> . The default number is 6.						
<code>norm_schema_name</code>	Schema of <i>norm_table_name</i> . If no schema is specified, the current schema is used.						

**Table 45–31 (Cont.) INSERT\_NORM\_LIN\_ZSCORE Procedure Parameters**

Parameter	Description
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.

## Examples

In this example, `INSERT_NORM_LIN_ZSCORE` normalizes the `cust_year_of_birth` column and inserts the transformation in a transformation definition table. The [STACK\\_NORM\\_LIN Procedure](#) creates a transformation list from the contents of the definition table.

The SQL expression that computes the transformation is shown in the view `MINING_DATA_STACK`. The view is for display purposes only; it cannot be used to embed the transformations in a model.

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_gender, cust_year_of_birth
  FROM sh.customers;

DESCRIBE mining_data
Name                               Null?    Type
-----
CUST_ID                            NOT NULL NUMBER
CUST_GENDER                        NOT NULL CHAR(1)
CUST_YEAR_OF_BIRTH                 NOT NULL NUMBER(4)

BEGIN
  dbms_data_mining_transform.CREATE_NORM_LIN(
    norm_table_name => 'norm_tbl');
  dbms_data_mining_transform.INSERT_NORM_LIN_ZSCORE(
    norm_table_name => 'norm_tbl',
    data_table_name => 'mining_data',
    exclude_list   => dbms_data_mining_transform.COLUMN_LIST('cust_id'),
    round_num      => 3);
END;
/

SELECT col, shift, scale FROM norm_tbl;

COL                               SHIFT SCALE
-----
CUST_YEAR_OF_BIRTH               1960    15

DECLARE
  xforms      dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_NORM_LIN (
    norm_table_name => 'norm_tbl',
    xform_list      => xforms);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => xforms,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack');
END;
/

set long 3000
SQL> SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_STACK';
```

TEXT

-----  
SELECT "CUST\_ID", "CUST\_GENDER", ("CUST\_YEAR\_OF\_BIRTH"-1960)/15 "CUST\_YEAR\_OF\_BIRTH"  
FROM mining\_data

## SET\_EXPRESSION Procedure

This procedure appends a row to a VARCHAR2 array that stores a SQL expression. The array can be used for specifying a transformation expression that is too long to be used with the [SET\\_TRANSFORM Procedure](#).

The [GET\\_EXPRESSION Function](#) returns a row in the array.

When you use SET\_EXPRESSION to build a transformation expression, you must build a corresponding reverse transformation expression, create a transformation record, and add the transformation record to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.SET_EXPRESSION (
    expression    IN OUT NOCOPY EXPRESSION_REC,
    chunk         VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–32 SET\_EXPRESSION Procedure Parameters**

Parameter	Description
<code>expression</code>	An expression record (EXPRESSION_REC) that specifies a transformation expression or a reverse transformation expression for an attribute. Each expression record includes a VARCHAR2 array and index fields for specifying upper and lower boundaries within the array.  There are two EXPRESSION_REC fields within a transformation record (TRANSFORM_REC): one for the transformation expression; the other for the reverse transformation expression.  See <a href="#">Table 45–1</a> for a description of the EXPRESSION_REC type.
<code>chunk</code>	A VARCHAR2 chunk (row) to be appended to <code>expression</code> .

### Notes

1. You can pass NULL in the `chunk` argument to SET\_EXPRESSION to clear the previous chunk. The default value of `chunk` is NULL.
2. See "[About Transformation Lists](#)" on page 45-7.
3. See "[Operational Notes](#)" on page 45-6.

### Examples

In this example, two calls to SET\_EXPRESSION construct a transformation expression and two calls construct the reverse transformation.

---

**Note:** This example is for illustration purposes only. It shows how SET\_EXPRESSION appends the text provided in `chunk` to the text that already exists in `expression`. The SET\_EXPRESSION procedure is meant for constructing very long transformation expressions that cannot be specified in a VARCHAR2 argument to SET\_TRANSFORM.

Similarly while transformation lists are intended for embedding in a model, the transformation list `v_xlst` is shown in an external view for illustration purposes.

---

```

CREATE OR REPLACE VIEW mining_data AS
    SELECT cust_id, cust_year_of_birth, cust_postal_code, cust_credit_limit
    FROM sh.customers;

DECLARE
    v_expr dbms_data_mining_transform.EXPRESSION_REC;
    v_rexp dbms_data_mining_transform.EXPRESSION_REC;
    v_xrec dbms_data_mining_transform.TRANSFORM_REC;
    v_xlst dbms_data_mining_transform.TRANSFORM_LIST :=
        dbms_data_mining_transform.TRANSFORM_LIST(NULL);
BEGIN
    dbms_data_mining_transform.SET_EXPRESSION(
        EXPRESSION => v_expr,
        CHUNK      => ('"CUST_YEAR_OF_BIRTH"-1910'));
    dbms_data_mining_transform.SET_EXPRESSION(
        EXPRESSION => v_expr,
        CHUNK      => '/77');
    dbms_data_mining_transform.SET_EXPRESSION(
        EXPRESSION => v_rexp,
        CHUNK      => '"CUST_YEAR_OF_BIRTH"*77');
    dbms_data_mining_transform.SET_EXPRESSION(
        EXPRESSION => v_rexp,
        CHUNK      => '+1910');

    v_xrec := null;
    v_xrec.attribute_name := 'CUST_YEAR_OF_BIRTH';
    v_xrec.expression := v_expr;
    v_xrec.reverse_expression := v_rexp;
    v_xlst.TRIM;
    v_xlst.extend(1);
    v_xlst(1) := v_xrec;

    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => v_xlst,
        data_table_name => 'mining_data',
        xform_view_name => 'v_xlst_view');

    dbms_output.put_line('====');
    FOR i IN 1..v_xlst.count LOOP
        dbms_output.put_line('ATTR: '||v_xlst(i).attribute_name);
        dbms_output.put_line('SUBN: '||v_xlst(i).attribute_subname);
        FOR j IN v_xlst(i).expression.lb..v_xlst(i).expression.ub LOOP
            dbms_output.put_line('EXPR: '||v_xlst(i).expression.lstmt(j));
        END LOOP;
        FOR j IN v_xlst(i).reverse_expression.lb..
            v_xlst(i).reverse_expression.ub LOOP
            dbms_output.put_line('REXP: '||v_xlst(i).reverse_expression.lstmt(j));
        END LOOP;
        dbms_output.put_line('====');
    END LOOP;
END;
/
====
ATTR: CUST_YEAR_OF_BIRTH
SUBN:
EXPR: ("CUST_YEAR_OF_BIRTH"-1910)
EXPR: /77
REXP: "CUST_YEAR_OF_BIRTH"*77
REXP: +1910
====

```

## SET\_TRANSFORM Procedure

This procedure appends the transformation instructions for an attribute to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.SET_TRANSFORM (
    xform_list          IN OUT NOCOPY TRANSFORM_LIST,
    attribute_name      VARCHAR2,
    attribute_subname   VARCHAR2,
    expression          VARCHAR2,
    reverse_expression  VARCHAR2,
    attribute_spec      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–33** SET\_TRANSFORM Procedure Parameters

Parameter	Description
xform_list	A transformation list. See <a href="#">Table 45–1</a> for a description of the TRANSFORM_LIST object type.
attribute_name	Name of the attribute to be transformed
attribute_subname	Name of the nested attribute if <i>attribute_name</i> is a nested column, otherwise NULL.
expression	A SQL expression that specifies the transformation of the attribute.
reverse_expression	A SQL expression that reverses the transformation for readability in model details and in the target of a supervised model (if the attribute is a target)
attribute_spec	You can specify the value NOPREP for <i>attribute_spec</i> to prevent this attribute from being automatically transformed when Automatic Data Preparation is being used.

### Usage Notes

- See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:
  - ["About Transformation Lists"](#) on page 45-7
  - ["Nested Data Transformations"](#) on page 45-10
- As shown in the following example, you can eliminate an attribute by specifying a null transformation expression and reverse expression. You can also use the STACK interface to remove a column ([CREATE\\_COL\\_REM Procedure](#) and [STACK\\_COL\\_REM Procedure](#)).

### Examples

This example uses SET\_TRANSFORM to append transformations to *cust\_stack* for the data set *cust\_info* and displays one row of the transformed data.

SET\_TRANSFORM divides the *country\_id* column by 10, removes the *cust\_year\_of\_birth* column, and multiplies the nested attribute 'custprods.mouse pad' by 10. (See ["DESCRIBE\\_STACK Procedure"](#) on page 45-32 for the definition of *cust\_info*.)

```

describe cust_info
Name                                     Null?   Type
-----
CUST_ID                                 NOT NULL NUMBER
COUNTRY_ID                             NOT NULL NUMBER
CUST_YEAR_OF_BIRTH                      NOT NULL NUMBER(4)
CUSTPRODS                               SYS.DM_NESTED_NUMERICALS

```

```

DECLARE
  cust_stack dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'country_id', NULL, 'country_id/10', 'country_id*10');
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'cust_year_of_birth', NULL, NULL, NULL);
  dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
    'custprods', 'Mouse Pad', 'value*100', 'value/100');
  dbms_data_mining_transform.XFORM_STACK (cust_stack,
    'cust_info', 'xform_cust_view');
END;
/

```

```
select * from xform_cust_view where cust_id = 100004;
```

```

CUST_ID  COUNTRY_ID  CUSTPRODS(ATTRIBUTE_NAME, VALUE)
-----
100004   5279       DM_NESTED_NUMERICALS(DM_NESTED_NUMERICAL
('External 8X CD-ROM', 1),
DM_NESTED_NUMERICAL('Keyboard Wrist Rest', 1))

```

## STACK\_BIN\_CAT Procedure

This procedure adds categorical binning transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_BIN_CAT (
  bin_table_name      IN          VARCHAR2,
  xform_list          IN OUT NOCOPY TRANSFORM_LIST,
  literal_flag        IN          BOOLEAN DEFAULT FALSE,
  bin_schema_name     IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–34** STACK\_BIN\_CAT Procedure Parameters

Parameter	Description
<code>bin_table_name</code>	Name of the transformation definition table for categorical binning. You can use the <a href="#">CREATE_BIN_CAT Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>STACK_BIN_CAT</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for categorical binning or you can write your own SQL.  See <a href="#">Table 45–4, "Columns in a Transformation Definition Table for Categorical Binning"</a>
<code>xform_list</code>	A transformation list. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
<code>literal_flag</code>	Indicates whether the values in the <code>bin</code> column in the transformation definition table are valid SQL literals. When <code>literal_flag</code> is <code>FALSE</code> (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes.  Set <code>literal_flag</code> to <code>TRUE</code> if the bin identifiers are numbers that should have a numeric data type, as is the case for an O-Cluster model.  See <a href="#">"INSERT_BIN_NUM_EQWIDTH Procedure"</a> on page 45-44 for an example.
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how a binning transformation for the categorical column `cust_postal_code` could be added to a stack called `mining_data_stack`.



---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. XFORM\_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE\_MODEL in the xform\_list parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```

CREATE or REPLACE VIEW mining_data AS
  SELECT cust_id, cust_postal_code, cust_credit_limit
     FROM sh.customers
     WHERE cust_id BETWEEN 100050 AND 100100;
BEGIN
  dbms_data_mining_transform.CREATE_BIN_CAT ('bin_cat_tbl');
  dbms_data_mining_transform.INSERT_BIN_CAT_FREQ (
    bin_table_name => 'bin_cat_tbl',
    data_table_name => 'mining_data',
    bin_num        => 3);
END;
/
DECLARE
  MINING_DATA_STACK  dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_CAT (
    bin_table_name => 'bin_cat_tbl',
    xform_list     => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list     => mining_data_stack,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_view');
END;
/
-- Before transformation
column cust_postal_code format a16
SELECT * from mining_data
       WHERE cust_id BETWEEN 100050 AND 100053
       ORDER BY cust_id;

  CUST_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
-----
  100050 76486                1500
  100051 73216                9000
  100052 69499                5000
  100053 45704                7000

-- After transformation
SELECT * FROM mining_data_stack_view
       WHERE cust_id BETWEEN 100050 AND 100053
       ORDER BY cust_id;

  CUST_ID CUST_POSTAL_CODE CUST_CREDIT_LIMIT
-----
  100050 4                    1500
  100051 1                    9000
  100052 4                    5000
  100053 4                    7000

```

## STACK\_BIN\_NUM Procedure

This procedure adds numerical binning transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_BIN_NUM (
  bin_table_name      IN          VARCHAR2,
  xform_list          IN OUT NOCOPY TRANSFORM_LIST,
  literal_flag        IN          BOOLEAN DEFAULT FALSE,
  bin_schema_name     IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–35** *STACK\_BIN\_NUM Procedure Parameters*

Parameter	Description
<code>bin_table_name</code>	Name of the transformation definition table for numerical binning. You can use the <a href="#">CREATE_BIN_NUM Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>STACK_BIN_NUM</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for numerical binning or you can write your own SQL.  See <a href="#">Table 45–6, "Columns in a Transformation Definition Table for Numerical Binning"</a> .
<code>xform_list</code>	A transformation list. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
<code>literal_flag</code>	Indicates whether the values in the <code>bin</code> column in the transformation definition table are valid SQL literals. When <code>literal_flag</code> is <code>FALSE</code> (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes.  Set <code>literal_flag</code> to <code>TRUE</code> if the bin identifiers are numbers that should have a numeric data type, as is the case for an O-Cluster model.  See <a href="#">"INSERT_BIN_NUM_EQWIDTH Procedure"</a> on page 45-44 for an example.
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how a binning transformation for the numerical column `cust_credit_limit` could be added to a stack called `mining_data_stack`.

---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. XFORM\_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE\_MODEL in the xform\_list parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```

CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_postal_code, cust_credit_limit
     FROM sh.customers
     WHERE cust_id BETWEEN 100050 and 100100;
BEGIN
  dbms_data_mining_transform.create_bin_num ('bin_num_tbl');
  dbms_data_mining_transform.insert_bin_num_qtile (
    bin_table_name => 'bin_num_tbl',
    data_table_name => 'mining_data',
    bin_num        => 5,
    exclude_list  => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/
DECLARE
  MINING_DATA_STACK dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_BIN_CAT (
    bin_table_name => 'bin_num_tbl',
    xform_list     => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list     => mining_data_stack,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT cust_id, cust_postal_code, ROUND(cust_credit_limit) FROM mining_data
     WHERE cust_id BETWEEN 100050 AND 100055
     ORDER BY cust_id;
CUST_ID  CUST_POSTAL_CODE  ROUND(CUST_CREDIT_LIMIT)
-----  -
100050   76486              1500
100051   73216              9000
100052   69499              5000
100053   45704              7000
100055   74673             11000
100055   74673             11000

-- After transformation
SELECT cust_id, cust_postal_code, ROUND(cust_credit_limit)
     FROM mining_data_stack_view
     WHERE cust_id BETWEEN 100050 AND 100055
     ORDER BY cust_id;
CUST_ID  CUST_POSTAL_CODE  ROUND(CUST_CREDIT_LIMIT)
-----  -
100050   76486
100051   73216              2
100052   69499              1
100053   45704
100054   88021              3

```

100055 74673

3

## STACK\_CLIP Procedure

This procedure adds clipping transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_CLIP (
    clip_table_name      IN          VARCHAR2,
    xform_list           IN OUT NOCOPY TRANSFORM_LIST,
    clip_schema_name     IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–36** STACK\_CLIP Procedure Parameters

Parameter	Description
<code>clip_table_name</code>	Name of the transformation definition table for clipping. You can use the <a href="#">CREATE_CLIP Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>STACK_CLIP</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for clipping or you can write your own SQL.  See <a href="#">Table 45–8, "Columns in a Transformation Definition Table for Clipping or Winsorizing"</a>
<code>xform_list</code>	A transformation list. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
<code>clip_schema_name</code>	Schema of <code>clip_table_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how a clipping transformation for the numerical column `cust_credit_limit` could be added to a stack called `mining_data_stack`.

---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, cust_postal_code, cust_credit_limit
FROM sh.customers
WHERE cust_id BETWEEN 100050 AND 100100;
```

```

BEGIN
  dbms_data_mining_transform.create_clip ('clip_tbl');
  dbms_data_mining_transform.insert_clip_winsor_tail (
    clip_table_name => 'clip_tbl',
    data_table_name => 'mining_data',
    tail_frac       => 0.25,
    exclude_list    => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
END;
/
DECLARE
  MINING_DATA_STACK dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_CLIP (
    clip_table_name => 'clip_tbl',
    xform_list      => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => mining_data_stack,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT cust_id, cust_postal_code, round(cust_credit_limit)
   FROM mining_data
      WHERE cust_id BETWEEN 100050 AND 100054
      ORDER BY cust_id;

CUST_ID  CUST_POSTAL_CODE  ROUND(CUST_CREDIT_LIMIT)
-----  -
100050   76486              1500
100051   73216              9000
100052   69499              5000
100053   45704              7000
100054   88021              11000

-- After transformation
SELECT cust_id, cust_postal_code, round(cust_credit_limit)
   FROM mining_data_stack_view
      WHERE cust_id BETWEEN 100050 AND 100054
      ORDER BY cust_id;

CUST_ID  CUST_POSTAL_CODE  ROUND(CUST_CREDIT_LIMIT)
-----  -
100050   76486              5000
100051   73216              9000
100052   69499              5000
100053   45704              7000
100054   88021              11000

```

## STACK\_COL\_REM Procedure

This procedure adds column removal transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_COL_REM (
    rem_table_name      IN          VARCHAR2,
    xform_list          IN OUT NOCOPY TRANSFORM_LIST,
    rem_schema_name     IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–37** STACK\_COL\_REM Procedure Parameters

Parameter	Description
rem_table_name	Name of the transformation definition table for column removal. You can use the <a href="#">CREATE_COL_REM Procedure</a> to create the definition table. See <a href="#">Table 45–10</a> , "Columns in a Transformation Definition Table for Column Removal".  The table must be populated with column names before you call STACK_COL_REM. The <a href="#">INSERT_BIN_SUPER Procedure</a> and the <a href="#">INSERT_AUTOBIN_NUM_EQWIDTH Procedure</a> can optionally be used to populate the table. You can also use SQL INSERT statements.
xform_list	A transformation list. See <a href="#">Table 45–1</a> for a description of the TRANSFORM_LIST object type.
rem_schema_name	Schema of <i>rem_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how the column `cust_credit_limit` could be removed in a transformation list called `mining_data_stack`.

---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. XFORM\_STACK simply generates an external view of the transformed data. The actual purpose of the STACK procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to CREATE\_MODEL in the `xform_list` parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, country_id, cust_postal_code, cust_credit_limit
  FROM sh.customers;
```

```

BEGIN
    dbms_data_mining_transform.create_col_rem ('rem_tbl');
END;
/

INSERT into rem_tbl VALUES (upper('cust_postal_code'), null);

DECLARE
    MINING_DATA_STACK  dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.stack_col_rem (
        rem_table_name => 'rem_tbl',
        xform_list      => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => mining_data_stack,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack_view');
END;
/

SELECT * FROM mining_data
    WHERE cust_id BETWEEN 100050 AND 100051
    ORDER BY cust_id;

CUST_ID  COUNTRY_ID  CUST_POSTAL_CODE  CUST_CREDIT_LIMIT
-----  -
100050      52773    76486                1500
100051      52790    73216                9000

SELECT * FROM mining_data_stack_view
    WHERE cust_id BETWEEN 100050 AND 100051
    ORDER BY cust_id;

CUST_ID  COUNTRY_ID  CUST_CREDIT_LIMIT
-----  -
100050      52773                1500
100051      52790                9000

```



## STACK\_MISS\_CAT Procedure

This procedure adds categorical missing value transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_MISS_CAT (
    miss_table_name      IN      VARCHAR2,
    xform_list           IN OUT  NOCOPY TRANSFORM_LIST,
    miss_schema_name     IN      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–38** STACK\_MISS\_CAT Procedure Parameters

Parameter	Description
miss_table_name	Name of the transformation definition table for categorical missing value treatment. You can use the <a href="#">CREATE_MISS_CAT Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>STACK_MISS_CAT</code> . To populate the table, you can use the <a href="#">INSERT_MISS_CAT_MODE Procedure</a> or you can write your own SQL.  See <a href="#">Table 45–12</a> , "Columns in a Transformation Definition Table for Categorical Missing Value Treatment".
xform_list	A transformation list. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
miss_schema_name	Schema of <i>miss_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

See "Operational Notes" on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how the missing values in the column `cust_marital_status` could be replaced with the mode in a transformation list called `mining_data_stack`.

---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, country_id, cust_marital_status
FROM sh.customers
where cust_id BETWEEN 1 AND 10;
```

```

BEGIN
  dbms_data_mining_transform.create_miss_cat ('miss_cat_tbl');
  dbms_data_mining_transform.insert_miss_cat_mode ('miss_cat_tbl', 'mining_data');
END;
/

```

```

DECLARE
  MINING_DATA_STACK dbms_data_mining_transform.TRANSFORM_LIST;

```

```

BEGIN
  dbms_data_mining_transform.stack_miss_cat (
    miss_table_name => 'miss_cat_tbl',
    xform_list      => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => mining_data_stack,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_view');

```

```

END;
/
SELECT * FROM mining_data
ORDER BY cust_id;

```

CUST_ID	COUNTRY_ID	CUST_MARITAL_STATUS
1	52789	
2	52778	
3	52770	
4	52770	
5	52789	
6	52769	single
7	52790	single
8	52790	married
9	52770	divorced
10	52790	widow

```

SELECT * FROM mining_data_stack_view
ORDER By cust_id;

```

CUST_ID	COUNTRY_ID	CUST_MARITAL_STATUS
1	52789	single
2	52778	single
3	52770	single
4	52770	single
5	52789	single
6	52769	single
7	52790	single
8	52790	married
9	52770	divorced
10	52790	widow

## STACK\_MISS\_NUM Procedure

This procedure adds numeric missing value transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_MISS_NUM (
    miss_table_name    IN      VARCHAR2,
    xform_list         IN OUT  NOCOPY TRANSFORM_LIST,
    miss_schema_name   IN      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–39** STACK\_MISS\_NUM Procedure Parameters

Parameter	Description
miss_table_name	Name of the transformation definition table for numerical missing value treatment. You can use the <a href="#">CREATE_MISS_NUM Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>STACK_MISS_NUM</code> . To populate the table, you can use the <a href="#">INSERT_MISS_NUM_MEAN Procedure</a> or you can write your own SQL.  See <a href="#">Table 45–14</a> , "Columns in a Transformation Definition Table for Numerical Missing Value Treatment".
xform_list	A transformation list. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
miss_schema_name	Schema of <i>miss_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how the missing values in the column `cust_credit_limit` could be replaced with the mean in a transformation list called `mining_data_stack`.

---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```
describe mining_data
Name                                                    Null?   Type
-----
CUST_ID                                                NOT NULL NUMBER
```

```

CUST_CREDIT_LIMIT                                NUMBER

BEGIN
  dbms_data_mining_transform.create_miss_num ('miss_num_tbl');
  dbms_data_mining_transform.insert_miss_num_mean ('miss_num_tbl','mining_data');
END;
/
SELECT * FROM miss_num_tbl;

COL          ATT          VAL
-----
CUST_ID          5.5
CUST_CREDIT_LIMIT 185.71

DECLARE
  MINING_DATA_STACK  dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
  dbms_data_mining_transform.STACK_MISS_NUM (
    miss_table_name => 'miss_num_tbl',
    xform_list      => mining_data_stack);
  dbms_data_mining_transform.XFORM_STACK (
    xform_list      => mining_data_stack,
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_stack_view');
END;
/
-- Before transformation
SELECT * FROM mining_data
  ORDER BY cust_id;
CUST_ID CUST_CREDIT_LIMIT
-----
      1             100
      2
      3             200
      4
      5             150
      6             400
      7             150
      8
      9             100
     10             200

-- After transformation
SELECT * FROM mining_data_stack_view
  ORDER BY cust_id;
CUST_ID CUST_CREDIT_LIMIT
-----
      1             100
      2            185.71
      3             200
      4            185.71
      5             150
      6             400
      7             150
      8            185.71
      9             100
     10             200

```

## STACK\_NORM\_LIN Procedure

This procedure adds linear normalization transformations to a transformation list.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.STACK_NORM_LIN (
    norm_table_name      IN          VARCHAR2,
    xform_list           IN OUT      NOCOPY TRANSFORM_LIST,
    norm_schema_name     IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–40** STACK\_NORM\_LIN Procedure Parameters

Parameter	Description
norm_table_name	Name of the transformation definition table for linear normalization. You can use the <a href="#">CREATE_NORM_LIN Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>STACK_NORM_LIN</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for normalization or you can write your own SQL.  See <a href="#">Table 45–16, "Columns in a Transformation Definition Table for Linear Normalization"</a> .
xform_list	A transformation list. See <a href="#">Table 45–1</a> for a description of the <code>TRANSFORM_LIST</code> object type.
norm_schema_name	Schema of <i>norm_table_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

### Examples

This example shows how the column `cust_credit_limit` could be normalized in a transformation list called `mining_data_stack`.

---

**Note:** This example invokes the [XFORM\\_STACK Procedure](#) to show how the data is transformed by the stack. `XFORM_STACK` simply generates an external view of the transformed data. The actual purpose of the `STACK` procedures is to assemble a list of transformations for embedding in a model. The transformations are passed to `CREATE_MODEL` in the `xform_list` parameter. See [INSERT\\_BIN\\_NUM\\_EQWIDTH Procedure](#) for an example.

---

```
CREATE OR REPLACE VIEW mining_data AS
SELECT cust_id, country_id, cust_postal_code, cust_credit_limit
FROM sh.customers;
BEGIN
```

```

dbms_data_mining_transform.create_norm_lin ('norm_lin_tbl');
dbms_data_mining_transform.insert_norm_lin_minmax (
    norm_table_name => 'norm_lin_tbl',
    data_table_name => 'mining_data',
    exclude_list   => dbms_data_mining_transform.COLUMN_LIST('cust_id',
                                                                'country_id'));

END;
/
SELECT * FROM norm_lin_tbl;
COL          ATT      SHIFT  SCALE
-----
CUST_CREDIT_LIMIT      1500  13500

DECLARE
    MINING_DATA_STACK  dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.stack_norm_lin (
        norm_table_name => 'norm_lin_tbl',
        xform_list      => mining_data_stack);
    dbms_data_mining_transform.XFORM_STACK (
        xform_list      => mining_data_stack,
        data_table_name => 'mining_data',
        xform_view_name => 'mining_data_stack_view');
END;
/
SELECT * FROM mining_data
    WHERE cust_id between 1 and 10
    ORDER BY cust_id;
CUST_ID COUNTRY_ID CUST_POSTAL_CODE    CUST_CREDIT_LIMIT
-----
      1      52789 30828                9000
      2      52778 86319               10000
      3      52770 88666                1500
      4      52770 87551                1500
      5      52789 59200                1500
      6      52769 77287                1500
      7      52790 38763                1500
      8      52790 58488                3000
      9      52770 63033                3000
     10      52790 52602                3000

SELECT * FROM mining_data_stack_view
    WHERE cust_id between 1 and 10
    ORDER BY cust_id;
CUST_ID COUNTRY_ID CUST_POSTAL_CODE    CUST_CREDIT_LIMIT
-----
      1      52789 30828                .55556
      2      52778 86319               .62963
      3      52770 88666                0
      4      52770 87551                0
      5      52789 59200                0
      6      52769 77287                0
      7      52790 38763                0
      8      52790 58488               .11111
      9      52770 63033               .11111
     10      52790 52602               .11111

```

## XFORM\_BIN\_CAT Procedure

This procedure creates a view that implements the categorical binning transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_CAT (
    bin_table_name      IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    xform_view_name     IN VARCHAR2,
    literal_flag        IN BOOLEAN DEFAULT FALSE,
    bin_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL,
    xform_schema_name   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–41 XFORM\_BIN\_CAT Procedure Parameters**

Parameter	Description
<code>bin_table_name</code>	Name of the transformation definition table for categorical binning. You can use the <a href="#">CREATE_BIN_CAT Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>XFORM_BIN_CAT</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for categorical binning or you can write your own SQL.  See <a href="#">Table 45–4, "Columns in a Transformation Definition Table for Categorical Binning"</a> .
<code>data_table_name</code>	Name of the table containing the data to be transformed.
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>bin_table_name</code> .
<code>literal_flag</code>	Indicates whether the values in the <code>bin</code> column in the transformation definition table are valid SQL literals. When <code>literal_flag</code> is <code>FALSE</code> (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes.  Set <code>literal_flag</code> to <code>TRUE</code> if the bin identifiers are numbers that should have a numeric data type, as is the case for an O-Cluster model.  See <a href="#">"INSERT_BIN_NUM_EQWIDTH Procedure"</a> on page 45-44 for an example.
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6.

## Examples

This example creates a view that bins the `cust_postal_code` column. The data source consists of three columns from `sh.customer`.

```
describe mining_data
```

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_POSTAL_CODE	NOT NULL	VARCHAR2(10)
CUST_CREDIT_LIMIT		NUMBER

```
SELECT * FROM mining_data WHERE cust_id between 104066 and 104069;
```

CUST_ID	CUST_POSTAL_CODE	CUST_CREDIT_LIMIT
104066	69776	7000
104067	52602	9000
104068	55787	11000
104069	55977	5000

```
BEGIN
```

```
  dbms_data_mining_transform.create_bin_cat(
    bin_table_name => 'bin_cat_tbl');
  dbms_data_mining_transform.insert_bin_cat_freq(
    bin_table_name => 'bin_cat_tbl',
    data_table_name => 'mining_data',
    bin_num => 10);
  dbms_data_mining_transform.xform_bin_cat(
    bin_table_name => 'bin_cat_tbl',
    data_table_name => 'mining_data',
    xform_view_name => 'bin_cat_view');
```

```
END;
```

```
/
```

```
SELECT * FROM bin_cat_view WHERE cust_id between 104066 and 104069;
```

CUST_ID	CUST_POSTAL_CODE	CUST_CREDIT_LIMIT
104066	6	7000
104067	11	9000
104068	3	11000
104069	11	5000

```
SELECT text FROM user_views WHERE view_name IN 'BIN_CAT_VIEW';
```

```
TEXT
```

```
-----
SELECT "CUST_ID",DECODE("CUST_POSTAL_CODE",'38082','1','45704','9','48346','5','55787','3','63736','2','67843','7','69776','6','72860','10','78558','4','80841','8',NULL,NULL,'11') "CUST_POSTAL_CODE","CUST_CREDIT_LIMIT" FROM mining_data
```



## XFORM\_BIN\_NUM Procedure

This procedure creates a view that implements the numerical binning transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_NUM (
    bin_table_name      IN VARCHAR2,
    data_table_name     IN VARCHAR2,
    xform_view_name    IN VARCHAR2,
    literal_flag        IN BOOLEAN DEFAULT FALSE,
    bin_schema_name     IN VARCHAR2 DEFAULT NULL,
    data_schema_name    IN VARCHAR2 DEFAULT NULL,
    xform_schema_name  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–42 XFORM\_BIN\_NUM Procedure Parameters**

Parameter	Description
<code>bin_table_name</code>	Name of the transformation definition table for numerical binning. You can use the <a href="#">CREATE_BIN_NUM Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>XFORM_BIN_NUM</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for numerical binning or you can write your own SQL.  See "Columns in a Transformation Definition Table for Numerical Binning" on page 45-21.
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>bin_table_name</code> .
<code>literal_flag</code>	Indicates whether the values in the <code>bin</code> column in the transformation definition table are valid SQL literals. When <code>literal_flag</code> is <code>FALSE</code> (the default), the bin identifiers will be transformed to SQL literals by surrounding them with single quotes.  Set <code>literal_flag</code> to <code>TRUE</code> if the bin identifiers are numbers that should have a numeric data type, as is the case for an O-Cluster model.  See " <a href="#">INSERT_BIN_NUM_EQWIDTH Procedure</a> " on page 45-44 for an example.
<code>bin_schema_name</code>	Schema of <code>bin_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See "[Operational Notes](#)" on page 45-6.

## Examples

This example creates a view that bins the `cust_credit_limit` column. The data source consists of three columns from `sh.customer`.

```
describe mining_data
Name                                         Null?    Type
-----
CUST_ID                                     NOT NULL NUMBER
CUST_POSTAL_CODE                           NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                           NUMBER

column cust_credit_limit off
SELECT * FROM mining_data WHERE cust_id between 104066 and 104069;

      CUST_ID CUST_POSTAL_CODE  CUST_CREDIT_LIMIT
-----
104066 69776                    7000
104067 52602                    9000
104068 55787                   11000
104069 55977                    5000

BEGIN
  dbms_data_mining_transform.create_bin_num(
    bin_table_name => 'bin_num_tbl');
  dbms_data_mining_transform.insert_autobin_num_eqwidth(
    bin_table_name => 'bin_num_tbl',
    data_table_name => 'mining_data',
    bin_num => 5,
    max_bin_num => 10,
    exclude_list => dbms_data_mining_transform.COLUMN_LIST('cust_id'));
  dbms_data_mining_transform.xform_bin_num(
    bin_table_name => 'bin_num_tbl',
    data_table_name => 'mining_data',
    xform_view_name => 'mining_data_view');
END;
/
describe mining_data_view
Name                                         Null?    Type
-----
CUST_ID                                     NOT NULL NUMBER
CUST_POSTAL_CODE                           NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                           VARCHAR2(2)

col cust_credit_limit on
col cust_credit_limit format a25
SELECT * FROM mining_data_view WHERE cust_id between 104066 and 104069;

      CUST_ID CUST_POSTAL_CODE  CUST_CREDIT_LIMIT
-----
104066 69776                    5
104067 52602                    6
104068 55787                    8
104069 55977                    3

set long 2000
SELECT text FROM user_views WHERE view_name IN 'MINING_DATA_VIEW';

TEXT
-----
SELECT "CUST_ID", "CUST_POSTAL_CODE", CASE WHEN "CUST_CREDIT_LIMIT"<1500 THEN NULL
```

```
WHEN "CUST_CREDIT_LIMIT"<=2850 THEN '1' WHEN "CUST_CREDIT_LIMIT"<=4200 THEN '2'  
WHEN "CUST_CREDIT_LIMIT"<=5550 THEN '3' WHEN "CUST_CREDIT_LIMIT"<=6900 THEN '4'  
WHEN "CUST_CREDIT_LIMIT"<=8250 THEN '5' WHEN "CUST_CREDIT_LIMIT"<=9600 THEN '6'  
WHEN "CUST_CREDIT_LIMIT"<=10950 THEN '7' WHEN "CUST_CREDIT_LIMIT"<=12300 THEN '  
8' WHEN "CUST_CREDIT_LIMIT"<=13650 THEN '9' WHEN "CUST_CREDIT_LIMIT"<=15000 THEN  
'10' END "CUST_CREDIT_LIMIT" FROM mining_data
```

## XFORM\_CLIP Procedure

This procedure creates a view that implements the clipping transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_CLIP (
  clip_table_name      IN VARCHAR2,
  data_table_name      IN VARCHAR2,
  xform_view_name      IN VARCHAR2,
  clip_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name     IN VARCHAR2,DEFAULT NULL,
  xform_schema_name    IN VARCHAR2,DEFAULT NULL);
```

### Parameters

**Table 45–43 XFORM\_CLIP Procedure Parameters**

Parameter	Description
<code>clip_table_name</code>	Name of the transformation definition table for clipping. You can use the <a href="#">CREATE_CLIP Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>XFORM_CLIP</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for clipping you can write your own SQL.  See <a href="#">Table 45–8, "Columns in a Transformation Definition Table for Clipping or Winsorizing"</a> .
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>clip_table_name</code> .
<code>clip_schema_name</code>	Schema of <code>clip_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Examples

This example creates a view that clips the `cust_credit_limit` column. The data source consists of three columns from `sh.customer`.

```
describe mining_data
Name                               Null?    Type
-----
CUST_ID                             NOT NULL NUMBER
CUST_POSTAL_CODE                     NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                     NUMBER

BEGIN
  dbms_data_mining_transform.create_clip(
    clip_table_name => 'clip_tbl');
  dbms_data_mining_transform.insert_clip_trim_tail(
```

```

        clip_table_name => 'clip_tbl',
        data_table_name => 'mining_data',
        tail_frac       => 0.05,
        exclude_list    => dbms_data_mining_transform.COLUMN_LIST('cust_id');
    dbms_data_mining_transform.xform_clip(
        clip_table_name => 'clip_tbl',
        data_table_name => 'mining_data',
        xform_view_name => 'clip_view');
END;
/
describe clip_view
Name                               Null?    Type
-----
CUST_ID                             NOT NULL NUMBER
CUST_POSTAL_CODE                     NOT NULL VARCHAR2(10)
CUST_CREDIT_LIMIT                     NUMBER

SELECT MIN(cust_credit_limit), MAX(cust_credit_limit) FROM mining_data;

MIN(CUST_CREDIT_LIMIT) MAX(CUST_CREDIT_LIMIT)
-----
                1500                15000

SELECT MIN(cust_credit_limit), MAX(cust_credit_limit) FROM clip_view;

MIN(CUST_CREDIT_LIMIT) MAX(CUST_CREDIT_LIMIT)
-----
                1500                11000

set long 2000
SELECT text FROM user_views WHERE view_name IN 'CLIP_VIEW';

TEXT
-----
SELECT "CUST_ID","CUST_POSTAL_CODE",CASE WHEN "CUST_CREDIT_LIMIT" < 1500 THEN NU
LL WHEN "CUST_CREDIT_LIMIT" > 11000 THEN NULL ELSE "CUST_CREDIT_LIMIT" END "CUST
_CREDIT_LIMIT" FROM mining_data

```

## XFORM\_COL\_REM Procedure

This procedure creates a view that implements the column removal transformations specified in a definition table. Only the columns that are specified in the definition table are removed; the remaining columns from the data table are present in the view.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_COL_REM (
  rem_table_name      IN      VARCHAR2,
  data_table_name     IN      VARCHAR2,
  xform_view_name     IN      VARCHAR2,
  rem_schema_name     IN      VARCHAR2 DEFAULT NULL,
  data_schema_name    IN      VARCHAR2 DEFAULT NULL,
  xform_schema_name   IN      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–44 XFORM\_COL\_REM Procedure Parameters**

Parameter	Description
rem_table_name	Name of the transformation definition table for column removal. You can use the <a href="#">CREATE_COL_REM Procedure</a> to create the definition table. See <a href="#">Table 45–10, "Columns in a Transformation Definition Table for Column Removal"</a> .  The table must be populated with column names before you call XFORM_COL_REM. The <a href="#">INSERT_BIN_SUPER Procedure</a> and the <a href="#">INSERT_AUTOBIN_NUM_EQWIDTH Procedure</a> can optionally be used to populate the table. You can also use SQL INSERT statements.
data_table_name	Name of the table containing the data to be transformed
xform_view_name	Name of the view to be created. The view presents the columns in <i>data_table_name</i> that are not specified in <i>rem_table_name</i> .
rem_schema_name	Schema of <i>rem_table_name</i> . If no schema is specified, the current schema is used.
data_schema_name	Schema of <i>data_table_name</i> . If no schema is specified, the current schema is used.
xform_schema_name	Schema of <i>xform_view_name</i> . If no schema is specified, the current schema is used.

### Usage Notes

See "[Operational Notes](#)" on page 45-6.

### Examples

This example creates a view that includes all but one column from the table *customers* in the current schema.

```
describe customers
Name                                                    Null?    Type
-----
CUST_ID                                                NOT NULL NUMBER
CUST_MARITAL_STATUS                                   VARCHAR2 (20)
OCCUPATION                                             VARCHAR2 (21)
```

```

AGE                                NUMBER
YRS_RESIDENCE                       NUMBER

BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_COL_REM ('colrem_xtbl');
END;
/
INSERT INTO colrem_xtbl VALUES('CUST_MARITAL_STATUS', null);

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_COL_REM (
    rem_table_name      => 'colrem_xtbl',
    data_table_name     => 'customers',
    xform_view_name     => 'colrem_view');
END;
/
describe colrem_view

```

Name	Null?	Type
-----	-----	-----
CUST_ID	NOT NULL	NUMBER
OCCUPATION		VARCHAR2 (21)
AGE		NUMBER
YRS_RESIDENCE		NUMBER

## XFORM\_EXPR\_NUM Procedure

This procedure creates a view that implements the specified numeric transformations. Only the columns that you specify are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_NUM (
  expr_pattern      IN      VARCHAR2,
  data_table_name  IN      VARCHAR2,
  xform_view_name  IN      VARCHAR2,
  exclude_list     IN      COLUMN_LIST DEFAULT NULL,
  include_list     IN      COLUMN_LIST DEFAULT NULL,
  col_pattern      IN      VARCHAR2 DEFAULT ':col',
  data_schema_name IN      VARCHAR2 DEFAULT NULL,
  xform_schema_name IN     VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–45 XFORM\_EXPR\_NUM Procedure Parameters**

Parameter	Description
<code>expr_pattern</code>	A numeric transformation expression
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>expr_pattern</code> and <code>col_pattern</code> .
<code>exclude_list</code>	List of numeric columns to exclude. If NULL, no numeric columns are excluded.  The format of <code>exclude_list</code> is:  <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')</code>
<code>include_list</code>	List of numeric columns to include. If NULL, all numeric columns are included.  The format of <code>include_list</code> is:  <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')</code>
<code>col_pattern</code>	The value within <code>expr_pattern</code> that will be replaced with a column name. The value of <code>col_pattern</code> is case-sensitive.  The default value of <code>col_pattern</code> is <code>:col</code>
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

1. The XFORM\_EXPR\_NUM procedure constructs numeric transformation expressions from the specified expression pattern (`expr_pattern`) by replacing every occurrence of the specified column pattern (`col_pattern`) with an actual column name.



XFORM\_EXPR\_NUM uses the SQL REPLACE function to construct the transformation expressions.

```
REPLACE (expr_pattern,col_pattern,'"column_name"' || '"column_name"')
```

If there is a column match, then the replacement is made in the transformation expression; if there is not a match, then the column is used without transformation.

**See:** *Oracle Database SQL Language Reference* for information about the REPLACE function

2. Because of the include and exclude list parameters, the XFORM\_EXPR\_NUM and XFORM\_EXPR\_STR procedures allow you to easily specify individual columns for transformation within large data sets. The other XFORM\_\* procedures support an exclude list only. In these procedures, you must enumerate every column that you do not want to transform.
3. See "[Operational Notes](#)" on page 45-6

## Examples

This example creates a view that transforms the data type of numeric columns.

```
describe customers
Name                                         Null?   Type
-----
CUST_ID                                     NOT NULL NUMBER
CUST_MARITAL_STATUS                        VARCHAR2(20)
OCCUPATION                                  VARCHAR2(21)
AGE                                          NUMBER
YRS_RESIDENCE                               NUMBER

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_NUM(
    expr_pattern      => 'to_char(:col)',
    data_table_name   => 'customers',
    xform_view_name   => 'cust_nonum_view',
    exclude_list      => dbms_data_mining_transform.COLUMN_LIST('cust_id'),
    include_list      => null,
    col_pattern       => ':col');
END;
/
describe cust_nonum_view
Name                                         Null?   Type
-----
CUST_ID                                     NOT NULL NUMBER
CUST_MARITAL_STATUS                        VARCHAR2(20)
OCCUPATION                                  VARCHAR2(21)
AGE                                          VARCHAR2(40)
YRS_RESIDENCE                               VARCHAR2(40)
```

## XFORM\_EXPR\_STR Procedure

This procedure creates a view that implements the specified categorical transformations. Only the columns that you specify are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_STR (
  expr_pattern      IN      VARCHAR2,
  data_table_name  IN      VARCHAR2,
  xform_view_name  IN      VARCHAR2,
  exclude_list     IN      COLUMN_LIST DEFAULT NULL,
  include_list     IN      COLUMN_LIST DEFAULT NULL,
  col_pattern      IN      VARCHAR2 DEFAULT ':col',
  data_schema_name IN      VARCHAR2 DEFAULT NULL,
  xform_schema_name IN     VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45-46 XFORM\_EXPR\_STR Procedure Parameters**

Parameter	Description
<code>expr_pattern</code>	A character transformation expression
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>expr_pattern</code> and <code>col_pattern</code> .
<code>exclude_list</code>	List of character columns to exclude. If NULL, no character columns are excluded.  The format of <code>exclude_list</code> is: <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')</code>
<code>include_list</code>	List of character columns to include. If NULL, all character columns are included.  The format of <code>include_list</code> is: <code>dbms_data_mining_transform.COLUMN_LIST('col1', 'col2', ... 'coln')</code>
<code>col_pattern</code>	The value within <code>expr_pattern</code> that will be replaced with a column name. The value of <code>col_pattern</code> is case-sensitive.  The default value of <code>col_pattern</code> is <code>:col</code>
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

1. The XFORM\_EXPR\_STR procedure constructs character transformation expressions from the specified expression pattern (`expr_pattern`) by replacing every

occurrence of the specified column pattern (*col\_pattern*) with an actual column name.

XFORM\_EXPR\_STR uses the SQL REPLACE function to construct the transformation expressions.

```
REPLACE (expr_pattern,col_pattern,'"column_name"') || '"column_name"'
```

If there is a column match, then the replacement is made in the transformation expression; if there is not a match, then the column is used without transformation.

**See:** *Oracle Database SQL Language Reference* for information about the REPLACE function

- Because of the include and exclude list parameters, the XFORM\_EXPR\_STR and XFORM\_EXPR\_NUM procedures allow you to easily specify individual columns for transformation within large data sets. The other XFORM\_\* procedures support an exclude list only. In these procedures, you must enumerate every column that you do not want to transform.
- See "[Operational Notes](#)" on page 45-6

## Examples

This example creates a view that transforms character columns to upper case.

```
describe customers
```

Name	Null?	Type
-----		
CUST_ID	NOT NULL	NUMBER
CUST_MARITAL_STATUS		VARCHAR2 (20)
OCCUPATION		VARCHAR2 (21)
AGE		NUMBER
YRS_RESIDENCE		NUMBER

```
SELECT cust_id, cust_marital_status, occupation FROM customers
WHERE cust_id > 102995
ORDER BY cust_id desc;
```

CUST_ID	CUST_MARITAL_STATUS	OCCUPATION
-----		
103000	Divorc.	Cleric.
102999	Married	Cleric.
102998	Married	Exec.
102997	Married	Exec.
102996	NeverM	Other

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_EXPR_STR(
    expr_pattern      => 'upper(:col)',
    data_table_name   => 'customers',
    xform_view_name   => 'cust_upcase_view');
END;
```

```
/
describe cust_upcase_view
```

Name	Null?	Type
-----		
CUST_ID	NOT NULL	NUMBER
CUST_MARITAL_STATUS		VARCHAR2 (20)

```
OCCUPATION          VARCHAR2(21)
AGE                 NUMBER
YRS_RESIDENCE       NUMBER
```

```
SELECT cust_id, cust_marital_status, occupation FROM cust_upcase_view
WHERE cust_id > 102995
ORDER BY cust_id desc;
```

```
CUST_ID CUST_MARITAL_STATUS OCCUPATION
-----
103000 DIVORC.           CLERIC.
102999 MARRIED           CLERIC.
102998 MARRIED           EXEC.
102997 MARRIED           EXEC.
102996 NEVERM           OTHER
```

## XFORM\_MISS\_CAT Procedure

This procedure creates a view that implements the categorical missing value treatment transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT (
  miss_table_name      IN VARCHAR2,
  data_table_name      IN VARCHAR2,
  xform_view_name      IN VARCHAR2,
  miss_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name     IN VARCHAR2 DEFAULT NULL,
  xform_schema_name    IN VARCHAR2 DEFAULT NULL;
```

### Parameters

**Table 45–47 XFORM\_MISS\_CAT Procedure Parameters**

Parameter	Description
<code>miss_table_name</code>	Name of the transformation definition table for categorical missing value treatment. You can use the <a href="#">CREATE_MISS_CAT Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>XFORM_MISS_CAT</code> . To populate the table, you can use the <a href="#">INSERT_MISS_CAT_MODE Procedure</a> or you can write your own SQL.  See <a href="#">Table 45–12, "Columns in a Transformation Definition Table for Categorical Missing Value Treatment"</a> .
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>miss_table_name</code> .
<code>miss_schema_name</code>	Schema of <code>miss_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6.

### Examples

This example creates a view that replaces missing categorical values with the mode.

```
SELECT * FROM geog;
```

```
REG_ID REGION
```

```
-----
1 NE
2 SW
3 SE
```

```

4 SW
5
6 NE
7 NW
8 NW
9
10
11 SE
12 SE
13 NW
14 SE
15 SE

SELECT STATS_MODE(region) FROM geog;

STATS_MODE(REGION)
-----
SE

BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_CAT('misscat_xtbl');
  DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_CAT_MODE (
    miss_table_name      => 'misscat_xtbl',
    data_table_name      => 'geog' );
END;
/

SELECT col, val FROM misscat_xtbl;

COL          VAL
-----
REGION      SE

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_CAT (
    miss_table_name      => 'misscat_xtbl',
    data_table_name      => 'geog',
    xform_view_name      => 'geogxf_view');
END;
/

SELECT * FROM geogxf_view;

REG_ID REGION
-----
1 NE
2 SW
3 SE
4 SW
5 SE
6 NE
7 NW
8 NW
9 SE
10 SE
11 SE
12 SE
13 NW
14 SE
15 SE

```



## XFORM\_MISS\_NUM Procedure

This procedure creates a view that implements the numerical missing value treatment transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
  miss_table_name      IN VARCHAR2,
  data_table_name      IN VARCHAR2,
  xform_view_name      IN VARCHAR2,
  miss_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name     IN VARCHAR2 DEFAULT NULL,
  xform_schema_name    IN VARCHAR2 DEFAULT NULL;
```

### Parameters

**Table 45–48 XFORM\_MISS\_NUM Procedure Parameters**

Parameter	Description
<code>miss_table_name</code>	Name of the transformation definition table for numerical missing value treatment. You can use the <a href="#">CREATE_MISS_NUM Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>XFORM_MISS_NUM</code> . To populate the table, you can use the <a href="#">INSERT_MISS_NUM_MEAN Procedure</a> or you can write your own SQL.  See Table 45–14, "Columns in a Transformation Definition Table for Numerical Missing Value Treatment".
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>miss_table_name</code> .
<code>miss_schema_name</code>	Schema of <code>miss_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See "Operational Notes" on page 45-6.

### Examples

This example creates a view that replaces missing numerical values with the mean.

```
SELECT * FROM items;
```

```
ITEM_ID      QTY
-----
aa           200
bb           200
cc           250
```



```

dd
ee
ff          100
gg          250
hh          200
ii
jj          200

SELECT AVG(qty) FROM items;

AVG(QTY)
-----
      200

BEGIN
  DBMS_DATA_MINING_TRANSFORM.CREATE_MISS_NUM('missnum_xtbl');
  DBMS_DATA_MINING_TRANSFORM.INSERT_MISS_NUM_MEAN (
    miss_table_name      => 'missnum_xtbl',
    data_table_name     => 'items' );
END;
/

SELECT col, val FROM missnum_xtbl;

COL          VAL
-----
QTY          200

BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_MISS_NUM (
    miss_table_name      => 'missnum_xtbl',
    data_table_name     => 'items',
    xform_view_name     => 'items_view');
END;
/

SELECT * FROM items_view;

ITEM_ID      QTY
-----
aa           200
bb           200
cc           250
dd           200
ee           200
ff           100
gg           250
hh           200
ii           200
jj           200

```

## XFORM\_NORM\_LIN Procedure

This procedure creates a view that implements the linear normalization transformations specified in a definition table. Only the columns that are specified in the definition table are transformed; the remaining columns from the data table are present in the view, but they are not changed.

### Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN (
  norm_table_name      IN VARCHAR2,
  data_table_name      IN VARCHAR2,
  xform_view_name      IN VARCHAR2,
  norm_schema_name     IN VARCHAR2 DEFAULT NULL,
  data_schema_name     IN VARCHAR2 DEFAULT NULL,
  xform_schema_name    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 45–49 XFORM\_NORM\_LIN Procedure Parameters**

Parameter	Description
<code>norm_table_name</code>	Name of the transformation definition table for linear normalization. You can use the <a href="#">CREATE_NORM_LIN Procedure</a> to create the definition table. The table must be populated with transformation definitions before you call <code>XFORM_NORM_LIN</code> . To populate the table, you can use one of the <code>INSERT</code> procedures for normalization or you can write your own SQL.  See <a href="#">Table 45–12, "Columns in a Transformation Definition Table for Categorical Missing Value Treatment"</a> .
<code>data_table_name</code>	Name of the table containing the data to be transformed
<code>xform_view_name</code>	Name of the view to be created. The view presents columns in <code>data_table_name</code> with the transformations specified in <code>miss_table_name</code> .
<code>norm_schema_name</code>	Schema of <code>miss_table_name</code> . If no schema is specified, the current schema is used.
<code>data_schema_name</code>	Schema of <code>data_table_name</code> . If no schema is specified, the current schema is used.
<code>xform_schema_name</code>	Schema of <code>xform_view_name</code> . If no schema is specified, the current schema is used.

### Usage Notes

See ["Operational Notes"](#) on page 45-6.

### Examples

This example creates a view that normalizes the `cust_year_of_birth` and `cust_credit_limit` columns. The data source consists of three columns from `sh.customer`.

```
CREATE OR REPLACE VIEW mining_data AS
  SELECT cust_id, cust_year_of_birth, cust_credit_limit
  FROM sh.customers;

describe mining_data
```

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_YEAR_OF_BIRTH	NOT NULL	NUMBER(4)
CUST_CREDIT_LIMIT		NUMBER

```
SELECT * FROM mining_data WHERE cust_id > 104495
ORDER BY cust_year_of_birth;
```

CUST_ID	CUST_YEAR_OF_BIRTH	CUST_CREDIT_LIMIT
104496	1947	3000
104498	1954	10000
104500	1962	15000
104499	1970	3000
104497	1976	3000

```
BEGIN
  dbms_data_mining_transform.CREATE_NORM_LIN(
    norm_table_name=> 'normx_tbl');
  dbms_data_mining_transform.INSERT_NORM_LIN_MINMAX(
    norm_table_name => 'normx_tbl',
    data_table_name => 'mining_data',
    exclude_list => dbms_data_mining_transform.COLUMN_LIST( 'cust_id'),
    round_num => 3);
END;
/
```

```
SELECT col, shift, scale FROM normx_tbl;
```

COL	SHIFT	SCALE
CUST_YEAR_OF_BIRTH	1910	77
CUST_CREDIT_LIMIT	1500	13500

```
BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_NORM_LIN (
    norm_table_name => 'normx_tbl',
    data_table_name => 'mining_data',
    xform_view_name => 'norm_view');
END;
/
```

```
SELECT * FROM norm_view WHERE cust_id > 104495
ORDER BY cust_year_of_birth;
```

CUST_ID	CUST_YEAR_OF_BIRTH	CUST_CREDIT_LIMIT
104496	.4805195	.1111111
104498	.5714286	.6296296
104500	.6753247	1
104499	.7792208	.1111111
104497	.8571429	.1111111

```
set long 2000
```

```
SQL> SELECT text FROM user_views WHERE view_name IN 'NORM_VIEW';
```

```
TEXT
-----
```

```
SELECT "CUST_ID", ("CUST_YEAR_OF_BIRTH"-1910)/77 "CUST_YEAR_OF_BIRTH", ("CUST  
_CREDIT_LIMIT"-1500)/13500 "CUST_CREDIT_LIMIT" FROM mining_data
```

## XFORM\_STACK Procedure

This procedure creates a view that implements the transformations specified by the stack. Only the columns and nested attributes that are specified in the stack are transformed. Any remaining columns and nested attributes from the data table appear in the view without changes.

To create a list of objects that describe the transformed columns, use the [DESCRIBE\\_STACK Procedure](#).

### See Also:

["Overview"](#) on page 45-3

*Oracle Data Mining Application Developer's Guide* for more information about data mining attributes

## Syntax

```
DBMS_DATA_MINING_TRANSFORM.XFORM_STACK (
  xform_list          IN      TRANSFORM_list,
  data_table_name     IN      VARCHAR2,
  xform_view_name     IN      VARCHAR2,
  data_schema_name    IN      VARCHAR2 DEFAULT NULL,
  xform_schema_name   IN      VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 45–50 XFORM\_STACK Procedure Parameters**

Parameter	Description
xform_list	The transformation list. See <a href="#">Table 45–1</a> for a description of the TRANSFORM_LIST object type.
data_table_name	Name of the table containing the data to be transformed
xform_view_name	Name of the view to be created. The view applies the transformations in <i>xform_list</i> to <i>data_table_name</i> .
data_schema_name	Schema of <i>data_table_name</i> . If no schema is specified, the current schema is used.
xform_schema_name	Schema of <i>xform_view_name</i> . If no schema is specified, the current schema is used.

## Usage Notes

See ["Operational Notes"](#) on page 45-6. The following sections are especially relevant:

- ["About Transformation Lists"](#) on page 45-7
- ["About Stacking"](#) on page 45-9
- ["Nested Data Transformations"](#) on page 45-10

## Examples

This example applies a transformation list to the view `dmuser.cust_info` and shows how the data is transformed. The CREATE statement for `cust_info` is shown in ["DESCRIBE\\_STACK Procedure"](#) on page 45-32.

```
BEGIN
  dbms_data_mining_transform.CREATE_BIN_NUM ('birth_yr_bins');
```

```

dbms_data_mining_transform.INSERT_BIN_NUM_QTILE (
    bin_table_name => 'birth_yr_bins',
    data_table_name => 'cust_info',
    bin_num        => 6,
    exclude_list   => dbms_data_mining_transform.column_list(
                        'cust_id','country_id'));

END;
/
SELECT * FROM birth_yr_bins;

COL          ATT          VAL BIN
-----
CUST_YEAR_OF_BIRTH          1922
CUST_YEAR_OF_BIRTH          1951 1
CUST_YEAR_OF_BIRTH          1959 2
CUST_YEAR_OF_BIRTH          1966 3
CUST_YEAR_OF_BIRTH          1973 4
CUST_YEAR_OF_BIRTH          1979 5
CUST_YEAR_OF_BIRTH          1986 6

DECLARE
    cust_stack  dbms_data_mining_transform.TRANSFORM_LIST;
BEGIN
    dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
        'country_id', NULL, 'country_id/10', 'country_id*10');
    dbms_data_mining_transform.STACK_BIN_NUM ('birth_yr_bins',
        cust_stack);
    dbms_data_mining_transform.SET_TRANSFORM (cust_stack,
        'custprods', 'Mouse Pad', 'value*100', 'value/100');
    dbms_data_mining_transform.XFORM_STACK(
        xform_list      => cust_stack,
        data_table_name => 'cust_info',
        xform_view_name => 'cust_xform_view');

END;
/

-- Two rows of data without transformations
SELECT * from cust_info WHERE cust_id BETWEEN 100010 AND 100011;

CUST_ID COUNTRY_ID CUST_YEAR_OF_BIRTH CUSTPRODS(ATTRIBUTE_NAME, VALUE)
-----
100010      52790      1975 DM_NESTED_NUMERICALS(
        DM_NESTED_NUMERICAL(
            '18" Flat Panel Graphics Monitor', 1),
        DM_NESTED_NUMERICAL(
            'SIMM- 16MB PCMCIAII card', 1))
100011      52775      1972 DM_NESTED_NUMERICALS(
        DM_NESTED_NUMERICAL(
            'External 8X CD-ROM', 1),
        DM_NESTED_NUMERICAL(
            'Mouse Pad', 1),
        DM_NESTED_NUMERICAL(
            'SIMM- 16MB PCMCIAII card', 1),
        DM_NESTED_NUMERICAL(
            'Keyboard Wrist Rest', 1),
        DM_NESTED_NUMERICAL(
            '18" Flat Panel Graphics Monitor', 1),
        DM_NESTED_NUMERICAL(
            'O/S Documentation Set - English', 1))

```

```
-- Same two rows of data with transformations
SELECT * FROM cust_xform_view WHERE cust_id BETWEEN 100010 AND 100011;
```

CUST_ID	COUNTRY_ID	C	CUSTPRODS(ATTRIBUTE_NAME, VALUE)
100010	5279	5	DM_NESTED_NUMERICALS( DM_NESTED_NUMERICAL( '18" Flat Panel Graphics Monitor', 1), DM_NESTED_NUMERICAL( 'SIMM- 16MB PCMCIAII card', 1))
100011	5277.5	4	DM_NESTED_NUMERICALS( DM_NESTED_NUMERICAL( 'External 8X CD-ROM', 1), DM_NESTED_NUMERICAL( <b>'Mouse Pad', 100</b> ), DM_NESTED_NUMERICAL( 'SIMM- 16MB PCMCIAII card', 1), DM_NESTED_NUMERICAL( 'Keyboard Wrist Rest', 1), DM_NESTED_NUMERICAL( '18" Flat Panel Graphics Monitor', 1), DM_NESTED_NUMERICAL( 'O/S Documentation Set - English', 1))





---

---

## DBMS\_DATAPUMP

The DBMS\_DATAPUMP package is used to move all, or part of, a database between databases, including both data and metadata.

**See Also:** *Oracle Database Utilities* for more information on the concepts behind the DBMS\_DATAPUMP API, how it works, and how it is implemented in the Data Pump Export and Import utilities

This chapter contains the following topics:

- [Using DBMS\\_DATAPUMP](#)
  - Overview
  - Security Model
  - Constants
- [Data Structures](#)
  - Data Structures - Object Types
- [Summary of DBMS\\_DATAPUMP Subprograms](#)

---

## Using DBMS\_DATAPUMP

This section contains topics that relate to using the DBMS\_DATAPUMP package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)

## Overview

The support and functionality provided by DBMS\_DATAPUMP is as follows:

- The source and target databases can have different hardware, operating systems, character sets, and time zones.
- All object types and data types existing in Oracle Database 11g are supported.
- Data and metadata can be transferred between databases without using any intermediary files.
- A subset of a database can be moved based upon object type and names of objects.
- Schema names, datafile names, and tablespace names can be transformed at import time.
- Previously aborted export and import jobs can be restarted without duplicating or omitting any data or metadata from the original job.
- The resources applied to an export or import job can be modified.
- Data in an Oracle proprietary format can be unloaded and loaded.

## Security Model

Security for the DBMS\_DATAPUMP package is implemented through roles.

### Roles

The DATAPUMP\_EXP\_FULL\_DATABASE and DATAPUMP\_IMP\_FULL\_DATABASE roles allow privileged users to take full advantage of the API. The Data Pump API will use these roles to determine whether privileged application roles should be assigned to the processes comprising the job.

#### **DATAPUMP\_EXP\_FULL\_DATABASE**

The DATAPUMP\_EXP\_FULL\_DATABASE role affects only Export operations. It allows users running these operations to do the following:

- Perform the operation outside of the scope of their schema
- Monitor jobs that were initiated by another user
- Export objects (for example, TABLESPACE definitions) that unprivileged users cannot reference

Although the SYS schema does not have the DATAPUMP\_EXP\_FULL\_DATABASE role assigned to it, all security checks performed by Data Pump that require the DATAPUMP\_EXP\_FULL\_DATABASE role will also grant access to the SYS schema.

#### **DATAPUMP\_IMP\_FULL\_DATABASE**

The DATAPUMP\_IMP\_FULL\_DATABASE role affects only Import and SQL\_FILE operations. It allows users running these operations to do the following:

- Perform the operation outside of the scope of their schema
- Monitor jobs that were initiated by another user
- Import objects (for example, DIRECTORY definitions) that unprivileged users cannot create

Although the SYS schema does not have the DATAPUMP\_IMP\_FULL\_DATABASE role assigned to it, all security checks performed by Data Pump that require the DATAPUMP\_IMP\_FULL\_DATABASE role will also grant access to the SYS schema.

## Constants

There are several public constants defined for use with the `DBMS_DATAPUMP.GET_STATUS` procedure. All such constants are defined as part of the `DBMS_DATAPUMP` package. Any references to these constants must be prefixed by `DBMS_DATAPUMP.` and followed by the symbols in the following lists:

### Mask Bit Definitions

The following mask bit definitions are used for controlling the return of data through the `DBMS_DATAPUMP.GET_STATUS` procedure.

- `KU$_STATUS_WIP`                    `CONSTANT BINARY_INTEGER := 1;`
- `KU$_STATUS_JOB_DESC`            `CONSTANT BINARY_INTEGER := 2;`
- `KU$_STATUS_JOB_STATUS`        `CONSTANT BINARY_INTEGER := 4;`
- `KU$_STATUS_JOB_ERROR`        `CONSTANT BINARY_INTEGER := 8;`

### Dump File Type Definitions

The following definitions are used for identifying types of dump files returned through the `DBMS_DATAPUMP.GET_STATUS` procedure.

- `KU$_DUMPFILE_TYPE_DISK`            `CONSTANT BINARY_INTEGER := 0;`
- `KU$_DUMPFILE_TYPE_TEMPLATE`        `CONSTANT BINARY_INTEGER := 3;`

## Data Structures

The `DBMS_DATAPUMP` package defines `OBJECT` types. The types described in this section are defined in the `SYS` schema for use by the `GET_STATUS` function. *The way in which these types are defined and used may be different than what you are accustomed to. Be sure to read this section carefully.*

The collection of types defined for use with the `GET_STATUS` procedure are version-specific and include version information in the names of the types. Once introduced, these types will always be provided and supported in future versions of Oracle Database and will not change. However, in future releases of Oracle Database, new versions of these types might be created that provide new or different information. The new versions of these types will have different version information embedded in the type names.

For example, in Oracle Database 10g, release 1 (10.1), there is a `sys.ku$_Status1010` type, and in the next Oracle Database release, there could be a `sys.ku$_Status1110` type defined. Both types could be used with the `GET_STATUS` procedure.

Public synonyms have been defined for each of the types used with the `GET_STATUS` procedure. This makes it easier to use the types and means that you do not have to be concerned with changes to the actual type names or schemas where they reside. Oracle recommends that you use these synonyms whenever possible.

For each of the types, there is a version-specific synonym and a generic synonym. For example, the version-specific synonym `ku$_Status1010` is defined for the `sys.ku$_Status1010` type.

The generic synonym always describes the latest version of that type. For example, in Oracle Database 10g, release 1, the generic synonym `ku$_Status` is defined as `ku$_Status1010`. In a future release, there might be a `ku$_Status1110` synonym for `sys.ku$Status1110`. Because the `ku$_Status` generic synonym always points to the latest definition, it would now point to `ku$_Status1110` rather than to `ku$_Status1010`.

The choice of whether to use version-specific synonyms or generic synonyms makes a significant difference in how you work. Using version-specific names protects your code from changes in future releases of Oracle Database because those types will continue to exist and be supported. However, access to new information will require code changes to use new synonym names for each of the types. Using the generic names implies that you always want the latest definition of the types and are prepared to deal with changes in different releases of Oracle Database.

When the version of Oracle Database that you are using changes, any C code that accesses types through generic synonym names will need to be recompiled.

---

---

**Note:** Languages other than PL/SQL must ensure that their type definitions are properly aligned with the version-specific definitions.

---

---

**See Also:** [GET\\_STATUS Procedure](#) on page 46-26 for additional information about how types are used

## Data Structures - Object Types

The DBMS\_DATAPUMP package defines the following kinds of OBJECT types:

- [Worker Status Types](#)
- [Log Entry and Error Types](#)
- [Job Status Types](#)
- [Job Description Types](#)
- [Status Types](#)

### Worker Status Types

The worker status types describe what each worker process in a job is doing. The schema, object name, and object type of an object being processed will be provided. For workers processing user data, the partition name for a partitioned table (if any), the number of bytes processed in the partition, and the number of rows processed in the partition are also returned. Workers processing metadata provide status on the last object that was processed. No status for idle threads is returned.

The `percent_done` refers to the amount completed for the current data item being processed. It is not updated for metadata objects.

The worker status types are defined as follows:

```
CREATE TYPE sys.ku$WorkerStatus1010 AS OBJECT (
    worker_number    NUMBER,
    process_name     VARCHAR2(30),
    state            VARCHAR2(30),
    schema           VARCHAR2(30),
    name             VARCHAR2(4000),
    object_type      VARCHAR2(200),
    partition        VARCHAR2(30),
    completed_objects NUMBER,
    total_objects    NUMBER,
    completed_rows   NUMBER,
    completed_bytes  NUMBER,
    percent_done     NUMBER)

CREATE OR REPLACE PUBLIC SYNONYM ku$WorkerStatus1010
FOR sys.ku$WorkerStatus1010;

CREATE TYPE sys.ku$WorkerStatus1020 AS OBJECT (
    worker_number    NUMBER,          -- Worker process identifier
    process_name     VARCHAR2(30),    -- Worker process name
    state            VARCHAR2(30),    -- Worker process state
    schema           VARCHAR2(30),    -- Schema name
    name             VARCHAR2(4000),  -- Object name
    object_type      VARCHAR2(200),   -- Object type
    partition        VARCHAR2(30),    -- Partition name
    completed_objects NUMBER,         -- Completed number of objects
    total_objects    NUMBER,         -- Total number of objects
    completed_rows   NUMBER,         -- Number of rows completed
    completed_bytes  NUMBER,         -- Number of bytes completed
    percent_done     NUMBER,         -- Percent done current object
    degree           NUMBER           -- Degree of parallelism)

CREATE OR REPLACE PUBLIC SYNONYM ku$WorkerStatus1020
FOR sys.ku$WorkerStatus1020;
```

```
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus FOR ku$_WorkerStatus1020;

CREATE TYPE sys.ku$_WorkerStatusList1010 AS TABLE OF sys.ku$_WorkerStatus1010

CREATE TYPE sys.ku$_WorkerStatusList1020 AS TABLE OF sys.ku$_WorkerStatus1020

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1010
  FOR sys.ku$_WorkerStatusList1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1020
  FOR sys.ku$_WorkerStatusList1020;

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList
  FOR ku$_WorkerStatusList1020;
```

## Log Entry and Error Types

These types provide informational and error text to attached clients and the log stream. The `ku$LogLine.errorNumber` type is set to NULL for informational messages but is specified for error messages. Each log entry may contain several lines of text messages.

The log entry and error types are defined as follows:

```
CREATE TYPE sys.ku$_LogLine1010 AS OBJECT (
    logLineNumber    NUMBER,
    errorNumber      NUMBER,
    LogText          VARCHAR2(2000))

CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine1010 FOR sys.ku$_LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine1020 FOR sys.ku$_LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine FOR ku$_LogLine1010;
CREATE TYPE sys.ku$_LogEntry1010 AS TABLE OF sys.ku$_LogLine1010

CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry1010 FOR sys.ku$_LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry1020 FOR sys.ku$_LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry FOR ku$_LogEntry1010;
```

## Job Status Types

The job status type returns status about a job. Usually, the status concerns a running job but it could also be about a stopped job when a client attaches. It is typically requested at attach time, when the client explicitly requests status from interactive mode and every N seconds when the client has requested status periodically.

The job status types are defined as follows (`percent_done` applies to data only):

```
CREATE TYPE sys.ku$_DumpFile1010 IS OBJECT (
    file_name        VARCHAR2(4000), -- Fully-qualified name
    file_type        NUMBER,         -- 0=Disk, 1=Pipe, etc.
    file_size        NUMBER,         -- Its length in bytes
    file_bytes_written NUMBER        -- Bytes written so far)

CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFile1010 FOR sys.ku$_DumpFile1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFile1020 FOR sys.ku$_DumpFile1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFile FOR ku$_DumpFile1010;

CREATE TYPE sys.ku$_DumpFileSet1010 AS TABLE OF sys.ku$_DumpFile1010;

CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFileSet1010 FOR
  sys.ku$_DumpFileSet1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFileSet1020 FOR
```



```

sys.ku$_DumpFileSet1010;

CREATE OR REPLACE PUBLIC SYNONYM ku$_DumpFileSet FOR ku$_DumpFileSet1010;

CREATE TYPE sys.ku$_JobStatus1010 IS OBJECT (
    job_name          VARCHAR2(30),
    operation          VARCHAR2(30),
    job_mode           VARCHAR2(30),
    bytes_processed    NUMBER,
    percent_done       NUMBER,
    degree             NUMBER,
    error_count        NUMBER,
    state              VARCHAR2(30),
    phase              NUMBER,
    restart_count      NUMBER,
    worker_status_list ku$_WorkerStatusList1010,
    files              ku$_DumpFileSet1010)

CREATE PUBLIC SYNONYM ku$_JobStatus1010 FOR
    sys.ku$_JobStatus1010;

CREATE TYPE sys.ku$_JobStatus1020 IS OBJECT (
    job_name          VARCHAR2(30),           -- Name of the job
    operation          VARCHAR2(30),         -- Current operation
    job_mode           VARCHAR2(30),         -- Current mode
    bytes_processed    NUMBER,               -- Bytes so far
    total_bytes        NUMBER,               -- Total bytes for job
    percent_done       NUMBER,               -- Percent done
    degree             NUMBER,               -- Of job parallelism
    error_count        NUMBER,               -- #errors so far
    state              VARCHAR2(30),         -- Current job state
    phase              NUMBER,               -- Job phase
    restart_count      NUMBER,               -- #Job restarts
    worker_status_list ku$_WorkerStatusList1020, -- job worker processes
    files              ku$_DumpFileSet1010  -- Dump file info)

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1020 FOR    sys.ku$_JobStatus1020;

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus FOR ku$_JobStatus1020;

```

## Job Description Types

The job description type holds all the environmental information about the job such as parameter settings and dump file set members. There are a couple of subordinate types required as well.

The job description types are defined as follows:

```

CREATE TYPE sys.ku$_ParamValue1010 AS OBJECT (
    param_name        VARCHAR2(30),
    param_op          VARCHAR2(30),
    param_type        VARCHAR2(30),
    param_length      NUMBER,
    param_value_n     NUMBER,
    param_value_t     VARCHAR2(4000));

CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValue1010 FOR sys.ku$_ParamValue1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValue1020 FOR sys.ku$_ParamValue1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValue FOR ku$_ParamValue1010;

CREATE TYPE sys.ku$_ParamValues1010 AS TABLE OF sys.ku$_ParamValue1010;

```

```

CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValues1010 FOR
  sys.ku$_ParamValues1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValues1020 FOR
  sys.ku$_ParamValues1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_ParamValues FOR ku$_ParamValues1010;

CREATE TYPE sys.ku$_JobDesc1010 AS OBJECT (
  job_name      VARCHAR2(30),
  guid          RAW(16),
  operation     VARCHAR2(30),
  job_mode     VARCHAR2(30),
  remote_link  VARCHAR2(4000),
  owner        VARCHAR2(30),
  instance     VARCHAR2(16),
  db_version   VARCHAR2(30),
  creator_privs VARCHAR2(30),
  start_time   DATE,
  max_degree   NUMBER,
  log_file     VARCHAR2(4000),
  sql_file     VARCHAR2(4000),
  params       ku$_ParamValues1010)

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1010 FOR sys.ku$_JobDesc1010;

CREATE TYPE sys.ku$_JobDesc1020 IS OBJECT (
  job_name      VARCHAR2(30),      -- The job name
  guid          RAW(16),          -- The job GUID
  operation     VARCHAR2(30),      -- Current operation
  job_mode     VARCHAR2(30),      -- Current mode
  remote_link  VARCHAR2(4000),    -- DB link, if any
  owner        VARCHAR2(30),      -- Job owner
  platform     VARCHAR2(101),     -- Current job platform
  exp_platform VARCHAR2(101),     -- Export platform
  global_name  VARCHAR2(4000),    -- Global name of DB
  exp_global_name VARCHAR2(4000), -- Export global name
  instance     VARCHAR2(16),      -- The instance name
  db_version   VARCHAR2(30),      -- Version of objects
  exp_db_version VARCHAR2(30),    -- Export version
  scn          NUMBER,           -- Job SCN
  creator_privs VARCHAR2(30),     -- Privs of job
  start_time   DATE,             -- This job start time
  exp_start_time DATE,           -- Export start time
  term_reason  NUMBER,           -- Job termination code
  max_degree   NUMBER,           -- Max. parallelism
  log_file     VARCHAR2(4000),    -- Log file name
  sql_file     VARCHAR2(4000),    -- SQL file name
  params       ku$_ParamValues1010 -- Parameter list)

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1020 FOR sys.ku$_JobDesc1020;
CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc FOR ku$_JobDesc1020;

```

## Status Types

The status type is an aggregate of some the previous types defined and is the return value for the GET\_STATUS call. The mask attribute indicates which types of information are being returned to the caller. It is created by a client's shadow process from information it retrieves off the status queue or directly from the master table.

For errors, the `ku$_LogEntry` that is returned has already had its log lines ordered for proper output. That is, the original `ku$_LogEntry` objects have been ordered from outermost context to innermost.

The status types are defined as follows:

```
CREATE TYPE sys.ku$_Status1010 AS OBJECT
(
  mask NUMBER,          /* Indicates which status types are present*/
  wip ku$_LogEntry1010, /* Work-In-Progress: std. exp/imp msgs */
  job_description ku$_JobDesc1010, /* Complete job description */
  job_status ku$_JobStatus1010, /* Detailed job status + per-worker sts */
  error ku$_LogEntry1010 /* Multi-level contextual errors */
)

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1010 FOR sys.ku$_Status1010;

CREATE TYPE sys.ku$_Status1020 IS OBJECT
(
  mask NUMBER,          -- Status types present
  wip ku$_LogEntry1010, -- Work in progress
  job_description ku$_JobDesc1020, -- Complete job description
  job_status ku$_JobStatus1020, -- Detailed job status
  error ku$_LogEntry1010 -- Multi-level context errors
)

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1020 FOR sys.ku$_Status1020;

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status FOR ku$_Status1020;
```

---

## Summary of DBMS\_DATAPUMP Subprograms

**Table 46–1 DBMS\_DATAPUMP Package Subprograms**

Subprogram	Description
<a href="#">ADD_FILE Procedure</a> on page 46-13	Adds dump files to the dump file set for an Export, Import, or SQL_FILE operation. In addition to dump files, other types of files can also be added by using the FILETYPE parameter provided with this procedure.
<a href="#">ATTACH Function</a> on page 46-16	Used to gain access to a Data Pump job that is in the Defining, Executing, Idling, or Stopped state
<a href="#">DATA_FILTER Procedures</a> on page 46-18	Specifies restrictions on the rows that are to be retrieved
<a href="#">DETACH Procedure</a> on page 46-23	Specifies that the user has no further interest in using the handle
<a href="#">GET_DUMPFILE_INFO Procedure</a> on page 46-24	Retrieves information about a specified dump file
<a href="#">GET_STATUS Procedure</a> on page 46-26	Monitors the status of a job or waits for the completion of a job or for more details on API errors
<a href="#">LOG_ENTRY Procedure</a> on page 46-29	Inserts a message into the log file
<a href="#">METADATA_FILTER Procedure</a> on page 46-30	Provides filters that allow you to restrict the items that are included in a job
<a href="#">METADATA_REMAP Procedure</a> on page 46-33	Specifies a remapping to be applied to objects as they are processed in the specified job
<a href="#">METADATA_TRANSFORM Procedure</a> on page 46-36	Specifies transformations to be applied to objects as they are processed in the specified job
<a href="#">OPEN Function</a> on page 46-39	Declares a new job using the Data Pump API, the handle returned being used as a parameter for calls to all other procedures (but not to the ATTACH function)
<a href="#">SET_PARALLEL Procedure</a> on page 46-42	Adjusts the degree of parallelism within a job
<a href="#">SET_PARAMETER Procedures</a> on page 46-44	Specifies job-processing options
<a href="#">START_JOB Procedure</a> on page 46-53	Begins or resumes execution of a job
<a href="#">STOP_JOB Procedure</a> on page 46-55	Terminates a job, but optionally, preserves the state of the job
<a href="#">WAIT_FOR_JOB Procedure</a> on page 46-57	Runs a job until it either completes normally or stops for some other reason

## ADD\_FILE Procedure

This procedure adds files to the dump file set for an Export, Import, or SQL\_FILE operation or specifies the log file or the output file for a SQL\_FILE operation.

### Syntax

```
DBMS_DATAPUMP.ADD_FILE (
  handle      IN NUMBER,
  filename    IN VARCHAR2,
  directory   IN VARCHAR2,
  filesize    IN VARCHAR2 DEFAULT NULL,
  filetype    IN NUMBER DEFAULT DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE),
  reusefile   IN NUMBER DEFAULT NULL;
```

### Parameters

**Table 46–2 ADD\_FILE Procedure Parameters**

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.
filename	The name of the file being added. <code>filename</code> must be a simple filename without any directory path information. For dump files, the <code>filename</code> can include a substitution variable, <code>%U</code> , which indicates that multiple files may be generated with the specified <code>filename</code> as a template. The <code>%U</code> is expanded in the resulting file names into a two-character, fixed-width, incrementing integer starting at 01. For example, the dump filename of <code>export%U</code> would cause <code>export01</code> , <code>export02</code> , <code>export03</code> , and so on, to be created depending on how many files are needed to perform the export. For filenames containing the <code>%</code> character, the <code>%</code> must be represented as <code>%%</code> to avoid ambiguity. Any <code>%</code> in a filename must be followed by either a <code>%</code> or a <code>U</code> .
directory	The name of a directory object within the database that is used to locate <code>filename</code> . A <code>directory</code> must be specified. See the Data Pump Export chapter in <i>Oracle Database Utilities</i> for information about the DIRECTORY command-line parameter.
filesize	The size of the dump file that is being added. It may be specified as the number of bytes, number of kilobytes (if followed by K), number of megabytes (if followed by M) or number of gigabytes (if followed by G). An Export operation will write no more than the specified number of bytes to the file. Once the file is full, it will be closed. If there is insufficient space on the device to write the specified number of bytes, the Export operation will fail, but it can be restarted. If not specified, <code>filesize</code> will default to an unlimited size. For Import and SQL_FILE operations, <code>filesize</code> is ignored. The minimum value for <code>filesize</code> is ten times the default Data Pump block size, which is 4 kilobytes. A <code>filesize</code> can only be specified for dump files.
filetype	The type of the file to be added. The legal values are as follows and must be preceded by <code>DBMS_DATAPUMP.:</code> <ul style="list-style-type: none"> <li>▪ <code>KU\$_FILE_TYPE_DUMP_FILE</code> (dump file for a job)</li> <li>▪ <code>KU\$_FILE_TYPE_LOG_FILE</code> (log file for a job)</li> <li>▪ <code>KU\$_FILE_TYPE_SQL_FILE</code> (output for SQL_FILE job)</li> </ul>

**Table 46–2 (Cont.) ADD\_FILE Procedure Parameters**

Parameter	Description
<code>reusefile</code>	If 0, a preexisting file will cause an error. If 1, a preexisting file will be overwritten. If NULL, the default action for the file type will be applied (that is, dump files will not be overwritten). This parameter should only be non-NULL for dump files. The <code>reusefile</code> parameter is restricted to export jobs.

## Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_ARGVAL`. An invalid value was supplied for an input parameter.
- `INVALID_STATE`. The job is completing, or the job is past the defining state for an import or `SQL_FILE` job or is past the defining state for LOG and SQL files.
- `INVALID_OPERATION`. A dump file was specified for a Network Import or `ESTIMATE_ONLY` export operation.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

## Usage Notes

- Adds files to a Data Pump job. Three types of files may be added to jobs: Dump files to contain the data that is being moved, log files to record the messages associated with an operation, and SQL files to record the output of a `SQL_FILE` operation. Log and SQL files will overwrite previously existing files. Dump files will never overwrite previously existing files. Instead, an error will be generated.
- Import and `SQL_FILE` operations require that all dump files be specified during the definition phase of the job. For Export operations, dump files can be added at any time. For example, if the user ascertains that the file space is running low during an Export, additional dump files may be added through this API. If the specified dump file already exists for an Export operation and `reusefile` is not set to 1, an error will be returned.
- For Export operations, the parallelism setting should be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the job will not be able to maximize parallelism to the degree specified by the `SET_PARALLEL` procedure.
- For Import operations, the parallelism setting should also be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the performance will not be optimal as multiple threads of execution try to access the same dump file.
- If the substitution variable (`%U`) is included in a filename, multiple dump files may be specified through a single call to `ADD_FILE`. For Export operations, the new dump files will be created as they are needed. Enough dump files will be created to allow all of the processes specified by the current `SET_PARALLEL` value to be active. If one of the dump files fills, it will be closed and a new dump file (with a new generated name) will be created to take its place. If multiple `ADD_FILES` with substitution variables have been specified for dump files in a job, they will be used to generate dump files in a round robin fashion. For example, if `expa%U`, `expb%U` and `expc%U` were all specified for a job having a parallelism of 6, the initial dump

files created would look like: `expa01`, `expb01`, `expc01`, `expa02`, `expb02`, and `expc02`.

- If presented with dump file specifications, `expa%U`, `expb%U` and `expc%U`, an Import or `SQL_FILE` operation will begin by attempting to open the dump files, `expa01`, `expb01`, and `expc01`. If the dump file containing the master table is not found in this set, the operation will expand its search for dump files by incrementing the substitution variable and looking up the new filenames (for example, `expa02`, `expb02`, and `expc02`). The DataPump API will keep expanding the search until it locates the dump file containing the master table. If the DataPump API determines that the dump file does not exist or is not part of the current dump set at any iteration, the DataPump API will stop incrementing the substitution variable for the dump file specification that was in error. Once the master table is found, the master table will be used to ascertain when all of dump files in the dump file set have been located.

## ATTACH Function

This function gains access to a previously-created job.

### Syntax

```
DBMS_DATAPUMP.ATTACH(
  job_name      IN VARCHAR2 DEFAULT NULL,
  job_owner     IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

### Parameters

**Table 46–3** ATTACH Function Parameters

Parameter	Description
job_name	The name of the job. The default is the job name owned by the user who is specified in the job_owner parameter (assuming that user has only one job in the Defining, Executing, or Idling states).
job_owner	The user who originally started the job. If NULL, the value defaults to the owner of the current session. To specify a job owner other than yourself, you must have either the DATAPUMP_EXP_FULL_DATABASE role (for export operations) or the DATAPUMP_IMP_FULL_DATABASE role (for import and SQL_FILE operations). Being a privileged user allows you to monitor another user's job, but you cannot restart another user's job.

### Return Values

An opaque handle for the job. This handle is used as input to the following procedures: ADD\_FILE, DATA\_FILTER, DETACH, GET\_STATUS, LOG\_ENTRY, METADATA\_FILTER, METADATA\_REMAP, METADATA\_TRANSFORM, SET\_PARALLEL, SET\_PARAMETER, START\_JOB, STOP\_JOB, and WAIT\_FOR\_JOB.

### Exceptions

- INVALID\_ARGVAL. An invalid value was supplied for an input parameter.
- OBJECT\_NOT\_FOUND. The specified job no longer exists or the user specified a job owned by another schema, but the user did not have the DATAPUMP\_EXP\_FULL\_DATABASE or DATAPUMP\_IMP\_FULL\_DATABASE role.
- SUCCESS\_WITH\_INFO. The function succeeded, but further information is available through the GET\_STATUS procedure.
- NO\_SUCH\_JOB. The specified job does not exist.

### Usage Notes

- If the job was in the Stopped state, the job is placed into the Idling state. Once the ATTACH succeeds, you can monitor the progress of the job or control the job. The stream of KU\$\_STATUS\_WIP and KU\$\_STATUS\_JOB\_ERROR messages returned through the GET\_STATUS procedure will be returned to the newly attached job starting at the approximate time of the client's attachment. There will be no repeating of status and error messages that were processed before the client attached to a job.
- If you want to perform a second attach to a job, you must do so from a different session.



- If the `ATTACH` fails, use a null handle in a subsequent call to `GET_STATUS` for more information about the failure.

## DATA\_FILTER Procedures

This procedure specifies restrictions on the rows that are to be retrieved.

### Syntax

```
DBMS_DATAPUMP.DATA_FILTER (
    handle      IN NUMBER,
    name        IN VARCHAR2,
    value       IN NUMBER,
    table_name  IN VARCHAR2 DEFAULT NULL,
    schema_name IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_DATAPUMP.DATA_FILTER(
    handle      IN NUMBER,
    name        IN VARCHAR2,
    value       IN VARCHAR2,
    table_name  IN VARCHAR2 DEFAULT NULL,
    schema_name IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_DATAPUMP.DATA_FILTER(
    handle      IN NUMBER,
    name        IN VARCHAR2,
    value       IN CLOB,
    table_name  IN VARCHAR2 DEFAULT NULL,
    schema_name IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 46–4 DATA\_FILTER Procedure Parameters**

Parameter	Description
handle	The handle that is returned from the OPEN function
name	The name of the filter
value	The value of the filter
table_name	The name of the table on which the data filter is applied. If no table name is supplied, the filter applies to all tables in the job.
schema_name	The name of the schema that owns the table on which the filter is applied. If no schema name is specified, the filter applies to all schemas in the job. If you supply a schema name you must also supply a table name.

### Exceptions

- **INVALID\_ARGVAL.** There can be several reasons for this message:
  - A bad filter name is specified
  - The mode is `TRANSPORTABLE`, which does not support data filters
  - The specified table does not exist
  - The filter has already been set for the specified values of `schema_name` and `table_name`
- **INVALID\_STATE.** The user called `DATA_FILTER` when the job was not in the Defining state.

- `INCONSISTENT_ARGS`. The `value` parameter is missing or its datatype does not match the filter name. Or a schema name was supplied, but not a table name.
- `PRIVILEGE_ERROR`. A schema name was supplied, but the user did not have the `DATAPUMP_EXP_FULL_DATABASE` or `DATAPUMP_IMP_FULL_DATABASE` role.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

## Usage Notes

- Each data filter can only appear once in each table (for example, you cannot supply multiple `SUBQUERY` filters to a table) or once in each job. If different filters using the same name are applied to both a particular table and to the whole job, the filter parameter supplied for the specific table will take precedence.

With the exception of the `INCLUDE_ROWS` filter, data filters are not supported on tables having nested tables or domain indexes defined upon them. Data filters are not supported in jobs performed in Transportable Tablespace mode.

The available data filters are described in [Table 46-5](#).

**Table 46-5 Data Filters**

Name	Datatype	Operations that Support Filter	Description
<code>INCLUDE_ROWS</code>	NUMBER	EXPORT, IMPORT	If nonzero, this filter specifies that user data for the specified table should be included in the job. The default is 1.
<code>PARTITION_EXPR</code> <code>PARTITION_LIST</code>	text	EXPORT, IMPORT	For Export jobs, these filters specify which partitions are unloaded from the database. For Import jobs, they specify which table partitions are loaded into the database. Partition names are included in the job if their names satisfy the specified expression (for <code>PARTITION_EXPR</code> ) or are included in the list (for <code>PARTITION_LIST</code> ). Whereas the expression version of the filter offers more flexibility, the list version provides for full validation of the partition names.  Double quotation marks around partition names are required only if the partition names contain special characters.  <code>PARTITION_EXPR</code> is not supported on jobs across a network link.  Default=All partitions are processed
<code>SAMPLE</code>	NUMBER	EXPORT, IMPORT	For Export jobs, specifies a percentage for sampling the data blocks to be moved. This filter allows subsets of large tables to be extracted for testing purposes.

**Table 46–5 (Cont.) Data Filters**

<b>Name</b>	<b>Datatype</b>	<b>Operations that Support Filter</b>	<b>Description</b>
SUBQUERY	text	EXPORT, IMPORT	Specifies a subquery that is added to the end of the SELECT statement for the table. If you specify a WHERE clause in the subquery, you can restrict the rows that are selected. Specifying an ORDER BY clause orders the rows dumped in the export which improves performance when migrating from heap-organized tables to index-organized tables.

## DATA\_REMAP Procedure

This procedure specifies transformations to be applied to column data as it is exported from, or imported into, a database.

### Syntax

```
DBMS_DATAPUMP.DATA_REMAP (
  handle          IN NUMBER,
  name            IN VARCHAR2,
  table_name      IN VARCHAR2,
  column          IN VARCHAR2,
  remap_function  IN VARCHAR2),
  schema         IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 46–6 DATA\_REMAP Procedure Parameters**

Parameter	Description
handle	The handle of the current job. The current session must have previously attached to the handle through a call to an OPEN function.
name	The name of the remap
table_name	The table containing the column to be remapped
column	The name of the column to be remapped
remap_function	The meaning of remap_function is dependent upon the value of name. See <a href="#">Table 46–7</a> for a list of possible names.
schema	The schema containing the column to be remapped. If NULL, the remapping applies to all schemas moved in the job that contain the specified table.

### Exceptions

- **INVALID\_ARGVAL.** The mode is transportable (which does not support data modifications) or it has specified that no data to be included in the job. An invalid remap name was supplied.
- **INVALID\_OPERATION.** Data remaps are only supported for Export and Import operations.
- **INVALID\_STATE.** The DATA\_REMAP procedure was called after the job started (that is, it was not in the defining state).
- **NO\_SUCH\_JOB.** The job handle is no longer valid.

### Usage Notes

- The DATA\_REMAP procedure is only supported for Export and Import operations. It allows you to manipulate user data being exported or imported. The name of the remap determines the remap operation to be performed.
- For export operations, you might wish to define a data remap to obscure sensitive data such as credit card numbers from a dump file, but leave the remainder of the data so that it can be read. To accomplish this, the remapping should convert each unique source number into a distinct generated number. So that the mapping is

consistent across the dump file set, the same function should be called for every column that contains the credit card number.

- For import operations, you might wish to define a data remap to reset the primary key when data is being merged into an existing table that contains colliding primary keys. A single remapping function should be provided for all columns defining or referencing the primary key to ensure that remapping is consistent.

---

**Note:** If the called function uses package state variables, then to ensure that remapping is performed consistently across all tables, the job should be run with a SET\_PARALLEL value of 1 and no restart operations should be performed.

---

The Data Remap functions are listed in [Table 46–7](#).

**Table 46–7** *Names of Data Remap Functions*

Name	Meaning of remap_ function	Meaning
COLUMN_FUNCTION	String having the format: [schema.]package.func tion	The name parameter references a PL/SQL package function which is called to modify the data for the specified column. The function accepts a single parameter, which has the same datatype as the remapped column, and returns a value having the same datatype as the remapped column. Note that the default for the schema is the schema of the user performing the export.

## DETACH Procedure

This procedure specifies that the user has no further interest in using the handle.

### Syntax

```
DBMS_DATAPUMP.DETACH(  
    handle IN NUMBER);
```

### Parameters

**Table 46–8** *DETACH Procedure Parameters*

Parameter	Description
handle	The handle of the job. The current session must have previously attached to the handle through a call to either an <code>OPEN</code> or <code>ATTACH</code> function.

### Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

### Usage Notes

- Through this call, you specify that you have no further interest in using the handle. Resources associated with a completed job cannot be reclaimed until all users are detached from the job. An implicit detach from a handle is performed when the user's session is exited or aborted. An implicit detach from a handle is also performed upon the expiration of the timeout associated with a `STOP_JOB` that was applied to the job referenced by the handle. All previously allocated `DBMS_DATAPUMP` handles are released when an instance is restarted.

## GET\_DUMPFILE\_INFO Procedure

This procedure retrieves information about a specified dump file.

### Syntax

```
DBMS_DATAPUMP.GET_DUMPFILE_INFO(
  filename      IN VARCHAR2,
  directory     IN VARCHAR2,
  info_table    OUT ku$_dumpfile_info,
  filetype      OUT NUMBER);
```

### Parameters

**Table 46–9** GET\_DUMPFILE\_INFO Procedure Parameters

Parameter	Description
filename	A simple filename with no directory path information
directory	A directory object that specifies where the file can be found
info_table	A PL/SQL table for storing information about the dump file
filetype	The type of file (Data Pump dump file, original Export dump file, or unknown)

### Exceptions

The GET\_DUMPFILE\_INFO procedure is a utility routine that operates outside the context of any Data Pump job. Exceptions are handled differently for this procedure than for procedures associated in some way with a Data Pump job. A full exception stack should be available directly, without the need to call the GET\_STATUS procedure to retrieve the detailed information. The exception for this procedure is as follows:

- NO\_DUMPFILE\_INFO. Unable to retrieve dump file information as specified.

### Usage Notes

You can use the GET\_DUMPFILE\_INFO procedure to request information about a specific file. If the file is not recognized as any type of dump file, then a filetype of zero will be returned and the dump file info\_table will remain empty.

A filetype value of one indicates a Data Pump dump file. A file type value of two indicates an original Export dump file. In both cases, the dump file info\_table will be populated with information retrieved from the dump file header. Rows of this table consist of item code and value pairs, where the item code indicates the type of information and the value column is a VARCHAR2 containing the actual data (converted to a string in some cases). The table is defined as follows:

```
CREATE TYPE sys.ku$_dumpfile_item IS OBJECT (
  item_code    NUMBER,          -- Identifies header item
  value        VARCHAR2(2048)  -- Text string value)
/

GRANT EXECUTE ON sys.ku$_dumpfile_item TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_item FOR sys.ku$_dumpfile_item;

CREATE TYPE sys.ku$_dumpfile_info AS TABLE OF sys.ku$_dumpfile_item
/
```



```
GRANT EXECUTE ON sys.ku$_dumpfile_info TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_info FOR sys.ku$_dumpfile_info;
```

The item codes, which can easily be extended to provide more information as needed, are currently defined as follows (prepended with the package name, DBMS\_DATAPUMP.):

KU\$DFHDR_FILE_VERSION	CONSTANT NUMBER := 1;
KU\$DFHDR_MASTER_PRESENT	CONSTANT NUMBER := 2;
KU\$DFHDR_GUID	CONSTANT NUMBER := 3;
KU\$DFHDR_FILE_NUMBER	CONSTANT NUMBER := 4;
KU\$DFHDR_CHARSET_ID	CONSTANT NUMBER := 5;
KU\$DFHDR_CREATION_DATE	CONSTANT NUMBER := 6;
KU\$DFHDR_FLAGS	CONSTANT NUMBER := 7;
KU\$DFHDR_JOB_NAME	CONSTANT NUMBER := 8;
KU\$DFHDR_PLATFORM	CONSTANT NUMBER := 9;
KU\$DFHDR_INSTANCE	CONSTANT NUMBER := 10;
KU\$DFHDR_LANGUAGE	CONSTANT NUMBER := 11;
KU\$DFHDR_BLOCKSIZE	CONSTANT NUMBER := 12;
KU\$DFHDR_DIRPATH	CONSTANT NUMBER := 13;
KU\$DFHDR_METADATA_COMPRESSED	CONSTANT NUMBER := 14;
KU\$DFHDR_DB_VERSION	CONSTANT NUMBER := 15;
KU\$DFHDR_MAX_ITEM_CODE	CONSTANT NUMBER := 20;
KU\$DFHDR_MASTER_PIECE_COUNT	CONSTANT NUMBER := 16;
KU\$DFHDR_MASTER_PIECE_NUMBER	CONSTANT NUMBER := 17;
KU\$DFHDR_DATA_COMPRESSED	CONSTANT NUMBER := 18;
KU\$DFHDR_METADATA_ENCRYPTED	CONSTANT NUMBER := 19;
KU\$DFHDR_DATA_ENCRYPTED	CONSTANT NUMBER := 20;

## GET\_STATUS Procedure

This procedure monitors the status of a job or waits for the completion of a job.

### Syntax

```
DBMS_DATAPUMP.GET_STATUS (
  handle      IN NUMBER,
  mask        IN BINARY_INTEGER,
  timeout     IN NUMBER DEFAULT NULL,
  job_state   OUT VARCHAR2,
  status      OUT ku$_Status1010);
```

### Parameters

**Table 46–10** GET\_STATUS Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the <code>OPEN</code> or <code>ATTACH</code> function. A null handle can be used to retrieve error information after <code>OPEN</code> and <code>ATTACH</code> failures.
mask	A bit mask that indicates which of four types of information to return: <ul style="list-style-type: none"> <li>▪ <code>KU\$_STATUS_WIP</code></li> <li>▪ <code>KU\$_STATUS_JOB_DESC</code></li> <li>▪ <code>KU\$_STATUS_JOB_STATUS</code></li> <li>▪ <code>KU\$_STATUS_JOB_ERROR</code></li> </ul> Each status has a numerical value. You can request multiple types of information by adding together different combinations of values. See <a href="#">Data Structures - Object Types</a> on page 46-7.
timeout	Maximum number of seconds to wait before returning to the user. A value of 0 requests an immediate return. A value of -1 requests an infinite wait. If <code>KU\$_STATUS_WIP</code> or <code>KU\$_STATUS_JOB_ERROR</code> information is requested and becomes available during the timeout period, then the procedure returns before the timeout period is over.
job_state	Current state of the job. If only the job state is needed, it is much more efficient to use this parameter than to retrieve the full <code>ku\$_Status</code> structure.
status	A <code>ku\$_Status</code> is returned. The <code>ku\$_Status</code> mask indicates what kind of information is included. This could be none if only <code>KU\$_STATUS_WIP</code> or <code>KU\$_STATUS_JOB_ERROR</code> information is requested and the timeout period expires. This can be a <code>ku\$_Status1010</code> or <code>ku\$_Status1020</code> object type.

### Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_VALUE`. The mask or timeout contains an illegal value.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

## Usage Notes

The `GET_STATUS` procedure is used to monitor the progress of an ongoing job and to receive error notification. You can request various type of information using the mask parameter. The `KU$STATUS_JOB_DESC` and `KU$STATUS_JOB_STATUS` values are classified as synchronous information because the information resides in the master table. The `KU$STATUS_WIP` and `KU$STATUS_JOB_ERROR` values are classified as asynchronous because the messages that embody these types of information can be generated at any time by various layers in the Data Pump architecture.

- If synchronous information *only* is requested, the interface will ignore the timeout parameter and simply return the requested information.
- If asynchronous information is requested, the interface will wait a *maximum* of timeout seconds before returning to the client. If a message of the requested asynchronous information type is received, the call will complete prior to timeout seconds. If synchronous information was also requested, it will be returned whenever the procedure returns.
- If the `job_state` returned by `GET_STATUS` does not indicate a terminating job, it is possible that the job could still terminate before the next call to `GET_STATUS`. This would result in an `INVALID_HANDLE` exception. Alternatively, the job could terminate during the call to `GET_STATUS`, which would result in a `NO_SUCH_JOB` exception. Callers should be prepared to handle these cases.

## Error Handling

There are two types of error scenarios that need to be handled using the `GET_STATUS` procedure:

- Errors resulting from other procedure calls: For example, the `SET_PARAMETER` procedure may produce an `INCONSISTENT_ARGS` exception. The client should immediately call `GET_STATUS` with `mask=8` (errors) and `timeout=0`. The returned `ku$Status.error` will contain a `ku$LogEntry` that describes the inconsistency in more detail.
- Errors resulting from events asynchronous to the client(s): An example might be `Table already exists` when trying to create a table. The `ku$Status.error` will contain a `ku$LogEntry` with all error lines (from all processing layers that added context about the error) properly ordered.

After a job has begun, a client's main processing loop will typically consist of a call to `GET_STATUS` with an infinite timeout (-1) "listening" for `KU$STATUS_WIP` and `KU$STATUS_JOB_ERROR` messages. If status was requested, then `JOB_STATUS` information will also be in the request.

When the `ku$Status` is interpreted, the following guidelines should be used:

- `ku$Status.ku$JobStatus.percent_done` refers only to the amount of data that has been processed in a job. Metadata is not considered in the calculation. It is determined using the following formulas:
  - `EXPORT` or `network IMPORT`--  $(\text{bytes\_processed} / \text{estimated\_bytes}) * 100$
  - `IMPORT`--  $(\text{bytes\_processed} / \text{total\_expected\_bytes}) * 100$
  - `SQL_FILE` or `estimate-only EXPORT`-- 0.00 if not done or 100.00 if done

The effects of the `QUERY` and `PARTITION_EXPR` data filters are not considered in computing `percent_done`.

It is expected that the status returned will be transformed by the caller into more user-friendly status. For example, when percent done is not zero, an estimate of completion time could be produced using the following formula:

```
((SYSDATE - start time) / ku$_Status.ku$_JobStatus.percent_done) * 100
```

- The caller should not use `ku$_Status.ku$_JobStatus.percent_done` for determining whether the job has completed. Instead, the caller should only rely on the state of the job as found in `job_state`.

## LOG\_ENTRY Procedure

This procedure inserts a message into the log file.

### Syntax

```
DBMS_DATAPUMP.LOG_ENTRY (
    handle          IN NUMBER,
    message         IN VARCHAR2
    log_file_only  IN NUMBER DEFAULT 0);
```

### Parameters

**Table 46–11 LOG\_ENTRY Procedure Parameters**

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.
message	A text line to be added to the log file
log_file_only	Specified text should be written only to the log file. It should not be returned in GET_STATUS work-in-progress (KU\$ STATUS_WIP) messages.

### Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- SUCCESS\_WITH\_INFO. The procedure succeeded, but further information is available through the GET\_STATUS procedure.
- NO\_SUCH\_JOB. The specified job does not exist.

### Usage Notes

The message is added to the log file. If `log_file_only` is zero (the default), the message is also broadcast as a `KU$ STATUS_WIP` message through the `GET_STATUS` procedure to all users attached to the job.

The `LOG_ENTRY` procedure allows applications to tailor the log stream to match the abstractions provided by the application. For example, the command-line interface supports `INCLUDE` and `EXCLUDE` parameters defined by the user. Identifying these values as calls to the underlying `METADATA_FILTER` procedure would be confusing to users. Instead, the command-line interface can enter text into the log describing the settings for the `INCLUDE` and `EXCLUDE` parameters.

Lines entered in the log stream from `LOG_ENTRY` are prefixed by the string, " ; ; ; "

## METADATA\_FILTER Procedure

This procedure provides filters that allow you to restrict the items that are included in a job.

### Syntax

```
DBMS_DATAPUMP.METADATA_FILTER(
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN VARCHAR2,
  object_path IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_DATAPUMP.METADATA_FILTER(
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN CLOB,
  object_path IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 46–12 METADATA\_FILTER Procedure Parameters**

Parameter	Description
handle	The handle returned from the OPEN function
name	The name of the filter. See <a href="#">Table 46–13</a> for descriptions of the available filters.
value	The value of the filter
object_path	The object path to which the filter applies. If the default is used, the filter applies to all applicable objects. Lists of the object paths supported for each mode are contained in the catalog views for DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, and TABLE_EXPORT_OBJECTS. (Note that the TABLE_EXPORT_OBJECTS view is applicable to both Table and Tablespace mode because their object paths are the same.)  For an import operation, object paths reference the mode used to create the dump file rather than the mode being used for the import.

[Table 46–13](#) describes the name, the object type, and the meaning of the filters available with the METADATA\_FILTER procedure. The datatype for all the filters is a text expression. All operations support all filters.

**Table 46–13 Filters Provided by METADATA\_FILTER Procedure**

Name	Object Type	Meaning
NAME_EXPR	Named objects	Defines which object names are included in the job. You use the object type parameter to limit the filter to a particular object type.
NAME_LIST		For Table mode, identifies which tables are to be processed.

**Table 46–13 (Cont.) Filters Provided by METADATA\_FILTER Procedure**

Name	Object Type	Meaning
SCHEMA_EXPR SCHEMA_LIST	Schema objects	Restricts the job to objects whose owning schema name is satisfied by the expression.  For Table mode, only a single SCHEMA_EXPR filter is supported. If specified, it must only specify a single schema (for example, 'IN (' SCOTT ' )').  For Schema mode, identifies which users are to be processed.
TABLESPACE_EXPR TABLESPACE_LIST	TABLE, CLUSTER, INDEX, ROLLBACK_SEGMENT	Restricts the job to objects stored in a tablespace whose name is satisfied by the expression.  For Tablespace mode, identifies which tablespaces are to be processed. If a partition of an object is stored in the tablespace, the entire object is added to the job.  For Transportable mode, identifies which tablespaces are to be processed. If a table has a single partition in the tablespace set, all partitions must be in the tablespace set. An index is not included within the tablespace set unless all of its partitions are in the tablespace set. A domain index is not included in the tablespace set unless all of its secondary objects are included in the tablespace set.
INCLUDE_PATH_EXPR INCLUDE_PATH_LIST EXCLUDE_PATH_EXPR EXCLUDE_PATH_LIST	All	Defines which object paths are included in, or excluded from, the job. You use these filters to select only certain object types from the database or dump file set. Objects of paths satisfying the condition are included (INCLUDE_PATH_*) or excluded (EXCLUDE_PATH_*) from the operation. The object_path parameter is not supported for these filters.

## Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID\_ARGVAL. This exception can indicate any of the following conditions:
  - An object\_path was specified for an INCLUDE\_PATH\_EXPR or EXCLUDE\_PATH\_EXPR filter.
  - The specified object\_path is not supported for the current mode.
  - The SCHEMA\_EXPR filter specified multiple schemas for a Table mode job.
- INVALID\_STATE. The user called the METADATA\_FILTER procedure after the job left the defining state.
- INCONSISTENT\_ARGS. The filter value is of the wrong datatype or is missing.
- SUCCESS\_WITH\_INFO. The procedure succeeded but further information is available through the GET\_STATUS procedure.
- NO\_SUCH\_JOB. The specified job does not exist.

## Usage Notes

- Metadata filters identify a set of objects to be included or excluded from a Data Pump operation. Except for EXCLUDE\_PATH\_EXPR and INCLUDE\_PATH\_EXPR, dependent objects of an identified object will be processed along with the identified object. For example, if an index is identified for inclusion by a filter,

grants upon that index will also be included by the filter. Likewise, if a table is excluded by a filter, then indexes, constraints, grants and triggers upon the table will also be excluded by the filter.

- Two versions of each filter are supported: SQL expression and List. The SQL expression version of the filters offer maximum flexibility for identifying objects (for example the use of LIKE to support use of wild cards). The names of the expression filters are as follows:

- NAME\_EXPR
- SCHEMA\_EXPR
- TABLESPACE\_EXPR
- INCLUDE\_PATH\_EXPR
- EXCLUDE\_PATH\_EXPR

The list version of the filters allow maximum validation of the filter. An error will be reported if one of the elements in the filter is not found within the source database (for Export and network-based jobs) or is not found within the dump file (for file-based Import and SQLFILE jobs). The names of the list filters are as follows:

- NAME\_LIST
- SCHEMA\_LIST
- TABLESPACE\_LIST
- INCLUDE\_PATH\_LIST
- EXCLUDE\_PATH\_LIST

- Filters allow a user to restrict the items that are included in a job. For example, a user could request a full export, but without Package Specifications or Package Bodies.
- If multiple filters are specified for a object type, they are implicitly 'ANDed' together (that is, objects participating in the job must pass all of the filters applied to their object types).
- The same filter name can be specified multiple times within a job. For example, specifying NAME\_EXPR as '!='EMP'' and NAME\_EXPR as '!='DEPT'' on a Table mode export would produce a file set containing all of the tables except for EMP and DEPT.



## METADATA\_REMAP Procedure

This procedure specifies a remapping to be applied to objects as they are processed in the specified job.

### Syntax

```
DBMS_DATAPUMP.METADATA_REMAP (
  handle      IN NUMBER,
  name        IN VARCHAR2,
  old_value   IN VARCHAR2,
  value       IN VARCHAR2,
  object_type IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 46–14** METADATA\_REMAP Procedure Parameters

Parameter	Description
handle	The handle for the current job. The current session must have previously attached to the handle through a call to the OPEN function.
name	The name of the remap. See <a href="#">Table 46–15</a> for descriptions of the available remaps.
old_value	Specifies which value in the dump file set should be reset to value
value	The value of the parameter for the remap. This signifies the new value that old_value should be translated into.
object_type	Designates the object type to which the remap applies. The list of object types supported for each mode are contained in the DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, TABLE_EXPORT_OBJECTS, and TABLESPACE_EXPORT_OBJECTS catalog views.  By default, the remap applies to all applicable objects within the job. The object_type parameter allows a caller to specify different parameters for different object types within a job. Remaps that explicitly specify an object type override remaps that apply to all object types.

[Table 46–15](#) describes the remaps provided by the METADATA\_REMAP procedure.

**Table 46–15** Remaps Provided by the METADATA\_REMAP Procedure

Name	Datatype	Object Type	Meaning
REMAP_SCHEMA	Text	Schema objects	Any schema object in the job that matches the object_type parameter and was located in the old_value schema will be moved to the value schema.  Privileged users can perform unrestricted schema remaps.  Nonprivileged users can perform schema remaps only if their schema is the target schema of the remap.  For example, SCOTT can remap his BLAKE 's objects to SCOTT, but SCOTT cannot remap SCOTT 's objects to BLAKE.

**Table 46–15 (Cont.) Remaps Provided by the METADATA\_REMAP Procedure**

Name	Datatype	Object Type	Meaning
REMAP_ TABLESPACE	Text	TABLE, INDEX, ROLLBACK_ SEGMENT, MATERIALIZED_ VIEW, MATERIALIZED_ VIEW_ LOG, TABLE_ SPACE	Any storage segment in the job that matches the <code>object_type</code> parameter and was located in the <code>old_value</code> tablespace will be relocated to the <code>value</code> tablespace.
REMAP_ DATAFILE	Text	LIBRARY, TABLESPACE, DIRECTORY	Any datafile reference in the job that matches the <code>object_type</code> parameter and referenced the <code>old_value</code> datafile will be redefined to use the <code>value</code> datafile.
REMAP_TABLE	Text	TABLE	Any reference to a table in the job that matches the <code>old_value</code> table name will be replaced with the <code>value</code> table name. The <code>old_value</code> parameter may refer to a partition such as <code>employees.low</code> . This allows names for tables constructed the by <code>PARTITION_OPTIONS=DEPARTITION</code> parameter to be specified by the user.

## Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_ARGVAL`. This message can indicate any of the following:
  - The job's mode does not include the specified `object_type`.
  - The remap has already been specified for the specified `old_value` and `object_type`.
- `INVALID_OPERATION`. Remaps are only supported for `SQL_FILE` and Import operations. The job's operation was Export, which does not support the use of metadata remaps.
- `INVALID_STATE`. The user called `METADATA_REMAP` after the job had started (that is, the job was not in the defining state).
- `INCONSISTENT_ARGS`. There was no `value` supplied or it was of the wrong datatype for the remap.
- `PRIVILEGE_ERROR`. A nonprivileged user attempted to do a `REMAP_SCHEMA` to a different user's schema or a `REMAP_DATAFILE`.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

## Usage Notes

- The `METADATA_REMAP` procedure is only supported for Import and `SQL_FILE` operations. It enables you to apply commonly desired, predefined remappings to the definition of objects as part of the transfer. If you need remaps that are not supported within this procedure, you should do a preliminary `SQL_FILE` operation to produce a SQL script corresponding to the dump file set. By editing

the DDL directly and then executing it, you can produce any remappings that you need.

- Transforms for the DataPump API are a subset of the remaps implemented by the `DBMS_METADATA.SET_TRANSFORM_PARAMETER` API. Multiple remaps can be defined for a single job. However, each remap defined must be unique according to its parameters. That is, two remaps cannot specify conflicting or redundant remaps.

## METADATA\_TRANSFORM Procedure

This procedure specifies transformations to be applied to objects as they are processed in the specified job.

### Syntax

```
DBMS_DATAPUMP.METADATA_TRANSFORM (
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN VARCHAR2,
  object_type IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_DATAPUMP.METADATA_TRANSFORM (
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN NUMBER,
  object_type IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 46–16 METADATA\_TRANSFORM Procedure Parameters**

Parameter	Description
handle	The handle for the current job. The current session must have previously attached to the handle through a call to the OPEN function.
name	The name of the transformation. See <a href="#">Table 46–17</a> for descriptions of the available transforms.
value	The value of the parameter for the transform
object_type	Designates the object type to which the transform applies. The list of object types supported for each mode are contained in the DATABASE_EXPORT_OBJECTS, SCHEMA_EXPORT_OBJECTS, TABLE_EXPORT_OBJECTS, and TABLESPACE_EXPORT_OBJECTS catalog views.  By default, the transform applies to all applicable objects within the job. The object_type parameter allows a caller to specify different transform parameters for different object types within a job. Transforms that explicitly specify an object type override transforms that apply to all object types.

[Table 46–17](#) describes the transforms provided by the METADATA\_TRANSFORM procedure.

**Table 46–17 Transforms Provided by the METADATA\_TRANSFORM Procedure**

Name	Datatype	Object Type	Meaning
PCTSPACE	NUMBER	TABLE INDEX TABLESPACE	Specifies a percentage multiplier used to alter extent allocations and datafile sizes. Used to shrink large tablespaces for testing purposes.  Defaults to 100.
SEGMENT_ATTRIBUTES	NUMBER	TABLE, INDEX	If nonzero (TRUE), emit storage segment parameters.  Defaults to 1.

**Table 46–17 (Cont.) Transforms Provided by the METADATA\_TRANSFORM Procedure**

Name	Datatype	Object Type	Meaning
STORAGE	NUMBER	TABLE	<p>If nonzero (TRUE), emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is zero.)</p> <p>Defaults to nonzero (TRUE).</p>
OID	NUMBER	TYPE TABLE	<p>If zero, inhibits the assignment of the exported OID during type or table creation. Instead, a new OID will be assigned.</p> <p>Use of this transform on Object Tables will cause breakage in REF columns that point to the table.</p> <p>Defaults to 1.</p>
SEGMENT_CREATION	NUMBER	TABLE	<p>If nonzero (TRUE), the SQL SEGMENT CREATION clause is added to the CREATE TABLE statement. That is, the CREATE TABLE statement will explicitly say either SEGMENT CREATION DEFERRED or SEGMENT CREATION IMMEDIATE.</p> <p>If the value is FALSE, then the SEGMENT CREATION clause is omitted from the CREATE TABLE statement. Set this parameter to FALSE to use the default segment creation attributes for the table(s) being loaded.</p> <p>Defaults to nonzero (TRUE).</p> <p>(This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).)</p>

## Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID\_ARGVAL. This message can indicate any of the following:
  - The mode is transportable, which doesn't support transforms.
  - The job's mode does not include the specified object\_type.
  - The transform has already been specified for the specified value and object\_type.
- INVALID\_OPERATION. Transforms are only supported for SQL\_FILE and Import operations. The job's operation was Export which does not support the use of metadata transforms.
- INVALID\_STATE. The user called METADATA\_TRANSFORM after the job had started (that is, the job was not in the defining state).
- INCONSISTENT\_ARGS. There was no value supplied or it was of the wrong datatype for the transform.
- PRIVILEGE\_ERROR. A nonprivileged user attempted to do a REMAP\_SCHEMA to a different user's schema or a REMAP\_DATAFILE.
- SUCCESS\_WITH\_INFO. The procedure succeeded, but further information is available through the GET\_STATUS procedure.

- NO\_SUCH\_JOB. The specified job does not exist.

### Usage Notes

- The METADATA\_TRANSFORM procedure is only supported for Import and SQL\_FILE operations. It enables you to apply commonly desired, predefined transformations to the definition of objects as part of the transfer. If you need transforms that are not supported within this procedure, you should do a preliminary SQL\_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any transformations that you need.
- Transforms for the DataPump API are a subset of the transforms implemented by the DBMS\_METADATA.SET\_TRANSFORM\_PARAMETER API. Multiple transforms can be defined for a single job. However, each transform defined must be unique according its parameters. That is, two transforms cannot specify conflicting or redundant transformations.

## OPEN Function

This function is used to declare a new job using the Data Pump API. The handle that is returned is used as a parameter for calls to all other procedures (but not to the `ATTACH` function).

### Syntax

```
DBMS_DATAPUMP.OPEN (
  operation      IN VARCHAR2,
  job_mode       IN VARCHAR2,
  remote_link    IN VARCHAR2 DEFAULT NULL,
  job_name       IN VARCHAR2 DEFAULT NULL,
  version        IN VARCHAR2 DEFAULT 'COMPATIBLE'
)
RETURN NUMBER;
```

### Parameters

**Table 46–18** OPEN Function Parameters

Parameter	Meaning
<code>operation</code>	The type of operation to be performed. <a href="#">Table 46–19</a> contains descriptions of valid operation types.
<code>job_mode</code>	The scope of the operation to be performed. <a href="#">Table 46–20</a> contains descriptions of valid modes. Specifying <code>NULL</code> generates an error.
<code>remote_link</code>	If the value of this parameter is non-null, it provides the name of a database link to the remote database that will be the source of data and metadata for the current job.
<code>job_name</code>	<p>The name of the job. The name is limited to 30 characters; it will be truncated if more than 30 characters are used. It may consist of printable characters and spaces. It is implicitly qualified by the schema of the user executing the <code>OPEN</code> function and must be unique to that schema (that is, there cannot be other Data Pump jobs using the same name).</p> <p>The name is used to identify the job both within the API and with other database components such as identifying the job in the <code>DBA_RESUMABLE</code> view if the job becomes suspended through lack of resources. If no name is supplied, a system generated name will be provided for the job in the following format:  <code>"SYS_&lt;OPERATION&gt;_&lt;MODE&gt;_%N"</code>.</p> <p>The default job name is formed where <code>%N</code> expands to a two-digit incrementing integer starting at '01' (for example, <code>"SYS_IMPORT_FULL_03"</code>). The name supplied for the job will also be used to name the master table and other resources associated with the job.</p>
<code>version</code>	<p>The version of database objects to be extracted. This option is only valid for Export, network Import, and <code>SQL_FILE</code> operations. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows:</p> <ul style="list-style-type: none"> <li>▪ <code>COMPATIBLE</code> - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the <code>V\$COMPATIBILITY</code> view). Database compatibility must be set to 9.2 or higher.</li> <li>▪ <code>LATEST</code> - the version of the metadata corresponds to the database version.</li> <li>▪ A specific database version, for example, '10.0.0'. In Oracle Database10g, this value cannot be lower than 10.0.0.</li> </ul>

Table 46–19 describes the valid operation types for the OPEN function.

**Table 46–19 Valid Operation Types for the OPEN Function**

Operation	Description
EXPORT	Saves data and metadata to a dump file set or obtains an estimate of the size of the data for an operation.
IMPORT	Restores data and metadata from a dump file set or across a database link.
SQL_FILE	Displays the metadata within a dump file set, or from across a network link, as a SQL script. The location of the SQL script is specified through the ADD_FILE procedure.

Table 46–20 describes the valid modes for the OPEN function.

**Table 46–20 Valid Modes for the OPEN Function**

Mode	Description
FULL	Operates on the full database or full dump file set except for the SYS, XDB, ORDSYS, MDSYS, CTXSYS, ORDPLUGINS, and LBACSYS schemas.
SCHEMA	Operates on a set of selected schemas. Defaults to the schema of the current user. All objects in the selected schemas are processed. Users cannot specify SYS, XDB, ORDSYS, MDSYS, CTXSYS, ORDPLUGINS, or LBACSYS schemas for this mode.
TABLE	Operates on a set of selected tables. Defaults to all of the tables in the current user's schema. Only tables and their dependent objects are processed.
TABLESPACE	Operates on a set of selected tablespaces. No defaulting is performed. Tables that have storage in the specified tablespaces are processed in the same manner as in Table mode.
TRANSPORTABLE	Operates on metadata for tables (and their dependent objects) within a set of selected tablespaces to perform a transportable tablespace export/import.

## Return Values

- An opaque handle for the job. This handle is used as input to the following procedures: ADD\_FILE, CREATE\_JOB\_VIEW, DATA\_FILTER, DETACH, GET\_STATUS, LOG\_ENTRY, LOG\_ERROR, METADATA\_FILTER, METADATA\_REMAP, METADATA\_TRANSFORM, SET\_PARALLEL, SET\_PARAMETER, START\_JOB, STOP\_JOB, and WAIT\_FOR\_JOB

## Exceptions

- INVALID\_ARGVAL. An invalid operation or mode was specified. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- JOB\_EXISTS. A table already exists with the specified job name.
- PRIVILEGE\_ERROR. The user does not have the necessary privileges or roles to use the specified mode.
- INTERNAL\_ERROR. The job was created under the wrong schema or the master table was of the wrong format.



- `SUCCESS_WITH_INFO`. The function succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

### Usage Notes

- When the job is created, a master table is created for the job under the caller's schema within the caller's default tablespace. A handle referencing the job is returned that attaches the current session to the job. Once attached, the handle remains valid until either an explicit or implicit detach occurs. The handle is only valid in the caller's session. Other handles can be attached to the same job from a different session by using the `ATTACH` function.
- If the call to the `OPEN` function fails, call the `GET_STATUS` procedure with a null handle to retrieve additional information about the failure.

## SET\_PARALLEL Procedure

This procedure adjusts the degree of parallelism within a job.

### Syntax

```
DBMS_DATAPUMP.SET_PARALLEL(
  handle      IN NUMBER,
  degree      IN NUMBER);
```

### Parameters

**Table 46–21** SET\_PARALLEL Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.
degree	The maximum number of worker processes that can be used for the job. You use this parameter to adjust the amount of resources used for a job.

### Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID\_OPERATION. The SET\_PARALLEL procedure is only valid for export and import operations.
- INVALID\_ARGVAL. An invalid value was supplied for an input parameter.
- SUCCESS\_WITH\_INFO. The procedure succeeded, but further information is available through the GET\_STATUS procedure.
- NO\_SUCH\_JOB. The specified job does not exist.

### Usage Notes

- The SET\_PARALLEL procedure is only available in the Enterprise Edition of the Oracle database.
- The SET\_PARALLEL procedure can be executed by any session attached to a job. The job must be in one of the following states: Defining, Idling, or Executing.
- The effect of decreasing the degree of parallelism may be delayed because ongoing work needs to find an orderly completion point before SET\_PARALLEL can take effect.
- Decreasing the parallelism will not result in fewer worker processes associated with the job. It will only decrease the number of worker processes that will be executing at any given time.
- Increasing the parallelism will take effect immediately if there is work that can be performed in parallel.
- The degree of parallelism requested by a user may be decreased based upon settings in the resource manager or through limitations introduced by the PROCESSES or SESSIONS initialization parameters in the init.ora file.
- To parallelize an Export job to a degree of *n*, the user should supply *n* files in the dump file set or specify a substitution variable in a file specification. Otherwise, some of the worker processes will be idle while waiting for files.

- SQL\_FILE operations always operate with a degree of 1. Jobs running in the Transportable mode always operate with a degree of 1.

## SET\_PARAMETER Procedures

This procedure is used to specify job-processing options.

### Syntax

```
DBMS_DATAPUMP.SET_PARAMETER(
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN VARCHAR2);
```

```
DBMS_DATAPUMP.SET_PARAMETER (
  handle      IN NUMBER,
  name        IN VARCHAR2,
  value       IN NUMBER);
```

### Parameters

**Table 46–22 SET\_PARAMETER Procedure Parameters**

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to the OPEN function.
name	The name of the parameter. <a href="#">Table 46–23</a> describes the valid parameter names.
value	The value for the specified parameter

[Table 46–23](#) describes the valid options for the name parameter of the SET\_PARAMETER procedure.

**Table 46–23 Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
CLIENT_COMMAND	Text	All	An opaque string used to describe the current operation from the client's perspective. The command-line procedures will use this string to store the original command used to invoke the job.
COMPRESSION	Text	Export	<p>Allows you to trade off the size of the dump file set versus the time it takes to perform export and import operations.</p> <p>The DATA_ONLY option compresses only user data in the dump file set.</p> <p>The METADATA_ONLY option compresses only metadata in the dump file set.</p> <p>The ALL option compresses both user data and metadata.</p> <p>The NONE option stores the dump file set in an uncompressed format.</p> <p>The METADATA_ONLY and NONE options require a job version of 10.2 or later. All other options require a job version of 11.1 or later.</p> <p>Default=METADATA_ONLY</p>
DATA_OPTIONS	Number	Export and Import	<p>A bitmask to supply special options for processing the job. The possible values are as follows:</p> <ul style="list-style-type: none"> <li>■ KU\$_DATAOPT_SKIP_CONST_ERR</li> <li>■ KU\$_DATAOPT_XMLTYPE_CLOB</li> <li>■ KU\$_DATAOPT_DISABL_APPEND_HINT</li> </ul> <p>Export supports the value KU\$_DATAOPT_XMLTYPE_CLOB. This option stores compressed XMLType columns in the dump file as CLOBs rather than as XML documents.</p> <p>Import supports the value KU\$_DATAOPT_SKIP_CONST_ERR. This option specifies that if constraint violations occur while data is being imported into user tables, the rows that cause the violations will be rejected and the load will continue. If this option is not set, a constraint error will abort the loading of the entire partition (or table for unpartitioned tables). Setting this option may affect performance, especially for pre-existing tables with unique indexes or constraints.</p> <p>Import also supports the value KU\$_DATAOPT_DISABL_APPEND_HINT. This option prevents the append hint from being applied to the data load. Disabling the APPEND hint can be useful if there is a small set of data objects to load that already exist in the database and some other application may be concurrently accessing one or more of the data objects.</p> <p>Use of this parameter requires that the version on the OPEN function be set to 11.1 or later.</p> <p>Default=0</p>

**Table 46–23 (Cont.) Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
ENCRYPTION	Text	Export	<p>Specifies what to encrypt in the dump file set, as follows:</p> <p>ALL enables encryption for all data and metadata in the export operation.</p> <p>DATA_ONLY specifies that only data is written to the dump file set in encrypted format.</p> <p>ENCRYPTED_COLUMNS_ONLY specifies that only encrypted columns are written to the dump file set in encrypted format.</p> <p>METADATA_ONLY specifies that only metadata is written to the dump file set in encrypted format.</p> <p>NONE specifies that no data is written to the dump file set in encrypted format.</p> <p>This parameter requires a job version of 11.1 or later.</p> <p>The default value depends upon the combination of encryption-related parameters that are used. To enable encryption, either ENCRYPTION or ENCRYPTION_PASSWORD or both, must be specified. If only ENCRYPTION_PASSWORD is specified, then ENCRYPTION defaults to ALL. If neither ENCRYPTION nor ENCRYPTION_PASSWORD is specified, then ENCRYPTION defaults to NONE.</p> <p>To specify ALL, DATA_ONLY, or METADATA_ONLY, the COMPATIBLE initialization parameter must be set to at least 11.1.</p> <p><b>NOTE:</b> If the data being exported includes SecureFiles that you want to be encrypted, then you must specify ENCRYPTION=ALL to encrypt the entire dump file set. Encryption of the entire dump file set is the only way to achieve encryption security for SecureFiles during a Data Pump export operation.</p>
ENCRYPTION_ALGORITHM	Text	Export	<p>Identifies which cryptographic algorithm should be used to perform encryption. Possible values are AES128, AES192, and AES256.</p> <p>The ENCRYPTION_ALGORITHM parameter requires that you also specify either ENCRYPTION or ENCRYPTION_PASSWORD; otherwise an error is returned. See <i>Oracle Database Advanced Security Administrator's Guide</i> for information about encryption algorithms.</p> <p>This parameter requires a job version of 11.1 or later.</p> <p>Default=AES128</p>

**Table 46–23 (Cont.) Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
ENCRYPTION_MODE	Text	Export	<p>Identifies the types of security used for encryption and decryption. The values are as follows:</p> <p><b>PASSWORD</b> requires that you provide a password when creating encrypted dump file sets. You will need to provide the same password when you import the dump file set. <b>PASSWORD</b> mode requires that you also specify the <code>ENCRYPTION_PASSWORD</code> parameter. The <b>PASSWORD</b> mode is best suited for cases in which the dump file set will be imported into a different or remote database, but which must remain secure in transit.</p> <p><b>TRANSPARENT</b> allows an encrypted dump file set to be created without any intervention from a database administrator (DBA), provided the required Oracle Encryption Wallet is available. Therefore, the <code>ENCRYPTION_PASSWORD</code> parameter is not required, and will in fact, cause an error if it is used in <b>TRANSPARENT</b> mode. This encryption mode is best suited for cases in which the dump file set will be imported into the same database from which it was exported.</p> <p><b>DUAL</b> creates a dump file set that can later be imported using either the Oracle Encryption Wallet or the password that was specified with the <code>ENCRYPTION_PASSWORD</code> parameter. <b>DUAL</b> mode is best suited for cases in which the dump file set will be imported on-site using the Oracle Encryption Wallet, but which may also need to be imported offsite where the Oracle Encryption Wallet is not available.</p> <p>When you use the <code>ENCRYPTION_MODE</code> parameter, you must also use either the <code>ENCRYPTION</code> or <code>ENCRYPTION_PASSWORD</code> parameter. Otherwise, an error is returned.</p> <p>To use <b>DUAL</b> or <b>TRANSPARENT</b> mode, the <code>COMPATIBLE</code> initialization parameter must be set to at least 11.1.</p> <p>The default mode depends on which other encryption-related parameters are used. If only <code>ENCRYPTION</code> is specified, then the default mode is <b>TRANSPARENT</b>. If <code>ENCRYPTION_PASSWORD</code> is specified and the Oracle Encryption Wallet is open, then the default is <b>DUAL</b>. If <code>ENCRYPTION_PASSWORD</code> is specified and the Oracle Encryption Wallet is closed, then the default is <b>PASSWORD</b>.</p>

**Table 46–23 (Cont.) Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
ENCRYPTION_PASSWORD	Text	Export and Import	<p>Specifies a key for re-encrypting encrypted table columns, metadata, or table data so that they are not written as clear text in the dump file set. If the export operation involves encrypted table columns, but an encryption password is not supplied, then the encrypted columns will be written to the dump file set as clear text and a warning will be issued.</p> <p><b>NOTE:</b> Data Pump encryption functionality has changed as of Oracle Database 11g release 1 (11.1). Prior to release 11.1, the ENCRYPTION_PASSWORD parameter applied only to encrypted columns. However, as of release 11.1, the new ENCRYPTION parameter provides options for encrypting other types of data. This means that if you now specify ENCRYPTION_PASSWORD without also specifying ENCRYPTION and a specific option, then <i>all</i> data written to the dump file will be encrypted (equivalent to specifying ENCRYPTION=ALL). If you want to re-encrypt <i>only</i> encrypted columns, you must now specify ENCRYPTION=ENCRYPTED_COLUMNS_ONLY in addition to ENCRYPTION_PASSWORD.</p> <p>For export operations, this parameter is required if ENCRYPTION_MODE is set to either PASSWORD or DUAL.</p> <p>If ENCRYPTION_PASSWORD is specified but ENCRYPTION_MODE is not specified, then it is not necessary to have Transparent Data Encryption set up since ENCRYPTION_MODE will default to PASSWORD.</p> <p>The ENCRYPTION_PASSWORD parameter is not valid if the requested encryption mode is TRANSPARENT.</p> <p>To use the ENCRYPTION_PASSWORD parameter if ENCRYPTION_MODE is set to DUAL, you must have Transparent Data Encryption set up. See <i>Oracle Database Advanced Security Administrator's Guide</i> for more information about Transparent Data Encryption.</p> <p>For network exports, the ENCRYPTION_PASSWORD parameter in conjunction with ENCRYPTION=ENCRYPTED_COLUMNS_ONLY is not supported with user-defined external tables that have encrypted columns. The table will be skipped and an error message will be displayed, but the job will continue.</p> <p>Encryption attributes for all columns must match between the exported table definition and the target table.</p> <p>This parameter requires a job version of 10.2 or later.</p>
ESTIMATE	Text	Export and Import	<p>Specifies that the estimate method for the size of the tables should be performed before starting the job.</p> <p>If BLOCKS, a size estimate for the user tables is calculated using the count of blocks allocated to the user tables.</p> <p>If STATISTICS, a size estimate for the user tables is calculated using the statistics associated with each table. If no statistics are available for a table, the size of the table is estimated using BLOCKS.</p> <p>The ESTIMATE parameter cannot be used in Transportable Tablespace mode.</p> <p>Default=BLOCKS</p>



**Table 46–23 (Cont.) Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
ESTIMATE_ONLY	Number	Export	Specifies that only the estimation portion of an export job should be performed. This option is useful for estimating the size of dump files when the size of the export is unknown.
FLASHBACK_SCN	NUMBER	Export and network Import	System change number (SCN) to serve as transactionally consistent point for reading user data. If neither FLASHBACK_SCN nor FLASHBACK_TIME is specified, there will be no transactional consistency between partitions, except for logical standby databases and Streams targets. FLASHBACK_SCN is not supported in Transportable mode.
FLASHBACK_TIME	Text	Export and network Import	<p>Either the date and time used to determine a consistent point for reading user data or a string of the form TO_TIMESTAMP ( . . . ).</p> <p>If neither FLASHBACK_SCN nor FLASHBACK_TIME is specified, there will be no transactional consistency between partitions.</p> <p>FLASHBACK_SCN and FLASHBACK_TIME cannot both be specified for the same job. FLASHBACK_TIME is not supported in Transportable mode.</p>
INCLUDE_METADATA	NUMBER	Export and Import	<p>If nonzero, metadata for objects will be moved in addition to user table data.</p> <p>If zero, metadata for objects will not moved. This parameter converts an Export operation into an unload of user data and an Import operation into a load of user data.</p> <p>INCLUDE_METADATA is not supported in Transportable mode.</p> <p>Default=1</p>
PARTITION_OPTIONS	Text	Import	<p>Specifies how partitioned tables should be handled during an import operation. The options are as follows:</p> <p>NONE means that partitioning is reproduced on the target database as it existed in the source database.</p> <p>DEPARTITION means that each partition or subpartition that contains storage in the job is reproduced as a separate unpartitioned table. Intermediate partitions that are subpartitioned are not re-created (although their subpartitions are converted into tables). The names of the resulting tables are system-generated from the original table names and partition names unless the name is overridden by the REMAP_TABLE metadata transform.</p> <p>MERGE means that each partitioned table is re-created in the target database as an unpartitioned table. The data from all of the source partitions is merged into a single storage segment. This option is not supported for transportable jobs or when the TRANSPORTABLE parameter is set to ALWAYS.</p> <p>This parameter requires a job version of 11.1 or later.</p> <p>Default=NONE</p>
SKIP_UNUSABLE_INDEXES	NUMBER	Import	<p>If nonzero, rows will be inserted into tables having unusable indexes. SKIP_UNUSABLE_INDEXES is not supported in Transportable mode.</p> <p>Default=1</p>

**Table 46–23 (Cont.) Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
SOURCE_EDI TION	Text	Export and network Import	The application edition that will be used for determining the objects that will be unloaded for export and for network import.
TABLE_EXISTS_ ACTION	Text	Import	<p>Specifies the action to be performed when data is loaded into a preexisting table. The possible actions are: TRUNCATE, REPLACE, APPEND, and SKIP.</p> <p>If INCLUDE_METADATA=0, only TRUNCATE and APPEND are supported.</p> <p>If TRUNCATE, rows are removed from a preexisting table before inserting rows from the Import.</p> <p>Note that if TRUNCATE is specified on tables referenced by foreign key constraints, the TRUNCATE will be modified into a REPLACE.</p> <p>If REPLACE, preexisting tables are replaced with new definitions. Before creating the new table, the old table is dropped.</p> <p>If APPEND, new rows are added to the existing rows in the table.</p> <p>If SKIP, the preexisting table is left unchanged.</p> <p>TABLE_EXISTS_ACTION is not supported in Transportable mode.</p> <p>The default is SKIP if metadata is included in the import. The default is APPEND if INCLUDE_METADATA is set to 0.</p>
TABLESPACE_ DATAFILE	Text	Import	<p>Specifies the full file specification for a datafile in the transportable tablespace set. TABLESPACE_DATAFILE is only valid for transportable mode imports.</p> <p>TABLESPACE_DATAFILE can be specified multiple times, but the value specified for each occurrence must be different.</p>
TARGET_EDI TION	Text	Import	The application edition that will be used for determining where the objects will be loaded for import and for network import.

**Table 46–23 (Cont.) Valid Options for the name Parameter in the SET\_PARAMETER Procedure**

Parameter Name	Datatype	Supported Operations	Meaning
TRANSPORTABLE	Text	Export	<p>For export operations done in table mode, allows the data to be moved using transportable tablespaces. Storage segments in the moved tablespaces that are not associated with the parent schemas (tables) will be reclaimed at import time. If individual partitions are selected in a table-mode job, only the tablespaces referenced by those partitions will be moved. During import, the moved partitions can only be reconstituted as tables by using the PARTITION_OPTIONS=DEPARTITION parameter.</p> <p>Use of the TRANSPORTABLE parameter prohibits the subsequent import of the dump file into a database at a lower version or using different character sets. Additionally, the data files may need to be converted if the target database is on a different platform. The TRANSPORTABLE parameter is not allowed if a network link is supplied on the OPEN call.</p> <p>The possible values for this parameter are as follows:  ALWAYS - data is always moved by moving data files  NEVER - data files are never used for copying user data</p> <p>This parameter requires a job version of 11.1 or later  Default=NEVER</p>
TTS_FULL_CHECK	NUMBER	Export	<p>If nonzero, verifies that a transportable tablespace set has no dependencies (specifically, IN pointers) on objects outside the set, and vice versa. Only valid for Transportable mode Exports.</p> <p>Default=0</p>
USER_METADATA	NUMBER	Export and network Import	<p>For schema-mode operations, specifies that the metadata to re-create the users' schemas (for example, privilege grants to the exported schemas) should also be part of the operation if set to nonzero. Users must be privileged to explicitly set this parameter.</p> <p>The USER_METADATA parameter cannot be used in Table, Tablespace, or Transportable Tablespace mode.</p> <p>Default=1 if user has DATAPUMP_EXP_FULL_DATABASE role; 0 otherwise.</p>

## Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID\_ARGVAL. This exception could be due to any of the following causes:
  - An invalid name was supplied for an input parameter
  - The wrong datatype was used for value
  - A value was not supplied
  - The supplied value was not allowed for the specified parameter name
  - A flashback parameter had been established after a different flashback parameter had already been established
  - A parameter was specified that did not support duplicate definitions
- INVALID\_OPERATION. The operation specified is invalid in this context.

- `INVALID_STATE`. The specified job is not in the Defining state.
- `INCONSISTENT_ARGS`. Either the specified parameter is not supported for the current operation type or it is not supported for the current mode.
- `PRIVILEGE_ERROR`. The user does not have the `DATAPUMP_EXP_FULL_DATABASE` or `DATAPUMP_IMP_FULL_DATABASE` role required for the specified parameter.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

### Usage Notes

- The `SET_PARAMETER` procedure is used to specify optional features for the current job. See [Table 46-23](#) for a list of supported options.

## START\_JOB Procedure

This procedure begins or resumes execution of a job.

### Syntax

```
DBMS_DATAPUMP.START_JOB (
  handle          IN NUMBER,
  skip_current    IN NUMBER DEFAULT 0,
  abort_step      IN NUMBER DEFAULT 0,
  cluster_ok      IN NUMBER DEFAULT 1,
  service_name    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 46–24** START\_JOB Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.
skip_current	If nonzero, causes actions that were 'in progress' on a previous execution of the job to be skipped when the job restarts. The skip will only be honored for Import jobs. This mechanism allows the user to skip actions that trigger fatal bugs and cause the premature termination of a job. Multiple actions can be skipped on a restart. The log file will identify which actions are skipped. If a domain index was being processed, all pieces of the domain index are skipped even if the error occurred in only a subcomponent of the domain index.  A description of the actions skipped is entered into the log file. skip_current is ignored for the initial START_JOB in a job.  If zero, no data or metadata is lost upon a restart.
abort_step	Value must be 0. Inserting values other than 0 into this argument will have unintended consequences.
cluster_ok	If = 0, all workers are started on the current instance. Otherwise, workers are started on instances usable by the job.
service_name	If specified, indicates a service name used to constrain the job to specific instances or to a specific resource group.

### Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID\_STATE. The causes of this exception can be any of the following:
  - No files have been defined for an Export, non-network Import, or SQL\_FILE job
  - An ADD\_FILE procedure has not been called to define the output for a SQL\_FILE job
  - A TABLESPACE\_DATAFILE parameter has not been defined for a Transportable Import job
  - A TABLESPACE\_EXPR metadata filter has not been defined for a Transportable or Tablespace mode Export or Network job
  - The dump file set on an Import or SQL\_FILE job was either incomplete or missing a master table specification

- `INVALID_OPERATION`. Unable to restore master table from a dump file set.
- `INTERNAL_ERROR`. An inconsistency was detected when the job was started. Additional information may be available through the `GET_STATUS` procedure.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

### Usage Notes

- When this procedure is called to request that the corresponding job be started or restarted, the state of the job is changed from either the Defining or Idling state to the Executing state.
- If the `SET_PARALLEL` procedure was not called prior to the `START_JOB` procedure, the initial level of parallelism used in the job will be 1. If `SET_PARALLEL` was called prior to the job starting, the `degree` specified by the last `SET_PARALLEL` call determines the parallelism for the job. On restarts, the parallelism is determined by the previous parallel setting for the job, unless it is overridden by another `SET_PARALLEL` call.
- To restart a stopped job, an `ATTACH` function must be performed prior to executing the `START_JOB` procedure.

## STOP\_JOB Procedure

This procedure terminates a job, but optionally, preserves the state of the job.

### Syntax

```
DBMS_DATAPUMP.STOP_JOB (
  handle      IN NUMBER,
  immediate   IN NUMBER DEFAULT 0,
  keep_master IN NUMBER DEFAULT NULL,
  delay       IN NUMBER DEFAULT 60);
```

### Parameters

**Table 46–25 STOP\_JOB Procedure Parameters**

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function. At the end of the procedure, the user is detached from the handle.
immediate	If nonzero, the worker processes are aborted immediately. This halts the job quickly, but parts of the job will have to be rerun if the job is ever restarted.  If zero, the worker processes are allowed to complete their current work item (either metadata or table data) before they are terminated. The job is placed in a Stop Pending state while the workers finish their current work.
keep_master	If nonzero, the master table is retained when the job is stopped. If zero, the master table is dropped when the job is stopped. If the master table is dropped, the job will not be restartable. If the master table is dropped during an export job, the created dump files are deleted.
delay	The number of seconds to wait until other attached sessions are forcibly detached. The delay allows other sessions attached to the job to be notified that a stop has been performed. The job keeps running until either all clients have detached or the delay has been satisfied. If no delay is specified, then the default delay is 60 seconds. If a shorter delay is used, clients might not be able to retrieve the final messages for the job through the GET_STATUS procedure.

### Exceptions

- INVALID\_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID\_STATE. The job is already in the process of being stopped or completed.
- SUCCESS\_WITH\_INFO. The procedure succeeded, but further information is available through the GET\_STATUS procedure.
- NO\_SUCH\_JOB. The specified job does not exist.

### Usage Notes

- This procedure is used to request that the corresponding job stop executing.
- The termination of a job that is in an Executing state may take several minutes to complete in an orderly fashion.
- For jobs in the Defining, Idling, or Completing states, this procedure is functionally equivalent to the DETACH procedure.

- Once a job is stopped, it can be restarted using the `ATTACH` function and `START_JOB` procedures, provided the master table and the dump file set are left intact.
- If the `KEEP_MASTER` parameter is not specified, and the job is in the Defining state or has a mode of Transportable, the master table is dropped. Otherwise, the master table is retained.



## WAIT\_FOR\_JOB Procedure

This procedure runs a job until it either completes normally or stops for some other reason.

### Syntax

```
DBMS_DATAPUMP.WAIT_FOR_JOB (
  handle      IN  NUMBER,
  job_state   OUT VARCHAR2);
```

### Parameters

**Table 46–26** *WAIT\_FOR\_JOB Procedure Parameters*

Parameter	Description
handle	The handle of the job. The current session must have previously attached to the handle through a call to either the <code>OPEN</code> or <code>ATTACH</code> function. At the end of the procedure, the user is detached from the handle.
job_state	The state of the job when it has stopped executing. This will be either <code>Stopped</code> or <code>Completed</code> .

### Exceptions

- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` API.
- `INVALID_HANDLE`. The job handle is no longer valid.

### Usage Notes

This procedure provides the simplest mechanism for waiting for the completion of a Data Pump job. The job should be started before calling `WAIT_FOR_JOB`. When `WAIT_FOR_JOB` returns, the job will no longer be executing. If the job completed normally, the final status will be `Completed`. If the job stopped executing because of a `STOP_JOB` request or an internal error, the final status will be `Stopped`.



---

---

## DBMS\_DBFS\_CONTENT

The DBMS\_DBFS\_CONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.

**See Also:**

- *Oracle Database SecureFiles and Large Objects Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_DBFS\\_CONTENT](#)
  - Overview
  - Security Model
  - Constants
  - Exceptions
  - Operational Notes
- [Data Structures](#)
- [Summary of DBMS\\_DBFS\\_CONTENT Subprograms](#)

---

## Using DBMS\_DBFS\_CONTENT

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Exceptions](#)
- [Operational Notes](#)

## Overview

The DBMS\_DBFS\_CONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers. The "Content" in the DBFS Content interface refers to a file including metadata, and it can map to a BLOB (and other columns) in a table or be dynamically created by user-written plug-ins in Java or PL/SQL that run inside the database. This latter form is referred to as a "Store Provider."

For applications that already use LOBs as columns in their schema, the DBFS Content interface comes with a default implementation to access the BLOB columns. This enables existing applications to easily add PL/SQL provider implementations and provide access through the DBFS Content interface without rewriting their schema or their business logic. Additionally, applications can read and write content that is stored in other (third party) stores through the standard DBFS Content programming interface.

Examples of providers include:

- Content applications like Content DB
- Packaged applications that want to surface data through file
- Custom applications that want to leverage the file system interface -- for example, an application that stores medical image

The DBS\_DBFS\_CONTENT package abstracts out the common features of various stores into a simple and minimalist interface used to build portable client applications while insulated from store-specific libraries and implementation.

The content interface aggregates the path namespace of one or more stores into a single unified namespace, using the first component of the path name as a disambiguator, and presents this namespace to client-applications.

This allows clients to access the underlying documents using either a full-absolute path name as a single string:

```
/store-name/store-specific-path-name
```

or a store-qualified path name as a string 2-tuple:

```
["store-name", "/store-specific-path-name"]
```

The interface then takes care of correctly dispatching various operations on path names to the appropriate stores, and integrating the results back into the client-desired namespace.

Store service providers must conform to the Service Provider Interface (SPI) as declared by the package DBMS\_DBFS\_CONTENT\_SPI - the SPI is not a client-side interface and serves as a private contract between the implementation of the content interface and various stores that wish to be pluggable into it.

The content interface defines client-visible behavior (normal and exceptional) of various store operations, while allowing different stores to implement as rich a set of features as they choose - the interface allows stores to self-describe their capabilities and allows intelligent client applications to tune their behavior based on these capabilities (rather than hard-code logic specific to stores identified by name or by implementation).

## Security Model

The `DBMS_DBFS_CONTENT` package runs under `AUTHID CURRENT_USER`.

## Constants

The DBMS\_DBFS\_CONTENT package uses the constants shown in the following tables:

- [DBMS\\_DBFS\\_CONTENT Constants - Path Names](#)
- [DBMS\\_DBFS\\_CONTENT Constants - ContentID](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Properties](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Path Name Types](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Lock Types](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Standard Properties](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Property Access Flags](#)
- [DBMS\\_DBFS\\_CONTENT Constants - Operation Codes](#)

### Path Name Constants and Types

The following constants are useful for declaring paths and item names. Paths are limited to 1024 characters and item names are limited to 256 characters.

**Table 47–1 DBMS\_DBFS\_CONTENT Constants - Path Names**

Constant	Type	Value	Description
NAME_MAX	PLS_INTEGER	256	Maximum length of an absolute path name visible to clients
NAME_T	VARCHAR2 (256)	NAME_MAX	Portable alias for string that can represent component names
PATH_MAX	PLS_INTEGER	1024	Maximum length of any individual component of an absolute path name visible to clients
PATH_T	VARCHAR2 (1024)	PATH_MAX	Portable alias for string that can represent path names

### ContentID Constants

Stores may expose to the user a unique identifier that represents a particular path item in the store. These identifiers are limited to 128 characters.

**Table 47–2 DBMS\_DBFS\_CONTENT Constants - ContentID**

Constant	Type	Value	Description
CONTENT_ID_MAX	PLS_INTEGER	128	Maximum length of a store-specific provider-generated contentID that identifies a file-type content item

**Table 47-2 (Cont.) DBMS\_DBFS\_CONTENT Constants - ContentID**

Constant	Type	Value	Description
CONTENT_ID_T	RAW(128)	CONTENT_ID_MAX	Portable alias for raw buffers that can represent contentID values

**Properties Constants**

Every path name in a store is associated with a set of properties. For simplicity and to provide generic basis, each property is identified by a string "name", has a string "value" (which might be NULL if unset or undefined or unsupported by a specific store implementation) and a value "typecode" (a numeric discriminant for the actual type of value held in the "value" string.)

Coercing property values to strings has the advantage of making the various interfaces uniform and compact (and can even simplify implementation of the underlying stores), but has the disadvantage of the potential for information loss during conversions to and from strings.

It is expected that clients and stores use well-defined database conventions for these conversions, and use the typecode field as appropriate.

A typecode is a numeric value representing the true type of a string-coerced property value. Simple scalar types (numbers, dates, timestamps, etc.) can be depended on by clients and must be implemented by stores.

Since standard RDBMS typecodes are positive integers, the DBMS\_DBFS\_CONTENT interface allows negative integers to represent client-defined types by negative typecodes. These typecodes do not conflict with standard typecodes, are maintained persistently and returned to the client as needed, but need not be interpreted by the DBFS content API or any particular store. Portable client applications should not use user-defined typecodes as a back door way of passing information to specific stores.

**Table 47-3 DBMS\_DBFS\_CONTENT Constants - Properties**

Constant	Type	Value	Description
PROPNAME_MAX	PLS_INTEGER	32	Maximum length of a property name
PROPNAME_T	VARCHAR2(32)	PROPNAME_MAX	Portable alias for string that can represent property names
PROPVAL_MAX	PLS_INTEGER	1024	Maximum length of the string value of a property
PROPVAL_T	VARCHAR2(1024)	PATH_MAX	Portable alias for string that can represent property values

**Path Name Types**

Path items in a store have a item type associated with them. These types represent the kind of entry the item represents in the store.



**Table 47-4 DBMS\_DBFS\_CONTENT Constants - Path Name Types**

Constant	Type	Value	Description
TYPE_FILE	PLS_INTEGER	1	A regular file storing content (a logically linear sequence of bytes accessed as a BLOB)
TYPE_DIRECTORY	PLS_INTEGER	2	A container of other path name types, including file types
TYPE_LINK	PLS_INTEGER	3	A symbolic link (that is, an uninterpreted string value associated with a path name). Since symbolic links may represent path names that fall outside the scope of any given store (or even the entire aggregation of stores managed by the DBMS_DBFS_CONTENT interface), or may not even represent path names, clients must be careful in creating symbolic links, and stores must be careful in trying to resolve these links internally.
TYPE_REFERENCE	PLS_INTEGER	4	A hard link which is always a valid path name alias to content

### Store Features

In order to provide a common programmatic interface to as many different types of stores as possible, the DBFS content API leaves some of the behavior of various operations to individual store providers to define and implement.

However, it is still important to provide client-side programmers with an API that is sufficiently rich and conducive to portable applications.

The DBFS content API achieves this by allowing different store service providers (and different stores) to describe themselves through a "feature set" (a bitmask indicating which features they support and which ones they do not).

Using the feature set, it is possible, albeit tricky, for client applications to compensate for the feature deficiencies of specific stores by implementing additional logic on the client-side, and deferring complex operations to stores capable of supporting them.

**Table 47-5 DBMS\_DBFS\_CONTENT Constants - Store Features**

Constant	Type	Value	Description
FEATURE_FOLDERS	PLS_INTEGER	1	Set if the store supports folders (or directories) as part of hierarchical path names

**Table 47–5 (Cont.) DBMS\_DBFS\_CONTENT Constants - Store Features**

Constant	Type	Value	Description
FEATURE_FOIAT	PLS_INTEGER	2	Set if implicit folder operations within the store (performed as part of a client-requested operation) runs inside autonomous transactions. In general, the use of autonomous transactions is a compromise between (a) simplicity in the implementation and client-controlled transaction scope for all operations, at the cost of greatly reduced concurrency (FEATURE_FOIAT not set), versus (b) more complex implementation and smaller client-controlled transaction scope, at the benefit of greatly increased concurrency (FEATURE_FOIAT set).
FEATURE_NOWAIT	PLS_INTEGER	4	Set if the store allows <code>nowait</code> gets of path elements. The default behavior is to wait for row locks; if <code>nowait</code> gets are implemented, the <code>get</code> operation raises an ORA-54 exception if the path element is already locked by another transaction.
FEATURE_ACLS	PLS_INTEGER	8	Set if the store supports Access Control Lists (ACLs) and internal authorization or checking based on these ACLs. ACLs are standard properties but a store may do nothing more than store and retrieve the ACLs without interpreting them in any way.
FEATURE_LINKS	PLS_INTEGER	16	Set if the store supports symbolic links, and if certain types of symbolic links (specifically non-absolute path names) can be internally resolved by the store itself
FEATURE_LINK_DEREF	PLS_INTEGER	32	Set if the store supports symbolic links, and if certain types of symbolic links (specifically non-absolute path names) can be internally resolved by the store itself
FEATURE_REFERENCES	PLS_INTEGER	64	Set if the store supports hard links

**Table 47-5 (Cont.) DBMS\_DBFS\_CONTENT Constants - Store Features**

Constant	Type	Value	Description
FEATURE_LOCKING	PLS_INTEGER	128	Set if the store supports user-level locks (read-only, write-only, read-write) that can be applied on various items of the store, and if the store uses these lock settings to control various types of access to the locked items. User-level locks are orthogonal to transaction locks and persist beyond the scope of any specific transaction, session, or connection — this implies that the store itself may not be able to clean up after dangling locks, and client-applications need to perform any garbage collection.
FEATURE_LOCK_HIERARCHY	PLS_INTEGER	256	Set if the store allows a user-lock to control access to the entire sub-tree under the locked path name. A simpler locking model would have locking semantics apply only to a specific path name, and depend on the locks placed on its parents or children (unless the requested operation would implicitly need to modify these parents or children).
FEATURE_LOCK_CONVERT	PLS_INTEGER	512	Set if the store supports upgrade or downgrade of locks from one mode to another
FEATURE_VERSIONING	PLS_INTEGER	1024	Set if the store supports at least a linear versioning and version management. Different versions of the same path name are identified by monotonic version numbers, with a version-nonqualified path name representing the latest version.
FEATURE_VERSION_PATH	PLS_INTEGER	2048	Set if the store supports a hierarchical namespace for different versions of a path name
FEATURE_SOFT_DELETES	PLS_INTEGER	4096	Set if the store supports a "soft-delete", that is, the ability to delete a path name and make it invisible to normal operations, but retain the ability to restore the path name later (as long as it has not been overwritten by a new create operation). The store also supports purging soft-deleted path names (making them truly deleted), and navigation modes that show soft-deleted items.
FEATURE_HASHING	PLS_INTEGER	8192	Set if the store automatically computes and maintains some type of a secure hash of the contents of a path name (typically a TYPE_FILE path).

**Table 47-5 (Cont.) DBMS\_DBFS\_CONTENT Constants - Store Features**

Constant	Type	Value	Description
FEATURE_HASH_LOOKUP	PLS_INTEGER	16384	Set if the store allows "content-based addressing", that is, the ability to locate a content item based, not on its path name, but on its content hash.
FEATURE_FILTERING	PLS_INTEGER	32768	<p>Set if the store allows clients to pass a filter function (a PL/SQL function conforming to the signature below) that returns a logical boolean indicating if a given store item satisfies a selection predicate. Stores that support filtering may be able to more efficiently perform item listing, directory navigation, and deletions by embedding the filtering logic inside their implementation. If filtering is not supported, clients can retrieve more items than necessary and perform the filtering checks themselves, albeit less efficiently.</p> <p>A filter predicate is a function with the following signature:</p> <pre>function filterFunction(   path          IN  VARCHAR2,   store_name    IN  VARCHAR2,   opcode        IN  INTEGER,   item_type     IN  INTEGER,   properties    IN  DBMS_DBFS_ CONTENT_PROPERTIES_T,   content       IN  BLOB)   RETURN INTEGER;</pre> <p>Any PL/SQL function conforming to this signature can examine the contents and properties of a store item, and determine if the item satisfies the selection criterion for the current operation. Any nonzero return value results in the DBMS_DBFS_CONTENT interface processing the item as part of the current operation; a return value that is zero or NULL results in the item being skipped from processing.</p>
FEATURE_SEARCHING	PLS_INTEGER	65536	Set if the store allows clients to pass a text-search filter query to locate <code>type_file</code> path names based on their content. Stores that support searching may use indexes to accelerate such searches; otherwise, clients need to build their own indexes, or else search a potentially larger set of items to locate the ones of interest for the current search.

**Table 47-5 (Cont.) DBMS\_DBFS\_CONTENT Constants - Store Features**

Constant	Type	Value	Description
FEATURE_ASOF	PLS_INTEGER	131072	Set if the store allows clients to use a flashback timestamp in query operations (non-mutating <a href="#">GETPATH Procedures</a> , <a href="#">LIST Function</a> , <a href="#">SEARCH Function</a> ).
FEATURE_PROVIDER_PROPS	PLS_INTEGER	262144	Set if the store allows per-operation properties (that control the behavior of the store with regard to the current operation, as opposed to properties associated with individual items).
FEATURE_SNAPSHOTS	PLS_INTEGER	524288	Set if the store allows the use of named, read-only snapshots of its contents. It is up to the provider to implement snapshots using any suitable means (including creating immediate copies of the content, or using copy-on-write) and managing dependencies between snapshots and its parent content view.
FEATURE_CLONES	PLS_INTEGER	1048576	Set if the store allows the use of named, writable clones of its contents. It is up to the provider to implement clones using any suitable means (including creating immediate copies of the content, or using copy-on-write) and managing dependencies between clones and its parent content view.
FEATURE_LOCATOR	PLS_INTEGER	2097152	Set if the store allows direct access to file contents through a LOB locator. Stores that internally manipulate the file contents, perhaps by shredding or reassembling them in separate pieces, performing other transformations, and so on, cannot transparently give out a LOB locator to clients. The file contents of these stores should be accessed using the buffer-based interfaces.
FEATURE_CONTENT_ID	PLS_INTEGER	4194304	Set if the store allows a "pathless", contentID-based access to files (there is no notion of a directory, link, or reference in this model)
FEATURE_LAZY_PATH	PLS_INTEGER	8388608	Set if the store allows a lazy binding of a path name to file content elements that are otherwise identified by a contentID; this feature makes sense only in conjunction with FEATURE_CONTENT_ID

### Lock types

Stores that support locking should implement 3 types of locks: `LOCK_READ_ONLY`, `LOCK_WRITE_ONLY`, and `LOCK_READ_WRITE`.

User-locks (of one of these 3 types) can be associated with a user-supplied `lock_data`. This is not interpreted by the store, but can be used by client applications for their own purposes. For example, the user-data could indicate the time at which the lock was placed, assuming some part of the client application is interested in later using this information to control its actions, such as garbage collect stale locks or explicitly break locks.

In the simplest locking model, a `LOCK_READ_ONLY` prevents all explicit modifications to a path name (but allows implicit modifications, and changes to parent/child path names). A `LOCK_WRITE_ONLY` prevents all explicit reads to the path name (but allows implicit reads, and reads to parent/child path names). A `LOCK_READ_WRITE` allows both.

All locks are associated with a "principal" performing the locking operation; stores that support locking are expected to preserve this information, and use it to perform read or write lock checking (see `opt_locker`).

More complex lock models: multiple read-locks, lock-scoping across path name hierarchies, lock conversions, group-locking, and other strategies, are possible but currently not defined by the content interface.

**Table 47-6 DBMS\_DBFS\_CONTENT Constants - Lock Types**

Constant	Type	Value	Description
<code>LOCK_READ_ONLY</code>	<code>PLS_INTEGER</code>	1	Locks as read-only
<code>LOCK_WRITE_ONLY</code>	<code>PLS_INTEGER</code>	2	Locks as write-only
<code>LOCK_READ_WRITE</code>	<code>PLS_INTEGER</code>	3	Locks as read-write

### Standard properties

Standard properties are well-defined, mandatory properties associated with all content path names that all stores should support (in the manner described by the content interface), with some concessions. For example, a read-only store need not implement a `modification_time` or `creation_time`; stores created against tables with a fixed-schema may choose reasonable defaults for as many of these properties as needed, and so on.

All standard properties informally use the `STD` namespace. Clients and stores should avoid using this namespace to define their own properties since this can cause conflicts in future.

The menu of standard properties is expected to be fairly stable over time.

**Table 47-7 DBMS\_DBFS\_CONTENT Constants - Standard Properties**

Constant	Type	Value	Description
<code>STD_ACCESS_TIME</code>	<code>VARCHAR2(32)</code>	'std:access_time'	<code>TYPECODE_TIMESTAMP</code> in UTC: The time of last access of a path name's contents

**Table 47-7 (Cont.) DBMS\_DBFS\_CONTENT Constants - Standard Properties**

Constant	Type	Value	Description
STD_ACL	VARCHAR2 (32)	'std:acl'	TYPECODE_VARCHAR2: The access control list (in standard ACL syntax) associated with the path name
STD_CANONICAL_PATH	VARCHAR2 (32)	'std:canonical_path'	TYPECODE_VARCHAR2: The canonical store-specific path name of an item, suitably cleaned up (leading or trailing "/" collapsed or trimmed, and so on)
STD_CHANGE_TIME	VARCHAR2 (32)	'std:change_time'	TYPECODE_TIMESTAMP in UTC: The time of last change to the metadata of a path name
STD_CHILDREN	VARCHAR2 (32)	'std:children'	TYPECODE_NUMBER: The number of child directories/folders a directory/folder path has (this property should be available in providers that support the FEATURE_FOLDERS feature)
STD_CONTENT_TYPE	VARCHAR2 (32)	'std:content_type'	TYPECODE_NUMBER: The number of child directories/folders a directory/folder path has (this property should be available in providers that support the FEATURE_FOLDERS feature)
STD_CREATION_TIME	VARCHAR2 (32)	'std:creation_time'	TYPECODE_TIMESTAMP in UTC: The time at which the item was created (once set, this value never changes for the lifetime of the path name)
STD_DELETED	VARCHAR2 (32)	'std:deleted'	TYPECODE_NUMBER as a BOOLEAN: Set to a nonzero number if the path name has been soft-deleted but not yet purged.
STD_GUID	VARCHAR2 (32)	'std:guid'	TYPECODE_NUMBER: A store-specific unique identifier for a path name. Clients must not depend on the GUID being unique across different stores, but a given ( <i>store-name</i> , <i>store-specific-path name</i> ) has a stable and unique GUID for its lifetime.
STD_LENGTH	VARCHAR2 (32)	'std:length'	TYPECODE_NUMBER: The length of the content (BLOB) of a TYPE_FILE/TYPE_REFERENCE path, or the length of the referent of a TYPE_LINK symbolic link. Directories do not have a well-defined length and stores are free to set this property to zero, NULL, or any other value.

**Table 47-7 (Cont.) DBMS\_DBFS\_CONTENT Constants - Standard Properties**

Constant	Type	Value	Description
STD_MODIFICATION_TIME	VARCHAR2 (32)	'std:modification_time'	TYPECODE_TIMESTAMP in UTC: The time of last change to the data associated with a path name. Change to the content of a TYPE_FILE/TYPE_REFERENCE path, the referent of the TYPE_LINK path, and addition or deletion of immediate children in a TYPE_DIRECTORY path, all constitute data changes.
STD_OWNER	VARCHAR2 (32)	'std:owner'	TYPECODE_VARCHAR2: A client-supplied (or implicit) owner name for the path name. The owner name may be used (along with the current "principal") for access checks by stores that support ACLs, locking, or both.
STD_PARENT_GUID	VARCHAR2 (32)	'std:parent_guid'	TYPECODE_NUMBER: A store-specific unique identifier for the parent of a path name. Clients must not depend on the GUID being unique across different stores, but a given ( <i>store-name, store-specific-path name</i> ) has a stable and unique GUID for its lifetime.  STD_PARENT_GUID(path name) == STD_GUID(parent (path name))
STD_REFERENT	VARCHAR2 (32)	'std:referent'	TYPECODE_VARCHAR2: The content of the symbolic link of a TYPE_LINK path; NULL otherwise. As mentioned, the STD_REFERENT can be an arbitrary string and must not necessarily be interpreted as path name by clients (or such interpretation should be done with great care).

**Optional Properties**

Optional properties are well-defined but non-mandatory properties associated with all content path names that all stores are free to support (but only in the manner described by the DBFS content API). Clients should be prepared to deal with stores that support none of the optional properties.

All optional properties informally use the "opt:" namespace. Clients and stores should avoid using this namespace to define their own properties since this can cause conflicts in the future.

The menu of optional properties is expected to be expand over time.



**Table 47-8 DBMS\_DBFS\_CONTENT Constants - Optional Properties**

Constant	Type	Value	Description
OPT_HASH_TYPE	VARCHAR2 (32)	'opt:hash_type'	TYPECODE_NUMBER: The number of (compatible) locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the <code>opt_locker</code> must specify all lockers (with repeats so that lock counts can be correctly maintained).
OPT_HASH_VALUE	VARCHAR2 (32)	'opt:hash_value'	TYPECODE_NUMBER: The hash value of type <code>OPT_HASH_TYPE</code> describing the content of the path name.
OPT_LOCK_COUNT	VARCHAR2 (32)	'opt:lock_count'	TYPECODE_NUMBER: The number of (compatible) locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the <code>opt_locker</code> must specify all lockers (with repeats so that lock counts can be correctly maintained).
OPT_LOCK_DATA	VARCHAR2 (32)	'opt:lock_data'	TYPECODE_NUMBER: The client-supplied user-data associated with a user-lock, uninterpreted by the store.
OPT_LOCKER	VARCHAR2 (32)	'opt:locker'	TYPECODE_NUMBER: One or more implicit or client-specified principals that applied a user-lock on a path name.
OPT_LOCK_STATUS	VARCHAR2 (32)	'opt:lock_status'	TYPECODE_NUMBER: One of the <code>LOCK_READ_ONLY</code> , <code>LOCK_WRITE_ONLY</code> , <code>LOCK_READ_WRITE</code> values describing the type of lock currently applied on a path name.
OPT_VERSION	VARCHAR2 (32)	'opt:version'	TYPECODE_NUMBER: A sequence number for linear versioning of a path name.
OPT_VERSION_PATH	VARCHAR2 (32)	'opt:version_path'	TYPECODE_NUMBER: A version-path name for hierarchical versioning of a path name.
OPT_CONTENT_ID	VARCHAR2 (32)	'opt:content_id'	TYPECODE_NUMBER: A provider-generated store-specific unique contentID in the form of a string for a file content element (that may optionally not be associated with a path; see <code>FEATURE_CONTENT_ID</code> and <code>FEATURE_LAZY_PATH</code> ).

### Property Access Flags

Content interface methods to get or set properties can use combinations of property access flags to fetch properties from different name spaces in a single interface call.

**Table 47–9 DBMS\_DBFS\_CONTENT Constants - Property Access Flags**

Constant	Type	Value	Description
PROP_NONE	PLS_INTEGER	0	None: used when the client is not interested in any properties, and is invoking the content access method for other reasons (path name existence or lockability validation, data access, and so on)
PROP_STD	PLS_INTEGER	1	Mandatory: used when the client is interested in the standard properties; all standard properties are retrieved if this flag is specified.
PROP_OPT	PLS_INTEGER	2	Optional: used when the client is interested in the optional properties; all optional properties are retrieved if this flag is specified.
PROP_USR	PLS_INTEGER	3	User-defined: used when the client is interested in the user-defined properties; all user-defined properties are retrieved if this flag is specified.
PROP_ALL	PLS_INTEGER	PROP_STD + PROP_OPT + PROP_USR;	All: an alias for the combination of all standard, optional, and user-defined properties
PROP_DATA	PLS_INTEGER	8	Content: used when the client is interested only in data access, and does not care about properties
PROP_SPC	PLS_INTEGER	16	Specific: used when the client is interested in a mix-and-match of different subsets of various property name spaces; the names of the specific properties to fetch are passed into the content interface method call as arguments, and only these property values are fetched and returned to the client. This is useful in cases where there are a very large number of properties potentially accessible, but the client is interested in only a small number of them (and knows the names of these "interesting" properties beforehand).  PROP_SPC is applicable only to the various GETPATH operations. Other operations that specify properties ignore PROP_SPC specifications.

**Operation Codes**

All of the operations in the DBFS content API are represented as abstract opcodes.

Clients can use these opcodes to directly and explicitly by invoking the [CHECKACCESS Function](#) to verify if a particular operation can be invoked by a given principal on a particular path name.

**Table 47-10 DBMS\_DBFS\_CONTENT Constants - Operation Codes**

Constant	Type	Value	Description
OP_CREATE	PLS_INTEGER	1	Create a path item
OP_CREATEFILE	PLS_INTEGER	OP_CREATE	Create a file
OP_CREATELINK	PLS_INTEGER	OP_CREATE	Create a soft link
OP_CREATEREFERENCE	PLS_INTEGER	OP_CREATE	Create a reference (hard link)
OP_DELETE	PLS_INTEGER	2	Soft-deletion, purge, and restore operations are all represented by OP_DELETE
OP_DELETEFILE	PLS_INTEGER	OP_DELETE	Delete a file
OP_DELETEDIRECTORY	PLS_INTEGER	OP_DELETE	Delete a directory
OP_RESTORE	PLS_INTEGER	OP_DELETE	Restore a soft-deleted path item
OP_PURGE	PLS_INTEGER	OP_DELETE	Purge a soft-deleted path item
OP_READ	PLS_INTEGER	3	Read from a path item
OP_GET	PLS_INTEGER	OP_READ	Get a path item for either read or update operations
OP_WRITE	PLS_INTEGER	4	Write a path item
OP_PUT	PLS_INTEGER	OP_WRITE	Put (write) to a path item
OP_RENAME	PLS_INTEGER	5	Rename a path item
OP_RENAMEFROM	PLS_INTEGER	OP_RENAME	Operations performed on the source of a rename
OP_RENAMETO	PLS_INTEGER	OP_RENAME	Operations performed on the destination of a rename
OP_SETPATH	PLS_INTEGER	OP_RENAME	Set a path item name
OP_LIST	PLS_INTEGER	6	Perform a path listing
OP_SEARCH	PLS_INTEGER	7	Perform a search
OP_LOCK	PLS_INTEGER	8	Lock a path item
OP_UNLOCK	PLS_INTEGER	9	Unlock a path item
OP_ACL	PLS_INTEGER	10	An implicit operation invoked during an OP_CREATE or OP_PUT that specifies a STD_ACL property; the operation tests to see if the principal is allowed to set or change the ACL of a store item
OP_STORE	PLS_INTEGER	11	A catch-all category for miscellaneous store operations that do not fall under any of the other operational interfaces

## Exceptions

DBFS content API operations can raise any one of the following top-level exceptions.

Clients can program against these specific exceptions in their error handlers without worrying about the specific store implementations of the underlying error signally code.

Store service providers, for their part, should do their best to trap or wrap any internal exceptions into one of the following exception types, as appropriate

**Table 47-11 DBMS\_DBFS\_CONTENT Exceptions**

<b>Exception</b>	<b>Code</b>	<b>Description</b>
PATH_EXISTS	64000	A specified path name already exists
INVALID_PARENT	64001	Parent of a specified path name does not exist
INVALID_PATH	64002	Specified path name does not exist, or is not valid
UNSUPPORTED_OPERATION	64003	An operation unsupported by a store was invoked
INVALID_ARGUMENTS	64004	An operation was invoked with invalid arguments
INVALID_ACCESS	64005	Access control checks failed for the current operation
LOCK_CONFLICT	64006	Current operation failed lock conflict check
INVALID_STORE	64007	An invalid store name was specified
INVALID_MOUNT	64008	An invalid mount point was specified
INVALID_PROVIDER	64009	An invalid provider-package was specified
READONLY_PATH	64010	A mutating operation was invoked on a read-only mount or store

## Operational Notes

- [Implementation](#)
- [Path Names](#)
- [Creation Operations](#)
- [Deletion Operations](#)
- [Get \(Retrieve\) and Put \(Insert\) Operations](#)
- [Rename and Move Operations](#)
- [Directory Navigation and Search](#)
- [Locking Operations](#)
- [Access Check Operation](#)

### Implementation

Since the interconnection of the [DBMS\\_DBFS\\_CONTENT](#) interface and the provider SPI is a 1-to-many pluggable architecture, the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

There are no explicit `INIT` or `FINI` methods to indicate when the `DBMS_DBFS_CONTENT` interface plugs or unplugs a particular provider SPI. Provider SPIs must be willing to auto-initialize themselves at any SPI entry-point.

All operations performed by a service provider are "stateless" in that they are complete operations unto themselves. If state is necessary to be maintained for some reason, then the state must be maintained in data structures such as auxiliary tables that can be queried as needed.

### Path Names

All path names used in the provider SPI are store-qualified in pair form (`store_name`, `pathname`) where the path name is rooted within the store namespace.

Stores and their providers that support contentID-based access (see `FEATURE_CONTENT_ID` in [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)) also support a form of addressing that is not based on path names. Content items are identified by an explicit store name, a `NULL` path name, and possibly a contentID specified as a parameter or by way of the `OPT_CONTENT_ID` (see [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)) property.

Not all operations are supported with contentID-based access, and applications should depend only on the simplest create or delete functionality being available.

### Creation Operations

The provider SPI must allow the DBFS content API to create directory, file, link, and reference elements subject to store feature support.

All of the creation subprograms require a valid path name, but note the special exemption for contentID-based access. Creation subprograms can optionally specify properties to be associated with the path name as it is created. It is also possible for clients to return item properties after the creation completes so that automatically generated properties, such as `STD_CREATION_TIME` (see [DBMS\\_DBFS\\_CONTENT Constants - Standard Properties](#)) are immediately available to clients. The exact set of properties fetched back is controlled by the various `PROP_XXX` bitmasks in the `prop_flags` parameter.

Links and references require an additional path name to associate with the primary path name.

File path names can optionally specify a BLOB value to use to initially populate the underlying file content (the provided BLOB may be any valid LOB). On creation, the underlying LOB is returned to the client, provided that `PROP_DATA` is specified in the `prop_flags` parameter.

Nondirectory path names require that their parent directory be created first. Directory path names themselves can be recursively created with the path name hierarchy leading up to a directory created in one call.

Attempts to create paths that already exist is an error; the sole exception is path names that are "soft-deleted" (as discussed in the context of [Deletion Operations](#)). In these cases, the soft-deleted item is implicitly purged, and the new item creation is attempted.

Stores and their providers that support contentID-based access accept an explicit store name and a `NULL` path to create a new content element. The contentID generated for this element is available by means of the `OPT_CONTENT_ID` property (see [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)), contentID-based creation being automatically implied by `PROP_OPT` property in the `prop_flags` parameter).

The newly created element may also have an internally generated path name if `FEATURE_LAZY_PATH` property is not supported (see [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)) and this path is available by way of the `STD_CANONICAL_PATH` property (see [DBMS\\_DBFS\\_CONTENT Constants - Standard Properties](#)).

Only file elements are candidates for contentID-based access.

### **Deletion Operations**

The provider SPI must allow the DBFS content API to delete directory, file, link, and reference elements (subject to store feature support).

By default, the deletions are permanent, removing the successfully deleted items on transaction commit, but stores may also support "soft-delete" features. If requested by the client, soft-deleted items are retained by the store, although they are not typically visible in normal listings or searches.

Soft-deleted items can be restored, or explicitly purged.

Directory path names can be recursively deleted, with the path name hierarchy below a directory deleted in one call. Non-recursive deletions can be performed only on empty directories. Recursive soft-deletions apply the soft-delete to all of the items being deleted.

Individual path names, as well as all soft-deleted path names under a directory, can be restored or purged by means of the various restore and purge subprograms.

Providers that support filtering can use the provider filter to identify subsets of items to delete. This makes most sense for bulk operations such as the [DELETEDIRECTORY Procedure](#), [PROPVARCHAR2 Function](#), and [RESTOREALL Procedure](#), but all of the deletion-related operations accept a `filter` argument.

Stores and their providers that support contentID-based access can also allow file items to be deleted by specifying their contentID.

### **Get (Retrieve) and Put (Insert) Operations**

Existing path items can be accessed (for query or for update) and modified using simple get and put subprograms. All path names allow their metadata (properties) to

be read and modified. On completion of the call, the client can request specific properties to be fetched by means of the `prop_flags` parameter.

File path names allow their data (content) to be read and modified. On completion of the call, the client can use the `PROP_DATA` bitmasks in the `prop_flags` parameter to request a new `BLOB` locator to continue data access.

Files can also be read or written without using `BLOB` locators by explicitly specifying logical offsets, buffer amounts, and a suitably sized buffer.

Update accesses must specify the `forUpdate` flag. Access to link path names can be implicitly and internally de-referenced by stores (subject to feature support) if the `deref` flag is specified, however, this may have undetermined outcomes since symbolic links are not always resolvable.

The read methods, such as the [GETPATH Procedures](#) where `forUpdate` is specified as 0, also accepts a valid `asof` timestamp in the `ctx` parameter that can be used by stores to implement "as of" style flashback queries. Mutating versions of the [GETPATH Procedures](#) and the [PUTPATH Procedures](#) methods do not support "as of" modes of operation.

The [GETPATHNOWAIT Procedures](#) implies that the operation is for an update, and, if implemented (see `FEATURE_NOWAIT` in [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)), this allows providers to return an exception (ORA-00054) rather than wait for row locks.

### Rename and Move Operations

Path names can be renamed or moved, possibly across directory hierarchies and mount-points, but within the same store.

Nondirectory path names previously accessible by way of specifying the `oldPath` parameter are renamed as a single item subsequently accessible by specifying `newPath`, assuming that `newPath` does not already exist.

If `newPath` exists and is not a directory, the action of renaming implicitly deletes the existing item before renaming `oldPath`. If the `newPath` exists and is a directory, `oldPath` is moved into the target directory.

Directory path names previously accessible by way of `oldPath` are renamed by moving the directory and all of its children to `newPath` (if it does not already exist) or as children of `newPath` (if it exists and is a directory).

Stores and their providers that support `contentID`-based access and lazy path name binding also support the [SETPATH Procedures](#) that associates an existing `contentID` with a new "path".

### Directory Navigation and Search

The `DBMS_DBFS_CONTENT` interface can list or search the contents of directory path names, optionally operating recursively into sub-directories, optionally seeing soft-deleted items, optionally using flashback "as of" a provided timestamp, and optionally filtering items in or out within the store based on list or search predicates.

### Locking Operations

Clients of the `DBMS_DBFS_CONTENT` interface can apply user-level locks to any valid path name (subject to store feature support), associate the lock with user-data, and subsequently unlock these path names.

The status of locked items is available using various optional properties (note the previous discussion regarding `opt_lock`).

It is the responsibility of the store (assuming it supports user-defined lock checking) to ensure that lock and unlock operations are performed in a consistent manner.

#### **Access Check Operation**

This operation ascertains if a given path name (`store_name`, `path`, `pathtype`) can be manipulated by operation (see the various `DBMS_DBFS_CONTENT.OP_XXX` opcodes in [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)) by the user acting on the store utilizing the `principal` parameter. This is a convenience function for the `DBMS_DBFS_CONTENT` interface; a store that supports access control still internally performs these checks to guarantee security.



---

## Data Structures

The `DBMS_DBFS_CONTENT` package defines `RECORD` types and `TABLE` types.

### RECORD Types

- [FEATURE\\_T Record Type](#)
- [MOUNT\\_T Record Type](#)
- [PATH\\_ITEM\\_T Record Type](#)
- [PROP\\_ITEM\\_T Record Type](#)
- [PROPERTY\\_T Record Type](#)
- [STORE\\_T Record Type](#)

### TABLE Types

- [FEATURES\\_T Table Type](#)
- [MOUNTS\\_T Table Type](#)
- [PATH\\_ITEMS\\_T Table Type](#)
- [PROP\\_ITEMS\\_T Table Type](#)
- [PROPERTIES\\_T Table Type](#)
- [STORES\\_T Table Type](#)

### Usage Notes

There is an approximate correspondence between `DBMS_DBFS_CONTENT_PROPERTY_T` and `PROPERTY_T` — the former is a SQL object type that describes the full property tuple, while the latter is a PL/SQL record type that describes only the property value component.

Likewise, there is an approximate correspondence between `DBMS_DBFS_CONTENT_PROPERTIES_T` and `PROPERTIES_T` — the former is a SQL nested table type, while the latter is a PL/SQL hash table type.

Dynamic SQL calling conventions force the use of SQL types, but PL/SQL code may be implemented more conveniently in terms of the hash-table types.

The `DBMS_DBFS_CONTENT` interface provides convenient utility functions to convert between `DBMS_DBFS_CONTENT_PROPERTIES_T` and `PROPERTIES_T` (see `propertiesT2H` and `propertiesH2T`).

Clients can query the `DBMS_DBFS_CONTENT` interface for the list of available stores, determine which store is to handle access to a given path name, and determine the feature set for the store.

## FEATURE\_T Record Type

This type describes a store mount point and its properties.

### Syntax

```
TYPE feature_t IS RECORD (  
    feature_name    VARCHAR2(32),  
    feature_mask    INTEGER,  
    feature_state   VARCHAR2(3));
```

### Fields

**Table 47–12** MOUNT\_T Fields

Field	Description
feature_name	Name of feature
feature_mask	Value used to mask off all other bits other than this feature in the feature value
feature_state	'YES' or 'NO' depending on whether the feature is supported on this store

## MOUNT\_T Record Type

This type describes a store mount point and its properties.

### Syntax

```
TYPE mount_t IS RECORD (
  store_name      VARCHAR2(32),
  store_id        NUMBER,
  provider_name   VARCHAR2(32),
  provider_pkg    VARCHAR2(32),
  provider_id     NUMBER,
  provider_version VARCHAR2(32),
  store_features  INTEGER,
  store_guid      NUMBER,
  store_mount     NAME_T,
  mount_properties DBMS_DBFS_CONTENT_PROPERTIES_T);
```

### Fields

**Table 47–13** MOUNT\_T Fields

Field	Description
store_name	Name of store
store_id	ID of store
provider_name	Name of the content store
provider_pkg	PL/SQL package name for the content store
provider_id	Unique identifier for the content store
provider_version	Version number for the content store
respos_features	Features supported by this content store
store_guid	Unique ID for this instance of the store
store_mount	Location at which this store instance is mounted
mount_properties	Properties for this mount point (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )

## PATH\_ITEM\_T Record Type

A `PATH_ITEM_T` is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.

### Syntax

```
TYPE path_item_t IS RECORD (
    store          NAME_T,
    mount         NAME_T,
    pathname      PATH_T,
    pathtype     VARCHAR2(32),
    filedata     BLOB,
    std_access_time  TIMESTAMP,
    std_acl       VARCHAR2(1024),
    std_change_time  TIMESTAMP,
    std_children  NUMBER,
    std_content_type VARCHAR2(1024),
    std_creation_time  TIMESTAMP,
    std_deleted   INTEGER,
    std_guid     INTEGER,
    std_modification_time  TIMESTAMP,
    std_owner     VARCHAR2(32),
    std_parent_guid  INTEGER,
    std_referent  VARCHAR2(1024),
    opt_hash_type VARCHAR2(32),
    opt_hash_value VARCHAR2(128),
    opt_lock_count  INTEGER,
    opt_lock_data  VARCHAR2(128),
    opt_locker     VARCHAR2(128),
    opt_lock_status  INTEGER,
    opt_version    INTEGER,
    opt_version_path  PATH_T,
    opt_content_id  CONTENT_ID_T);
```

### Fields

**Table 47–14** *PATH\_ITEM\_T Fields*

Field	Description
store	Name of store
mount	Location at which instance of store is mounted
pathname	Name of path to item
pathtype	Type of object path (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
filedata	BLOB locator that can be used to access data in the path item
std_access_time	Time of last access of a pathname's contents
std_acl	Access Control List (in standard ACL syntax)
std_change_time	Time of last change to the metadata of a path name
std_children	Number of child directories or folders a directory or folder path (this property should be available in providers that support the <code>feature_folders</code> feature).

**Table 47-14 (Cont.) PATH\_ITEM\_T Fields**

Field	Description
std_content_type	One or more client-supplied mime-types (in standard RFC syntax) describing the path name which is typically of <code>type_file</code> . The content type s not necessarily interpreted by the store.
std_creation_time	Time at which the item was created. Once set, this value remains the same for the lifetime of the path name.
std_deleted	Set to a nonzero number if the path name has been soft-deleted but not yet purged (see <a href="#">DBMS_DBFS_CONTENT Constants - Store Features</a> )
std_guid	Store-specific unique identifier for a path name. Clients must not depend on the GUID being unique across different stores, but a given <i>store-name, store-specific-pathname</i> has a stable and unique GUID for its lifetime.
std_modification_time	Time of last change to the data associated with a path name. Changes to the content of a <code>type_file</code> or <code>type_reference</code> path, the referent of the <code>type_link</code> path, and addition or deletion of immediate children in a <code>type_directory</code> path, all constitute data changes.
std_owner	Client-supplied (or implicit) owner name for the path name
std_parent_guid	Store-specific unique identifier for the parent of a path name. Clients must not depend on the GUID being unique across different stores, but a given <i>store-name, store-specific-pathname</i> has a stable and unique GUID for its lifetime.  <code>std_parent_guid(pathname) == std_guid(parent(pathname))</code>
std_referent	Content of the symbolic link of a <code>type_link</code> path, otherwise NULL. As mentioned before, the <code>std_referent</code> can be an arbitrary string and must not necessarily be interpreted as pathname by clients (or such interpretation should be done with great care).
opt_hash_type	Type of hash provided in the <code>opt_hash_value</code> property (see <a href="#">DBMS_CRYPT</a> for possible options)
opt_hash_value	Hash value of type <code>opt_hash_type</code> describing the content of the path name
opt_lock_count	Number of compatible locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the <code>opt_locker</code> must specify all lockers with repeats so that lock counts can be correctly maintained.
opt_lock_data	Client-supplied user-data associated with a user-lock, uninterpreted by the store
opt_locker	One or more implicit or client-specified principals that applied a user-lock on a path name
opt_lock_status	One of the <code>lock_read_only</code> , <code>lock_write_only</code> , <code>lock_read_write</code> values describing the type of lock currently applied on a path name
opt_version	Sequence number for linear versioning of a path name
opt_version_path	Version path name for hierarchical versioning of a path name
opt_content_id	Stringified provider-generated store-specific unique contentID for a file element (that may optionally not be associated with a path (see <code>FEATURE_CONTENT_ID</code> and <code>FEATURE_LAZY_PATH</code> in <a href="#">DBMS_DBFS_CONTENT Constants - Store Features</a> ))

## PROP\_ITEM\_T Record Type

A `PROP_ITEM_T` is a tuple describing a (`store`, `mount`) qualified path in a store, with all user-defined properties associated with it, expanded out into individual (`name`, `value`, `type`) tuples.

### Syntax

```
TYPE prop_item_t IS RECORD (  
    store          NAME_T,  
    mount          NAME_T,  
    pathname       PATH_T,  
    property_name  PROPNAME_T,  
    property_value PROPVAL_T,  
    property_type  INTEGER);
```

### Fields

**Table 47–15** *PROP\_ITEM\_T Fields*

Field	Description
<code>store</code>	Name of store
<code>mount</code>	Location at which instance of store is mounted
<code>pathname</code>	Name of path to item
<code>property_name</code>	Name of the property
<code>property_value</code>	Value of the property
<code>property_type</code>	PL/SQL typecode for the property value

## PROPERTY\_T Record Type

This type describes a single (value, typecode) property value tuple; the property name is implied (see [PROPERTIES\\_T Table Type](#)).

### Syntax

```
TYPE property_t IS RECORD (  
    propvalue    PROPVAL_T,  
    typecode     INTEGER);
```

### Fields

**Table 47–16** *PROPERTY\_T Fields*

Field	Description
propvalue	Value of property
typecode	Typecode

## STORE\_T Record Type

This type describes a store registered with and managed by the DBMS\_DBFS\_CONTENT interface.

### Syntax

```
TYPE store_t IS RECORD (  
    store_name          VARCHAR2(32),  
    store_id            NUMBER,  
    provider_name       VARCHAR2(32),  
    provider_pkg        VARCHAR2(32),  
    provider_id         NUMBER,  
    provider_version    VARCHAR2(32),  
    store_features      INTEGER,  
    store_guid          NUMBER);
```

### Fields

**Table 47–17** STORE\_T Fields

Field	Description
store_name	Name of store
store_id	ID of store
provider_name	Name of the content store
provider_pkg	PL/SQL package name for the content store
provider_id	Unique identifier for the content store
provider_version	Version number for the content store
store_features	Features supported by this content store
store_guid	Unique ID for this instance of the store



## FEATURES\_T Table Type

A table type of [FEATURE\\_T Record Type](#).

### Syntax

```
TYPE features_t IS TABLE OF feature_t;
```

## MOUNTS\_T Table Type

A table type of [MOUNT\\_T Record Type](#).

### Syntax

```
TYPE mounts_t IS TABLE OF mount_t;
```

## PATH\_ITEMS\_T Table Type

A table type of [PATH\\_ITEM\\_T Record Type](#)

### Syntax

```
TYPE path_items_t IS TABLE OF path_item_t;
```

## PROP\_ITEMS\_T Table Type

A table type of [PATH\\_ITEM\\_T Record Type](#).

### Syntax

```
TYPE prop_items_t IS TABLE OF prop_item_t;
```

## PROPERTIES\_T Table Type

This is a name-indexed hash table of property tuples. The implicit hash-table association between the index and the value allows the client to build up the full DBMS\_DBFS\_CONTENT\_PROPERTY\_T tuples for a PROPERTIES\_T.

### Syntax

```
TYPE properties_t IS TABLE OF property_t INDEX BY propname_t;
```

## STORES\_T Table Type

This type describes a store registered with and managed by the DBMS\_DBFS\_CONTENT interface.

### Syntax

```
TYPE stores_t IS TABLE OF store_t;
```

---

## Summary of DBMS\_DBFS\_CONTENT Subprograms

**Table 47–18 DBMS\_DBFS\_CONTENT Package Subprograms**

Subprogram	Description
<a href="#">CHECKACCESS Function</a> on page 47-40	Reports if the user ( <code>principal</code> ) can perform the specified operation on the given path
<a href="#">CHECKSPI Functions and Procedures</a> on page 47-41	Checks if a user-provided package implements all of the <code>DBMS_DBFS_CONTENT_SPI</code> subprograms with the proper signatures, and reports on the conformance.
<a href="#">CREATEDIRECTORY Procedures</a> on page 47-42	Creates a directory
<a href="#">CREATEFILE Procedures</a> on page 47-43	Creates a file
<a href="#">CREATELINK Procedures</a> on page 47-44	Creates a new reference to the source file system element
<a href="#">CREATEREFERENCE Procedures</a> on page 47-45	Creates a physical link to an already existing file system element
<a href="#">DECODEFEATURES Function</a> on page 47-46	Given a feature bit set integer value, returns a <code>FEATURES_T</code> table of the feature bits as <code>FEATURE_T</code> records
<a href="#">DELETECONTENT Procedure</a> on page 47-47	Deletes the file specified by the given <code>contentID</code>
<a href="#">DELETEDIRECTORY Procedure</a> on page 47-48	Deletes a directory
<a href="#">DELETEFILE Procedure</a> on page 47-46	Deletes a file
<a href="#">FEATURENAME Function</a> on page 47-50	Given a feature bit, returns a <code>VARCHAR2</code> of that feature's name
<a href="#">FLUSHSTATS Function</a> on page 47-51	Flushes <code>DBMS_DBFS_CONTENT</code> statistics to disk
<a href="#">GETDEFAULTACL Procedure</a> on page 47-52	Returns the <code>ACL</code> parameter of the default context
<a href="#">GETDEFAULTASOF Procedure</a> on page 47-53	Returns the <code>asof</code> parameter of the default context
<a href="#">GETDEFAULTCONTEXT Procedure</a> on page 47-54	Returns the default context
<a href="#">GETDEFAULTOWNER Procedure</a> on page 47-55	Returns the <code>owner</code> parameter of the default context
<a href="#">GETDEFAULTPRINCIPAL Procedure</a> on page 47-56	Returns the <code>principal</code> parameter of the default context
<a href="#">GETFEATURESBYMOUNT Function</a> on page 47-57	Returns features of a store by mount point
<a href="#">GETFEATURESBYNAME Function</a> on page 47-58	Returns features of a store by store name
<a href="#">GETFEATURESBYPATH Function</a> on page 47-59	Returns features of a store by path
<a href="#">GETPATHBYMOUNTID Function</a> on page 47-63	Returns the full absolute path name

**Table 47–18 (Cont.) DBMS\_DBFS\_CONTENT Package Subprograms**

Subprogram	Description
<a href="#">GETPATH Procedures</a> on page 47-63	Returns existing path items (such as files and directories)
<a href="#">GETPATHBYSTOREID Function</a> on page 47-64	If the underlying GUID is found in the underlying store, returns the store-qualified path name
<a href="#">GETPATHNOWAIT Procedures</a> on page 47-65	Implies that the operation is for an update, and, if implemented, allows providers to return an exception (ORA-00054) rather than wait for row locks.
<a href="#">GETSTOREBYMOUNT Function</a> on page 47-67	Returns a store by way of its mount point
<a href="#">GETSTOREBYNAME Function</a> on page 47-68	Returns a store by way of its name
<a href="#">GETSTOREBYPATH Function</a> on page 47-69	Returns a store by way of its path
<a href="#">GETSTATS Procedure</a> on page 47-70	Returns information about DBMS_DBFS_CONTENT statistics collection
<a href="#">GETTRACE Function</a> on page 47-71	Returns whether or not DBMS_DBFS_CONTENT tracing is turned on
<a href="#">GETVERSION Function</a> on page 47-72	Returns the version of the DBMS_DBFS_CONTENT interface in a standardized format associated with a store
<a href="#">LIST Function</a> on page 47-73	Lists the path items in the specified path meeting the specified filter and other criteria
<a href="#">LISTALLCONTENT Function</a> on page 47-75	Lists all path items in all mounts
<a href="#">LISTALLPROPERTIES Function</a> on page 47-74	Returns a table of all properties for all path items in all mounts
<a href="#">LISTMOUNTS Function</a> on page 47-76	Lists all available mount points, their backing stores, and the store features
<a href="#">LISTSTORES Function</a> on page 47-77	Lists all available stores and their features
<a href="#">LOCKPATH Procedure</a> on page 47-78	Applies user-level locks to the given valid path name
<a href="#">MOUNTSTORE Procedure</a> on page 47-79	Mounts a previously registered store and binds it to the mount point
<a href="#">NORMALIZEPATH Functions</a> on page 47-81	Converts a store-specific or full-absolute path name into normalized form
<a href="#">PROPANY Functions</a> on page 47-82	Provides constructors that take one of a variety of types and return a PROPERTY_T
<a href="#">PROPERTIESH2T Function</a> on page 47-83	Converts a PROPERTY_T hash to a DBMS_DBFS_CONTENT_PROPERTIES_T table
<a href="#">PROPERTIEST2H Function</a> on page 47-84	Converts a DBMS_DBFS_CONTENT_PROPERTIES_T table to a PROPERTY_T hash
<a href="#">PROPNUMBER Function</a> on page 47-85	Is a constructor that takes a NUMBER and returns a PROPERTY_T
<a href="#">PROPRAW Function</a> on page 47-86	Is a constructor that takes a RAW and returns a PROPERTY_T
<a href="#">PROPTIMESTAMP Function</a> on page 47-87	Is a constructor that takes a TIMESTAMP and returns a PROPERTY_T



**Table 47-18 (Cont.) DBMS\_DBFS\_CONTENT Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">PROPVARCHAR2 Function</a> on page 47-88	Is a constructor that takes a <code>VARCAHR2</code> and returns a <code>PROPERTY_T</code>
<a href="#">PURGEALL Procedure</a> on page 47-89	Purges all soft-deleted entries matching the path and optional filter criteria
<a href="#">PURGEPATH Procedure</a> on page 47-90	Purges any soft-deleted versions of the given path item
<a href="#">PUTPATH Procedures</a> on page 47-91	Creates a new path item
<a href="#">REGISTERSTORE Procedure</a> on page 47-93	Registers a new store
<a href="#">RENAMEPATH Procedures</a> on page 47-94	Renames or moves a path
<a href="#">RESTOREALL Procedure</a> on page 47-95	Restores all soft-deleted path items meeting the path and filter criteria
<a href="#">RESTOREPATH Procedure</a> on page 47-96	Restores all soft-deleted path items that match the given path and filter criteria
<a href="#">SEARCH Function</a> on page 47-97	Searches for path items matching the given path and filter criteria
<a href="#">SETDEFAULTACL Procedure</a> on page 47-98	Sets the ACL parameter of the default context
<a href="#">SETDEFAULTASOF Procedure</a> on page 47-99	Sets the "as of" parameter of the default context
<a href="#">SETDEFAULTCONTEXT Procedure</a> on page 47-100	Sets the default context
<a href="#">SETDEFAULTOWNER Procedure</a> on page 47-101	Sets the "owner" parameter of the default context
<a href="#">SETDEFAULTPRINCIPAL Procedure</a> on page 47-102	Sets the "principal" parameter of the default context
<a href="#">SETPATH Procedures</a> on page 47-103	Assigns a path name to a path item represented by <code>contentID</code>
<a href="#">SETSTATS Procedure</a> on page 47-104	Enables and disables statistics collection
<a href="#">SETTRACE Procedure</a> on page 47-105	Sets <code>DBMS_DBFS_CONTENT</code> tracing on or off
<a href="#">SPACEUSAGE Procedure</a> on page 47-106	Queries file system space usage statistics
<a href="#">TRACE Procedure</a> on page 47-107	Returns a <code>CLOB</code> that contains the evaluation results
<a href="#">TRACEENABLED Function</a> on page 47-108	Determines if the current trace "severity" set by the <a href="#">SETTRACE Procedure</a> is at least as high as the given trace level
<a href="#">UNLOCKPATH Procedure</a> on page 47-109	Unlocks path items that were previously locked with the <a href="#">LOCKPATH Procedure</a>
<a href="#">UNMOUNTSTORE Procedure</a> on page 47-110	Unmounts a registered store
<a href="#">UNREGISTERSTORE Procedure</a> on page 47-111	Unregisters a store

## CHECKACCESS Function

This function reports if the user (*principal*) can perform the specified operation on the given path. This enables verifying the validity of an operation without attempting to perform the operation. If `CHECKACCESS` returns 0, then the subprogram invoked to implement that operation should fail with an error.

### Syntax

```
DBMS_DBFS_CONTENT.CHECKACCESS (
  path          IN      VARCHAR2,
  pathtype     IN      INTEGER,
  operation     IN      VARCHAR2,
  principal    IN      VARCHAR2,
  store_name   IN      VARCHAR2 DEFAULT NULL)
RETURN BOOLEAN;
```

### Parameters

**Table 47–19** *CHECKACCESS Procedure Parameters*

Parameter	Description
<code>path</code>	Name of path to check for access
<code>pathtype</code>	Type of object <code>path</code> represents (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
<code>operation</code>	Operation to be checked (see <a href="#">DBMS_DBFS_CONTENT Constants - Optional Properties</a> )
<code>principal</code>	File system user for whom the access check is made
<code>store_name</code>	Name of store

### Usage Notes

Whether or not the user invokes this function, a store that supports access control internally performs these checks to guarantee security.

## CHECKSPI Functions and Procedures

Given the name of a putative DBMS\_DBFS\_CONTENT\_SPI conforming package, this function or procedure checks whether the package implements all of the provider subprograms with the proper signatures, and reports on the conformance.

### Syntax

```
DBMS_DBFS_CONTENT.CHECKSPI (
  package_name      IN          VARCHAR2)
RETURN CLOB;
```

```
DBMS_DBFS_CONTENT.CHECKSPI (
  schema_name      IN          VARCHAR2,
  package_name     IN          VARCHAR2)
return clob;
```

```
DBMS_DBFS_CONTENT.CHECKSPI (
  package_name     IN          VARCHAR2,
  chk              IN OUT NOCOPY CLOB);
```

```
DBMS_DBFS_CONTENT.CHECKSPI (
  schema_name     in          VARCHAR2,
  package_name    in          VARCHAR2,
  chk             in out nocopy CLOB);
```

### Parameters

**Table 47–20** CHECKSPI Procedure Parameters

Parameter	Description
package_name	Name of package
schema_name	Name of schema
chk	CLOB that contains the evaluation results

### Usage Notes

- The functional form returns a cached temporary LOB of session duration with the results of the analysis. The caller is expected to manage the lifetime of this LOB, as needed.
- The procedural form generates the results of the analysis into the `chk` LOB parameter; if the value passed in is `NULL`, the results are written to the foreground trace file provided that `DBMS_DBFS_CONTENT` interface tracing is enabled. If neither tracing is enabled nor a valid LOB passed in, the checker does not provide any useful indication of the analysis (other than raise exceptions if it encounters a serious error).
- If `schema_name` is `NULL`, standard name resolution rules (current schema, private synonym, public synonym) are used to try and locate a suitable package to analyze.

## CREATEDIRECTORY Procedures

This procedure creates a directory.

### Syntax

```
DBMS_DBFS_CONTENT.CREATEDIRECTORY (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  prop_flags    IN          INTEGER      DEFAULT PROP_STD,
  recurse       IN          BOOLEAN     DEFAULT FALSE,
  store_name    IN          VARCHAR2    DEFAULT NULL,
  principal     IN          VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.CREATEDIRECTORY (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY PROPERTIES_T,
  prop_flags    IN          INTEGER      DEFAULT PROP_STD,
  recurse       IN          BOOLEAN     DEFAULT FALSE,
  store_name    IN          VARCHAR2    DEFAULT NULL,
  principal     IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 47–21** *CREATEDIRECTORY Procedure Parameters*

Parameter	Description
path	Name of path to the directory
properties	One or more properties and their values to be set, returned, or both, depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
prop_flags	Determines which properties are set, returned, or both. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>PROP_SPC</code> (see <a href="#">DBMS_DBFS_CONTENT Constants - Property Access Flags</a> ), and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
recurse	If 0, do not execute recursively; otherwise, recursively create the directories above the given directory
store_name	Name of store
principal	File system user for whom the access check is made

## CREATEFILE Procedures

This procedure creates a file.

### Syntax

```
DBMS_DBFS_CONTENT.CREATEFILE (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  content       IN OUT NOCOPY BLOB,
  prop_flags    IN          INTEGER          DEFAULT (PROP_STD + PROP_DATA),
  store_name    IN          VARCHAR2        DEFAULT NULL,
  principal     IN          VARCHAR2        DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.CREATEFILE (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY PROPERTIES_T,
  content       IN OUT NOCOPY BLOB,
  prop_flags    IN          INTEGER          DEFAULT (PROP_STD + PROP_DATA),
  store_name    IN          VARCHAR2        DEFAULT NULL,
  principal     IN          VARCHAR2        DEFAULT NULL);
```

### Parameters

**Table 47–22 CREATEFILE Procedure Parameters**

Parameter	Description
path	Name of path to the file
properties	One or more properties and their values to be set, returned, or both, depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data with which to populate the file (optional)
prop_flags	Determines which properties are set, returned, or both. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
store_name	Name of store
principal	File system user for whom the access check is made

## CREATELINK Procedures

This procedure creates a new reference to the source file system element (such as a file, or directory). The resulting reference points to the source element but does not directly share metadata with the source element. This is analogous to a UNIX file system symbolic link.

### Syntax

```
DBMS_DBFS_CONTENT.CREATELINK (
  srcPath      IN          VARCHAR2,
  dstPath      IN          VARCHAR2,
  properties   IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  prop_flags   IN          INTEGER      DEFAULT PROP_STD,
  store_name   IN          VARCHAR2    DEFAULT NULL,
  principal    IN          VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.CREATELINK (
  srcPath      IN          VARCHAR2,
  dstPath      IN          VARCHAR2,
  properties   IN OUT NOCOPY PROPERTIES_T,
  prop_flags   IN          INTEGER      DEFAULT PROP_STD,
  store_name   IN          VARCHAR2    DEFAULT NULL,
  principal    IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 47–23** CREATELINK Procedure Parameters

Parameter	Description
srcPath	File system entry with which to link
dstPath	Path of the new link element to be created
properties	One or more properties and their values to be set, returned depending, or both, on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
store_name	Name of store
principal	File system user for whom the access check is made

## CREATEREFERENCE Procedures

This procedure creates a physical link to an already existing file system element (such as file or directory). The resulting entry shares the same metadata structures as the value of the `srcPath` parameter, and so is similar to incrementing a reference count on the file system element. This is analogous to a UNIX file system hard link.

### Syntax

```
DBMS_DBFS_CONTENT.CREATEREFERENCE (
  srcPath      IN          VARCHAR2,
  dstPath      IN          VARCHAR2,
  properties   IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  prop_flags  IN          INTEGER      DEFAULT PROP_STD,
  store_name   IN          VARCHAR2   DEFAULT NULL,
  principal   IN          VARCHAR2   DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.CREATEREFERENCE (
  srcPath      IN          VARCHAR2,
  dstPath      IN          VARCHAR2,
  properties   IN OUT NOCOPY PROPERTIES_T,
  prop_flags  IN          INTEGER      DEFAULT PROP_STD,
  store_name   IN          VARCHAR2   DEFAULT NULL,
  principal   IN          VARCHAR2   DEFAULT NULL);
```

### Parameters

**Table 47–24** *CREATEREFERENCE Procedure Parameters*

Parameter	Description
<code>srcPath</code>	File system entry with which to link
<code>dstPath</code>	Path of the new link element to be created
<code>properties</code>	One or more properties and their values to be set, returned, or both, depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
<code>prop_flags</code>	Determines which properties are set, returned. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
<code>store_name</code>	Name of store
<code>principal</code>	File system user for whom the access check is made

## DECODEFEATURES Function

Given a feature bit set integer value, this function returns a `FEATURES_T` table of the feature bits as `FEATURE_T` records.

### Syntax

```
DBMS_DBFS_CONTENT.DECODEFEATURES (  
    featureSet      IN      INTEGER)  
    RETURN FEATURES_T DETERMINISTIC PIPELINED;
```

### Parameters

**Table 47–25** *DECODEFEATURES Function Parameters*

Parameter	Description
featureSet	Feature set

### Return Values

[FEATURES\\_T Table Type](#)



## DELETECONTENT Procedure

This procedure deletes the file specified by the given contentID.

### Syntax

```
DBMS_DBFS_CONTENT.DELETECONTENT (
  store_name    IN    VARCHAR2    DEFAULT NULL,
  contentID    IN    RAW,
  filter        IN    VARCHAR2    DEFAULT NULL,
  soft_delete   IN    BOOLEAN     DEFAULT NULL,
  principal     IN    VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 47–26 DELETECONTENT Procedure Parameters**

Parameter	Description
store_name	Name of store
contentID	Unique identifier for the file to be deleted
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see " <a href="#">Deletion Operations</a> " on page 47-20).
principal	File system user for whom the access check is made

## DELETEDIRECTORY Procedure

This procedure deletes a directory. If `recurse` is nonzero, it recursively deletes all elements of the directory. A filter, if supplied, determines which elements of the directory are deleted.

### Syntax

```
DBMS_DBFS_CONTENT.DELETEDIRECTORY (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2  DEFAULT NULL,
  soft_delete   IN      BOOLEAN    DEFAULT NULL,
  recurse       IN      BOOLEAN    DEFAULT FALSE,
  store_name    IN      VARCHAR2  DEFAULT NULL,
  principal     IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47–27 DELETEDIRECTORY Procedure Parameters**

Parameter	Description
<code>path</code>	Name of path to the directory
<code>filter</code>	A filter, if any, to be applied
<code>soft_delete</code>	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see <a href="#">"Deletion Operations"</a> on page 47-20).
<code>recurse</code>	If 0, do not execute recursively. Otherwise, recursively delete the directories and files below the given directory.
<code>store_name</code>	Name of store
<code>principal</code>	File system user for whom the access check is made

## DELETEFILE Procedure

This procedure deletes the specified file.

### Syntax

```
DBMS_DBFS_CONTENT.DELETEFILE (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2  DEFAULT NULL,
  soft_delete   IN      BOOLEAN    DEFAULT NULL,
  store_name    IN      VARCHAR2  DEFAULT NULL,
  principal     IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47–28** *DELETEFILE Procedure Parameters*

Parameter	Description
path	Name of path to the file
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see " <a href="#">Deletion Operations</a> " on page 47-20).
store_name	Name of store
principal	File system user for whom the access check is made

## FEATURENAME Function

Given a feature bit, this function returns a VARCHAR2 of that feature's name.

### Syntax

```
DBMS_DBFS_CONTENT.FEATURENAME (  
    featureBit          IN          INTEGER)  
RETURN VARCHAR2 DETERMINISTIC;
```

### Parameters

**Table 47–29** *FEATURENAME Function Parameters*

Parameter	Description
featureBit	Bit representation of the feature (see <a href="#">DBMS_DBFS_CONTENT Constants - Store Features</a> )

### Return Values

Name of the feature

## **FLUSHSTATS Function**

This procedure flushes DBMS\_DBFS\_CONTENT statistics to disk.

### **Syntax**

```
DBMS_DBFS_CONTENT.FLUSHSTATS;
```

## GETDEFAULTACL Procedure

This procedure returns the ACL parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTACL (  
    acl    OUT NOCOPY    VARCHAR2);
```

### Parameters

**Table 47–30** *GETDEFAULTACL Procedure Parameters*

Parameter	Description
acl	ACL for all new elements created (implicitly or explicitly) by the current operation

## GETDEFAULTASOF Procedure

This procedure returns the "as of" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTASOF (  
    asof    OUT NOCOPY    TIMESTAMP);
```

### Parameters

**Table 47–31** *GETDEFAULTASOF Procedure Parameters*

Parameter	Description
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

## GETTDEFAULTCONTEXT Procedure

This procedure returns the default context. The information contained in the context can be inserted explicitly by way of arguments to the various method calls, allowing for fine-grained control over individual operations.

### Syntax

```
DBMS_DBFS_CONTENT.GETTDEFAULTCONTEXT (  
  principal    OUT NOCOPY    VARCHAR2,  
  owner        OUT NOCOPY    VARCHAR2,  
  acl          OUT NOCOPY    VARCHAR2,  
  asof         OUT NOCOPY    TIMESTAMP);
```

### Parameters

**Table 47–32** *GETTDEFAULTCONTEXT Procedure Parameters*

Parameter	Description
principal	Agent (principal) invoking the current operation
owner	Owner for new elements created (implicitly or explicitly) by the current operation
acl	ACL for all new elements created (implicitly or explicitly) by the current operation
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes



## GETDEFAULTOWNER Procedure

This procedure returns the "owner" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTOWNER (  
    principal    IN    VARCHAR2);
```

### Parameters

**Table 47–33** *GETDEFAULTOWNER Procedure Parameters*

Parameter	Description
owner	Owner for new elements created (implicitly or explicitly) by the current operation

## GETDEFAULTPRINCIPAL Procedure

This procedure returns the "principal" parameter of the default context. This information contained can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTPRINCIPAL (  
    principal    OUT NOCOPY    VARCHAR2);
```

### Parameters

**Table 47–34** *GETDEFAULTPRINCIPAL Procedure Parameters*

Parameter	Description
principal	Agent (principal) invoking the current operation

## GETFEATURESBYMOUNT Function

This function returns features of a store by mount point.

### Syntax

```
DBMS_DBFS_CONTENT.GETFEATURESBYMOUNT (  
    store_mount    IN    VARCHAR2)  
RETURN INTEGER;
```

### Parameters

**Table 47–35** *GETFEATURESBYMOUNT Function Parameters*

Parameter	Description
store_mount	Mount point

### Return Values

A bit mask of supported features (see [FEATURES\\_T Table Type](#))

## GETFEATURESBYNAME Function

This function returns features of a store by store name.

### Syntax

```
DBMS_DBFS_CONTENT.GETFEATURESBYNAME (  
    store_name      IN      VARCHAR2)  
RETURN INTEGER;
```

### Parameters

**Table 47–36** *GETFEATURESBYNAME Function Parameters*

Parameter	Description
store_name	Name of store

### Return Values

A bit mask of supported features (see [FEATURES\\_T Table Type](#))

## GETFEATURESBYPATH Function

This function returns features of a store by path.

### Syntax

```
DBMS_DBFS_CONTENT.GETFEATURESBYPATH (  
    path          IN          PATH_T)  
RETURN INTEGER;
```

### Parameters

**Table 47–37** *GETFEATURESBYPATH Function Parameters*

Parameter	Description
path	PATH_T

### Return Values

A bit mask of supported features (see [FEATURES\\_T Table Type](#))

## GETPATH Procedures

This procedure returns existing path items (such as files and directories). This includes both data and metadata (properties).

The client can request (using `prop_flags`) that specific properties be returned. File path names can be read either by specifying a BLOB locator using the `prop_data` bitmask in `prop_flags` (see [DBMS\\_DBFS\\_CONTENT Constants - Property Access Flags](#)) or by passing one or more RAW buffers.

When `forUpdate` is 0, this procedure also accepts a valid `asof` timestamp parameter as part of `ctx` that can be used by stores to implement "as of" style flashback queries. Mutating versions of the GETPATH Procedures do not support these modes of operation.

### Syntax

```
DBMS_DBFS_CONTENT.GETPATH (
    path          IN          VARCHAR2,
    properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
    content       OUT NOCOPY  BLOB,
    item_type     OUT          INTEGER,
    prop_flags    IN          INTEGER          DEFAULT (PROP_STD +
                                                PROP_OPT +
                                                PROP_DATA) ,

    asof          IN          TIMESTAMP      DEFAULT NULL,
    forUpdate     IN          BOOLEAN        DEFAULT FALSE,
    deref         IN          BOOLEAN        DEFAULT FALSE,
    store_name    IN          VARCHAR2       DEFAULT NULL,
    principal     IN          VARCHAR2       DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.GETPATH (
    path          IN          VARCHAR2,
    properties    IN OUT NOCOPY PROPERTIES_T,
    content       OUT NOCOPY  BLOB,
    item_type     OUT          INTEGER,
    prop_flags    IN          INTEGER          DEFAULT (PROP_STD +
                                                PROP_OPT +
                                                PROP_DATA) ,

    asof          IN          TIMESTAMP      DEFAULT NULL,
    forUpdate     IN          BOOLEAN        DEFAULT FALSE,
    deref         IN          BOOLEAN        DEFAULT FALSE,
    store_name    IN          VARCHAR2       DEFAULT NULL,
    principal     IN          VARCHAR2       DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.GETPATH (
    path          IN          VARCHAR2,
    properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
    amount        IN OUT          NUMBER,
    offset        IN          NUMBER,
    buffers       OUT NOCOPY  RAW,
    prop_flags    IN          INTEGER          DEFAULT (PROP_STD +
                                                PROP_OPT +
                                                PROP_DATA) ,

    asof          IN          TIMESTAMP      DEFAULT NULL,
    store_name    IN          VARCHAR2       DEFAULT NULL,
    principal     IN          VARCHAR2       DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.GETPATH (
```

```

path          IN          VARCHAR2,
properties    IN OUT NOCOPY  PROPERTIES_T,
amount        IN OUT        NUMBER,
offset        IN          NUMBER,
buffers       OUT          NOCOPY RAW,
prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                PROP_OPT +
                                PROP_DATA),

asof          IN          TIMESTAMP   DEFAULT NULL,
store_name    IN          VARCHAR2    DEFAULT NULL,
principal     IN          VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
path          IN          VARCHAR2,
properties    IN OUT NOCOPY  DBMS_DBFS_CONTENT_PROPERTIES_T,
amount        IN OUT        NUMBER,
offset        IN          NUMBER,
buffers       OUT          NOCOPY  DBMS_DBFS_CONTENT_RAW_T,
prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                PROP_OPT +
                                PROP_DATA),

asof          IN          TIMESTAMP   DEFAULT NULL,
store_name    IN          VARCHAR2    DEFAULT NULL,
principal     IN          VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
path          IN          VARCHAR2,
properties    IN OUT NOCOPY  PROPERTIES_T,
amount        IN OUT        NUMBER,
offset        IN          NUMBER,
buffers       OUT          NOCOPY  DBMS_DBFS_CONTENT_RAW_T,
prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                PROP_OPT +
                                PROP_DATA),

asof          IN          TIMESTAMP   DEFAULT NULL,
store_name    IN          VARCHAR2    DEFAULT NULL,
principal     IN          VARCHAR2    DEFAULT NULL);

```

## Parameters

**Table 47–38** *GETPATH Procedure Parameters*

Parameter	Description
path	Name of path to path items
properties	One or more properties and their values to be returned depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
amount	On input, number of bytes to be read. On output, number of bytes read
offset	Byte offset from which to begin reading
buffer	Buffer to which to write
buffers	Buffers to which to write

**Table 47-38 (Cont.) GETPATH Procedure Parameters**

Parameter	Description
<code>prop_flags</code>	Determines which properties are set, returned, or both. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
<code>asof</code>	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
<code>forUpdate</code>	Specifies that a lock should be taken to signify exclusive write access to the path item
<code>deref</code>	If nonzero, attempts to resolve the given path item to actual data provided it is a reference
<code>store_name</code>	Name of store
<code>principal</code>	Agent (principal) invoking the current operation



## GETPATHBYMOUNTID Function

If the underlying GUID is found in the underlying store, this function returns the full absolute path name.

### Syntax

```
DBMS_DBFS_CONTENT.GETPATHBYMOUNTID (  
    store_mount      IN      VARCHAR2,  
    guid             IN      INTEGER)  
RETURN VARCHAR2;
```

### Parameters

**Table 47–39** *GETPATHBYMOUNTID Function Parameters*

Parameter	Description
store_mount	Mount point in which the path item with guid resides
guid	Unique ID for the path item

### Usage Notes

If the GUID is unknown, a NULL value is returned. Clients are expected to handle this as appropriate.

### Return Values

Path of the path item represented by GUID in store\_mount

## GETPATHBYSTOREID Function

If the underlying GUID is found in the underlying store, this function returns the store-qualified path name.

### Syntax

```
DBMS_DBFS_CONTENT.GETPATHBYSTOREID (  
    store_name      IN      VARCHAR2,  
    guid            IN      INTEGER)  
RETURN VARCHAR2;
```

### Parameters

**Table 47–40** *GETPATHBYSTOREID Function Parameters*

Parameter	Description
store_name	Name of store
guid	Unique ID representing the desired path item

### Usage Notes

If the GUID is unknown, a NULL value is returned. Clients are expected to handle this as appropriate.

### Return Values

Store-qualified path name represented by the GUID

## GETPATHNOWAIT Procedures

This procedure implies that the operation is for an update, and, if implemented (see `FEATURE_NOWAIT` in [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)), allows providers to return an exception (ORA-00054) rather than wait for row locks.

### Syntax

```
DBMS_DBFS_CONTENT.GETPATHNOWAIT (
    path          IN          VARCHAR2,
    properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
    content       OUT NOCOPY  BLOB,
    item_type     OUT          INTEGER,
    prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                                PROP_OPT +
                                                PROP_DATA),
    deref         IN          BOOLEAN     DEFAULT FALSE,
    store_name    IN          VARCHAR2    DEFAULT NULL,
    principal     IN          VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.GETPATHNOWAIT (
    path          IN          VARCHAR2,
    properties    IN OUT NOCOPY PROPERTIES_T,
    content       OUT NOCOPY  BLOB,
    item_type     OUT          INTEGER,
    prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                                PROP_OPT +
                                                PROP_DATA),
    deref         IN          BOOLEAN     DEFAULT FALSE,
    store_name    IN          VARCHAR2    DEFAULT NULL,
    principal     IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 47-41 GETPATHNOWAIT Procedure Parameters**

Parameter	Description
<code>path</code>	Name of path to path items
<code>properties</code>	One or more properties and their values to be returned depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
<code>content</code>	BLOB holding data which populates the file (optional)
<code>item_type</code>	Type of the path item specified (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
<code>prop_flags</code>	Determines which properties are returned. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
<code>asof</code>	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
<code>deref</code>	If nonzero, attempts to resolve the given path item to actual data provided it is a reference
<code>store_name</code>	Name of store

**Table 47–41 (Cont.) GETPATHNOWAIT Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>principal</code>	Agent (principal) invoking the current operation

## GETSTOREBYMOUNT Function

This function returns a store by way of its name.

### Syntax

```
DBMS_DBFS_CONTENT.GETSTOREBYMOUNT (  
    store_mount      IN      VARCHAR2)  
    RETURN STORE_T;
```

### Parameters

**Table 47–42** *GETSTOREBYMOUNT Function Parameters*

Parameter	Description
store_mount	Name of store

### Return Values

[STORE\\_T Record Type](#)

## GETSTOREBYNAME Function

This function returns a store by way of its name.

### Syntax

```
DBMS_DBFS_CONTENT.GETSTOREBYNAME (  
    store_name      IN      VARCHAR2)  
RETURN STORE_T;
```

### Parameters

**Table 47–43** *GETSTOREBYNAME Function Parameters*

Parameter	Description
store_name	Name of store

### Return Values

[STORE\\_T Record Type](#)

## GETSTOREBYPATH Function

This function returns a store by way of its path.

### Syntax

```
DBMS_DBFS_CONTENT.GETSTOREBYPATH (  
    path      IN      PATH_T)  
RETURN STORE_T;
```

### Parameters

**Table 47–44** *GETSTOREBYPATH Function Parameters*

Parameter	Description
path	PATH_T s

### Return Values

[STORE\\_T Record Type](#)

## GETSTATS Procedure

This procedure returns information about DBMS\_DBFS\_CONTENT statistics collection.

### Syntax

```
DBMS_DBFS_CONTENT.GETSTATS (  
    enabled          OUT   BOOLEAN,  
    flush_time      OUT   INTEGER,  
    flush_count     OUT   INTEGER);
```

### Parameters

**Table 47–45** GETSTATS Procedure Parameters

Parameter	Description
enabled	Whether statistics collection is enabled
flush_time	How often to flush the statistics to disk in centiseconds
flush_count	Number of operations to allow between statistics flushes



## GETTRACE Function

This function returns whether DBMS\_DBFS\_CONTENT tracing is turned on or not.

### Syntax

```
DBMS_DBFS_CONTENT.GETTRACE  
RETURN INTEGER.
```

### Return Values

Returns 0 if tracing is off, non-zero if tracing is on.

## GETVERSION Function

This function marks each version of the DBMS\_DBFS\_CONTENT interface.

### Syntax

```
DBMS_DBFS_CONTENT.GETVERSION (  
    RETURN VARCHAR2;
```

### Return Values

A string enumerating the version of the DBMS\_DBFS\_CONTENT interface in standard naming convention: string: *a.b.c* corresponding to *major*, *minor*, and *patch* components.

## LIST Function

This function lists the path items in the specified path meeting the specified filter and other criteria.

### Syntax

```
DBMS_DBFS_CONTENT.LIST (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2  DEFAULT NULL,
  recurse       IN      INTEGER    DEFAULT 0,
  asof          IN      TIMESTAMP  DEFAULT NULL,
  store_name    IN      VARCHAR2  DEFAULT NULL,
  principal     IN      VARCHAR2  DEFAULT NULL)
RETURN DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;
```

### Parameters

**Table 47–46** LIST Function Parameters

Parameter	Description
path	Name of path to directories
filter	A filter, if any, to be applied
recurse	If 0, do not execute recursively. Otherwise, recursively list the contents of directories and files below the given directory.
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
store_name	Name of respository
principal	Agent (principal) invoking the current operation

### Return Values

[DBMS\\_DBFS\\_CONTENT\\_LIST\\_ITEMS\\_T Table Type](#)

## LISTALLPROPERTIES Function

This function returns a table of all properties for all path items in all mounts

### Syntax

```
DBMS_DBFS_CONTENT.LISTALLPROPERTIES  
RETURN PROP_ITEMS_T PIPELINED;
```

### Return Values

[PROP\\_ITEMS\\_T Table Type](#)

## LISTALLCONTENT Function

This function lists all path items in all mounts.

### Syntax

```
DBMS_DBFS_CONTENT.LISTALLCONTENT  
RETURN PATH_ITEMS_T PIPELINED;
```

### Return Values

[PATH\\_ITEMS\\_T Table Type](#)

## LISTMOUNTS Function

This function lists all available mount points, their backing stores, and the store features.

### Syntax

```
DBMS_DBFS_CONTENT.LISTMOUNTS  
  RETURN MOUNTS_T PIPELINED;
```

### Return Values

[MOUNTS\\_T Table Type](#)

### Usage Notes

A single mount results in a single returned row, with its `store_mount` field of the returned records set to NULL.

## LISTSTORES Function

This function lists all available stores and their features.

### Syntax

```
DBMS_DBFS_CONTENT.LISTSTORES  
  RETURN STORES_T PIPELINED;
```

### Return Values

[STORES\\_T Table Type](#)

### Usage Notes

The `store_mount` field of the returned records is set to `NULL` (since mount-points are separate from stores themselves).

## LOCKPATH Procedure

This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

### Syntax

```
DBMS_DBFS_CONTENT.LOCKPATH (  
  path          IN      VARCHAR2,  
  lock_type     IN      INTEGER      DEFAULT LOCK_READ_ONLY,  
  lock_data     IN      VARCHAR2     DEFAULT NULL,  
  store_name    IN      VARCHAR2     DEFAULT NULL,  
  principal     IN      VARCHAR2     DEFAULT NULL);
```

### Parameters

**Table 47-47** LOCKPATH Procedure Parameters

Parameter	Description
path	Name of path to file items
lock_type	One of the available lock types (see <a href="#">DBMS_DBFS_CONTENT Constants - Lock Types</a> )
lock_data	Optional user data to be associated with the lock
store_name	Name of store
principal	Agent (principal) invoking the current operation



## MOUNTSTORE Procedure

This procedure mounts a previously registered store and binds it to the mount point.

### Syntax

```
DBMS_DBFS_CONTENT.MOUNTSTORE (
  store_mount   in   VARCHAR2   DEFAULT NULL,
  singleton     in   BOOLEAN    DEFAULT FALSE,
  principal     in   VARCHAR2   DEFAULT NULL,
  owner         in   VARCHAR2   DEFAULT NULL,
  acl           in   VARCHAR2   DEFAULT NULL,
  asof         in   TIMESTAMP   DEFAULT NULL,
  read_only    in   BOOLEAN    DEFAULT FALSE);
```

### Parameters

**Table 47–48 MOUNTSTORE Procedure Parameters**

Parameter	Description
store_mount	Path name to use to mount this store
singleton	Whether the mount is a single backend store on the system
principal	Agent (principal) invoking the current operation
owner	Owner for new elements created (implicitly or explicitly) by the current operation
acl	ACL for all new elements created (implicitly or explicitly) by the current operation
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
read_only	Whether the mount is read-only

### Usage Notes

- Once mounted, accesses to path names of the form */store\_mount/xyz...* are redirected to *store\_name* and its content provider.
- Store mount points must be unique, and a syntactically valid path name component (specifically, a NAME\_T with no embedded /).
- If a mount point is not specified (NULL), the invoked subprogram attempts to use the store name itself as the mount point name (subject to the uniqueness and syntactic constraints).
- A special empty mount point is available for single stores, specifically a scenario where the content interface manages a single backend store - in such cases, the client can directly deal with full path names of the form */xyz...* since there is no ambiguity in how to redirect these accesses.
- Singleton mount points are indicated by the "singleton" boolean argument, and the *store\_mount* argument is ignored.
- The same store can be mounted multiple times, obviously at different mount points.
- Mount properties can be used to specify the execution environment, specifically, the default values of the principal, owner, ACL, and *asof* for a particular mount

point. Mount properties can also be used to specify a read-only mount. If a flashback mount is specified (through `asof`), it implies a read-only mount.

## NORMALIZEPATH Functions

This function converts a store-specific or full-absolute path name into normalized form:

- verifies that the path name is absolute, and so starts with "/"
- collapses multiple consecutive "/" into a single "/"
- strips trailing "/"
- breaks up a store-specific normalized path name into 2 components - parent pathname, trailing component name
- breaks up a full-absolute normalized path name into 3 components - store name, parent pathname, trailing component name

### Syntax

```
DBMS_DBFS_CONTENT.NORMALIZEPATH (
  path          IN          VARCHAR2,
  parent        OUT NOCOPY  VARCHAR2,
  tpath         OUT NOCOPY  VARCHAR2)
RETURN VARCHAR2;
```

```
DBMS_DBFS_CONTENT.NORMALIZEPATH (
  path          IN          VARCHAR2,
  store_name    OUT NOCOPY  VARCHAR2,
  parent        OUT NOCOPY  VARCHAR2,
  tpath         OUT NOCOPY  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 47–49** *NORMALIZEPATH Function Parameters*

Parameter	Description
path	Name of path to file items
store_name	Name of store
parent	Parent path name
tpath	Name of trailing path item

### Return Values

The completely normalized store-specific or full-absolute path name

## PROPANY Functions

This function provides constructors that take one of a variety of types and return a `PROPERTY_T`.

### Syntax

```
DBMS_DBFS_CONTENT.PROPANY (  
    val      IN      NUMBER)  
RETURN PROPERTY_T;
```

```
DBMS_DBFS_CONTENT.PROPANY (  
    val      IN      VARCHAR2)  
RETURN PROPERTY_T;
```

```
DBMS_DBFS_CONTENT.PROPANY (  
    val      IN      TIMESTAMP)  
RETURN PROPERTY_T;
```

```
DBMS_DBFS_CONTENT.PROPANY (  
    val      IN      RAW)  
RETURN PROPERTY_T;
```

### Parameters

**Table 47–50** *PROPANY Function Parameters*

Parameter	Description
val	Value

### Return Values

[PROPERTY\\_T Record Type](#)

## PROPERTIESH2T Function

This function converts a PROPERTY\_T hash to a DBMS\_DBFS\_CONTENT\_PROPERTIES\_T table.

### Syntax

```
DBMS_DBFS_CONTENT.PROPERTIESH2T (  
    pprops      IN      PROPERTIES_T)  
RETURN DBMS_DBFS_CONTENT_PROPERTIES_T;
```

### Parameters

**Table 47–51** *PROPERTIESH2T Function Parameters*

Parameter	Description
sprops	A PROPERTIES_T hash

### Return Values

[DBMS\\_DBFS\\_CONTENT\\_PROPERTIES\\_T Table Type](#)

## PROPERTIEST2H Function

This function converts a DBMS\_DBFS\_CONTENT\_PROPERTIES\_T table to a PROPERTY\_T hash.

### Syntax

```
DBMS_DBFS_CONTENT.PROPERTIEST2H (
    sprops      IN      DBMS_DBFS_CONTENT_PROPERTIES_T)
RETURN properties_t;
```

### Parameters

**Table 47–52** *PROPERTIEST2H Function Parameters*

Parameter	Description
sprops	A DBMS_DBFS_CONTENT_PROPERTIES_T table

### Return Values

[PROPERTIES\\_T Table Type](#)

## PROPNUMBER Function

This function is a constructor that takes a number and returns a `PROPERTY_T`.

### Syntax

```
DBMS_DBFS_CONTENT.PROPNUMBER (  
    val      IN      NUMBER)  
RETURN PROPERTY_T;
```

### Parameters

**Table 47–53** *PROPNUMBER Function Parameters*

Parameter	Description
val	Value

### Return Values

[PROPERTY\\_T Record Type](#)

## PROPRAW Function

This function is a constructor that takes a RAW and returns a PROPERTY\_T.

### Syntax

```
DBMS_DBFS_CONTENT.PROPRW (
    val      IN      RAW)
RETURN PROPERTY_T;
```

### Parameters

**Table 47–54** PROPRAW Function Parameters

Parameter	Description
val	Value

### Return Values

[PROPERTY\\_T Record Type](#)



## PROPTIMESTAMP Function

This function is a constructor that takes a `TIMESTAMP` and returns a `PROPERTY_T`.

### Syntax

```
DBMS_DBFS_CONTENT.PROPTIMESTAMP (  
    val      IN      TIMESTAMP)  
RETURN PROPERTY_T;
```

### Parameters

**Table 47–55** *PROPTIMESTAMP Function Parameters*

Parameter	Description
val	Value

### Return Values

[PROPERTY\\_T Record Type](#)

## PROPVARCHAR2 Function

This function is a constructor that takes a VARCHAR2 and returns a PROPERTY\_T.

### Syntax

```
DBMS_DBFS_CONTENT.PROPVARCHAR2 (  
    val      IN      VARCHAR2)  
RETURN PROPERTY_T;
```

### Parameters

**Table 47–56** PROPNUMBER Function Parameters

Parameter	Description
val	Value

### Return Values

[PROPERTY\\_T Record Type](#)

## PURGEALL Procedure

This procedure purges all soft-deleted entries matching the path and optional filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT.PURGEALL (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2  DEFAULT NULL,
  store_name    IN      VARCHAR2  DEFAULT NULL,
  principal     IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47-57** *PURGEALL Procedure Parameters*

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied based on specified criteria
store_name	Name of store
principal	Agent (principal) invoking the current operation

## PURGEPATH Procedure

This procedure purges any soft-deleted versions of the given path item.

### Syntax

```
DBMS_DBFS_CONTENT.PURGEPATH (  
    path          IN          VARCHAR2,  
    filter        IN          VARCHAR2  DEFAULT NULL,  
    store_name    IN          VARCHAR2  DEFAULT NULL,  
    principal     IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47–58** *PURGEPATH Procedure Parameters*

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied
store_name	Name of store
principal	Agent (principal) invoking the current operation

## PUTPATH Procedures

This procedure creates a new path item.

### Syntax

```
DBMS_DBFS_CONTENT.PUTPATH (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  content       IN OUT NOCOPY BLOB,
  item_type    OUT          INTEGER,
  prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                     PROP_OPT +
                                     PROP_DATA),
  store_name    IN          VARCHAR2    DEFAULT NULL,
  principal     IN          VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.PUTPATH (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY PROPERTIES_T,
  content       IN OUT NOCOPY BLOB,
  item_type    OUT          INTEGER,
  prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                     PROP_OPT +
                                     PROP_DATA),
  store_name    IN          VARCHAR2    DEFAULT NULL,
  principal     IN          VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.PUTPATH (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  amount        IN          NUMBER,
  offset        IN          NUMBER,
  buffer        IN          RAW,
  prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                     PROP_OPT),
  store_name    IN          VARCHAR2    DEFAULT NULL,
  principal     IN          VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.PUTPATH (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY PROPERTIES_T,
  amount        IN          NUMBER,
  offset        IN          NUMBER,
  buffer        IN          RAW,
  prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                     PROP_OPT),
  store_name    IN          VARCHAR2    DEFAULT NULL,
  principal     IN          VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.PUTPATH (
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  written       OUT          NUMBER,
  offset        IN          NUMBER,
  buffers       IN          DBMS_DBFS_CONTENT_RAW_T,
  prop_flags    IN          INTEGER      DEFAULT (PROP_STD +
                                     PROP_OPT),
  store_name    IN          VARCHAR2    DEFAULT NULL,
```

```

principal IN          VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.PUTPATH (
  path      IN          VARCHAR2,
  properties IN OUT NOCOPY PROPERTIES_T,
  written   OUT         NUMBER,
  offset    IN          NUMBER,
  buffers   IN          DBMS_DBFS_CONTENT_RAW_T,
  prop_flags IN        INTEGER     DEFAULT (PROP_STD +
                                         PROP_OPT),

  store_name IN        VARCHAR2    DEFAULT NULL,
  principal IN        VARCHAR2    DEFAULT NULL);

```

## Parameters

**Table 47–59** *PUTPATH Procedure Parameters*

Parameter	Description
path	Name of path to file items
properties	One or more properties and their values to be set depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
amount	Number of bytes to be read
offset	Byte offset from which to begin reading
buffer	Buffer to which to write
buffers	Buffers to which to write
prop_flags	Determines which properties are set. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
store_name	Name of store
principal	Agent (principal) invoking the current operation

## REGISTERSTORE Procedure

This procedure registers a new store backed by a provider that uses a store service provider (conforming to the DBMS\_DBFS\_CONTENT\_SPI package signature). This method is to be used primarily by service providers after they have created a new store.

### Syntax

```
DBMS_DBFS_CONTENT.REGISTERSTORE (  
    store_name          IN      VARCHAR2,  
    provider_name      IN      VARCHAR2,  
    provider_package   IN      VARCHAR2);
```

### Parameters

**Table 47–60 REGISTERSTORE Procedure Parameters**

Parameter	Description
store_name	Name of store, must be unique
provider_name	Name of provider
provider_package	Store service provider

## RENAMEPATH Procedures

This procedure renames or moves a path. This operation can be performed across directory hierarchies and mount-points as long as it is within the same store.

---

**Note:** See ["Rename and Move Operations"](#) on page 47-21

---

### Syntax

```
DBMS_DBFS_CONTENT.RENAMEPATH (
    oldPath      IN          VARCHAR2,
    newPath      IN          VARCHAR2,
    properties   IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
    store_name   IN          VARCHAR2  DEFAULT NULL,
    principal    IN          VARCHAR2  DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.RENAMEPATH (
    oldPath      IN          VARCHAR2,
    newPath      IN          VARCHAR2,
    properties   IN OUT NOCOPY PROPERTIES_T,
    store_name   IN          VARCHAR2  DEFAULT NULL,
    principal    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47–61** *RENAMEPATH Procedure Parameters*

Parameter	Description
oldPath	Name of path prior to renaming
newPath	Name of path after renaming
properties	One or more properties and their values to be set depending on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
store_name	Name of store, must be unique
principal	Agent (principal) invoking the current operation



## RESTOREALL Procedure

This procedure restores all soft-deleted path items meeting the path and optional filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT.RESTOREALL (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2  DEFAULT NULL,
  store_name    IN      VARCHAR2  DEFAULT NULL,
  principal     IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47-62** *RESTOREALL Procedure Parameters*

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied
store_name	Name of store
principal	Agent (principal) invoking the current operation

## RESTOREPATH Procedure

This procedure restores all soft-deleted path items that match the given path and optional filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT.RESTOREPATH (  
  path          IN          VARCHAR2,  
  filter        IN          VARCHAR2  DEFAULT NULL,  
  store_name    IN          VARCHAR2  DEFAULT NULL,  
  principal     IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47–63** *RESTOREPATH Procedure Parameters*

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied
store_name	Name of store
principal	Agent (principal) invoking the current operation

## SEARCH Function

This function...

### Syntax

```
DBMS_DBFS_CONTENT.SEARCH (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2  DEFAULT NULL,
  recurse       IN      INTEGER    DEFAULT 0,
  asof          IN      TIMESTAMP  DEFAULT NULL,
  store_name    IN      VARCHAR2  DEFAULT NULL,
  principal     IN      VARCHAR2  DEFAULT NULL)
RETURN DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;
```

### Parameters

**Table 47–64 LIST Function Parameters**

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied
recurse	If 0, do not execute recursively. Otherwise, recursively search the contents of directories and files below the given directory.
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
store_name	Name of store
principal	Agent (principal) invoking the current operation

### Return Values

DBMS\_DBFS\_CONTENT\_LIST\_ITEMS\_T

## SETDEFAULTACL Procedure

This procedure sets the ACL parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTACL (  
    acl    IN    VARCHAR2);
```

### Parameters

**Table 47–65** *SETDEFAULTACL Procedure Parameters*

Parameter	Description
acl	ACL for all new elements created (implicitly or explicitly) by the current operation

### Usage Notes

- NULL by default, this parameter be can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

## SETDEFAULTASOF Procedure

This procedure sets the "as of" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTASOF (
  asof IN TIMESTAMP);
```

### Parameters

**Table 47–66 SETDEFAULTASOF Procedure Parameters**

Parameter	Description
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

### Usage Notes

- NULL by default, this parameter can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

## SETDEFAULTCONTEXT Procedure

This procedure sets the default context. The information contained in the context can be inserted explicitly by way of arguments to the various method calls, allowing for fine-grained control over individual operations.

### Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTCONTEXT (
  principal IN VARCHAR2,
  owner     IN VARCHAR2,
  acl       IN VARCHAR2,
  asof      IN  TIMESTAMP);
```

### Parameters

**Table 47-67** *SETDEFAULTCONTEXT Procedure Parameters*

Parameter	Description
principal	Agent (principal) invoking the current operation
owner	Owner for new elements created (implicitly or explicitly) by the current operation
acl	ACL for all new elements created (implicitly or explicitly) by the current operation
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

### Usage Notes

- All of the context parameters are NULL by default, and be can be cleared by setting them to NULL.
- The context parameters, once set, remain as defaults for the duration of the session, and are inherited by all operations for which the defaults are not explicitly overridden.

## SETDEFAULTOWNER Procedure

This procedure sets the "owner" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTOWNER (
    principal    IN    VARCHAR2);
```

### Parameters

**Table 47–68** *SETDEFAULTOWNER Procedure Parameters*

Parameter	Description
owner	Owner for new elements created (implicitly or explicitly) by the current operation

### Usage Notes

- NULL by default, this parameter can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

## SETDEFAULTPRINCIPAL Procedure

This procedure sets the "principal" parameter of the default context. This information contained can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

### Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTPRINCIPAL (  
    principal    IN    VARCHAR2);
```

### Parameters

**Table 47–69** *SETDEFAULTPRINCIPAL Procedure Parameters*

Parameter	Description
principal	Agent (principal) invoking the current operation

### Usage Notes

- NULL by default, this parameter be can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.



## SETPATH Procedures

This procedure assigns a path name to a path item represented by contentID.

Stores and their providers that support contentID-based access and lazy path name binding also support the `SETPATH` Procedure that associates an existing contentID with a new path.

---



---

**Note:** See ["Rename and Move Operations"](#) on page 47-21

---



---

### Syntax

```
DBMS_DBFS_CONTENT.SETPATH (
  store_name  IN          VARCHAR2,
  contentID   IN          RAW,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_DBFS_CONTENT_PROPERTIES_T,
  principal   IN          VARCHAR2  DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.SETPATH (
  store_name  IN          VARCHAR2,
  contentID   IN          RAW,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY PROPERTIES_T,
  principal   IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 47–70** *SETPATH Procedure Parameters*

Parameter	Description
store_name	Name of the store
contentID	Unique identifier for the item to be associated
path	Name of path to path item
properties	One or more properties and their values to be set depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
principal	Agent (principal) invoking the current operation

## SETSTATS Procedure

This procedure enables and disables statistics collection. The client can optionally control the flush settings by specifying non-NULL values for the time, count or both parameters.

### Syntax

```
DBMS_DBFS_CONTENT.SETSTATS (  
  enable          IN    BOOLEAN,  
  flush_time      IN    INTEGER,  
  flush_count     IN    INTEGER);
```

### Parameters

**Table 47-71** SETSTATS Procedure Parameters

Parameter	Description
enable	If TRUE, enable statistics collection. If FALSE, disable statistics collection.
flush_time	How often to flush the statistics to disk in centiseconds
flush_count	Number of operations to allow between statistics flushes

### Usage Notes

The SETSTATS Procedure buffers statistics in-memory for a maximum of `flush_time` centiseconds or a maximum of `flush_count` operations (whichever limit is reached first), or both, at which time the buffers are implicitly flushed to disk.

## SETTRACE Procedure

This procedure sets the DBMS\_DBFS\_CONTENT tracing severity to the given level, 0 being "off".

### Syntax

```
DBMS_DBFS_CONTENT.SETTRACE  
    trclvl      IN      INTEGER);
```

### Parameters

**Table 47–72** SETTRACE Procedure Parameters

Parameter	Description
trclvl	Level of the tracing, higher values implying more tracing

## SPACEUSAGE Procedure

This procedure queries file system space usage statistics. Providers are expected to support this subprogram for their stores (and to make a best effort determination of space usage, especially if the store consists of multiple tables, indexes, LOBs, and so on).

### Syntax

```
DBMS_DBFS_CONTENT.SPACEUSAGE (
  path          IN          VARCHAR2,
  blksize      OUT         INTEGER,
  tbytes       OUT         INTEGER,
  fbytes       OUT         INTEGER,
  nfile        OUT         INTEGER,
  ndir         OUT         INTEGER,
  nlink        OUT         INTEGER,
  nref         OUT         INTEGER,
  store_name   IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 47–73 SPACEUSAGE Procedure Parameters**

Parameter	Description
path	Name of path to file items
blksize	Natural tablespace blocksize that holds the store. If multiple tablespaces with different blocksizes are used, any valid blocksize is acceptable.
tbytes	Total size of the store in bytes computed over all segments that comprise the store
fbytes	Free or unused size of the store in bytes computed over all segments that comprise the store
nfile	Number of currently available files in the store
ndir	Number of currently available directories in the store
nlink	Number of currently available links in the store
nref	Number of currently available references in the store
store_name	Name of store

### Usage Notes

- A space usage query on the top-level root directory returns a combined summary of the space usage of all available distinct stores under it (if the same store is mounted multiple times, is still counted only once).
- Since database objects are dynamically expandable, it is not easy to estimate the division between "free" space and "used" space.

## TRACE Procedure

This procedure outputs tracing to the current foreground trace file.

### Syntax

```
DBMS_DBFS_CONTENT.TRACE
  sev          IN          INTEGER,
  msg0         IN          VARCHAR2,
  msg1         IN          VARCHAR   DEFAULT '',
  msg2         IN          VARCHAR   DEFAULT '',
  msg3         IN          VARCHAR   DEFAULT '',
  msg4         IN          VARCHAR   DEFAULT '',
  msg5         IN          VARCHAR   DEFAULT '',
  msg6         IN          VARCHAR   DEFAULT '',
  msg7         IN          VARCHAR   DEFAULT '',
  msg8         IN          VARCHAR   DEFAULT '',
  msg9         IN          VARCHAR   DEFAULT '',
  msg10        IN          VARCHAR   DEFAULT '' );
```

### Parameters

**Table 47–74** TRACE Procedure Parameters

Parameter	Description
sev	Severity at which trace message is output
msg*	One or more message strings to be output. If more than one message is specified, all are output.

### Usage Notes

- Trace information is written to the foreground trace file, with varying levels of detail as specified by the trace level arguments.
- The global trace level consists of 2 components: "severity" and "detail". These can be thought of as additive bitmasks.

The "severity" allows the separation of top level as compared to low-level tracing of different components, and allows the amount of tracing to be increased as needed. There are no semantics associated with different levels, and users are free to set trace at any severity they choose, although a good rule of thumb would use severity "1" for top level API entry and exit traces, "2" for internal operations, and "3" or greater for very low-level traces.

The "detail" controls how much additional information: timestamps, short-stack, etc. is dumped along with each trace record.

## TRACEENABLED Function

This function determines if the current trace "severity" set by the [SETTRACE Procedure](#) is at least as high as the given trace level.

### Syntax

```
DBMS_DBFS_CONTENT.TRACEENABLED(  
    sev          IN          INTEGER)  
RETURN INTEGER;
```

### Parameters

**Table 47–75** *TRACEENABLED Procedure Parameters*

Parameter	Description
sev	Severity at which trace message is output

### Return Values

Returns 0 if the requested severity level is lower than the currently set trace severity level; 1 otherwise.

## UNLOCKPATH Procedure

This procedure unlocks path items that were previously locked with the [LOCKPATH Procedure](#).

### Syntax

```
DBMS_DBFS_CONTENT.UNLOCKPATH (  
    path          IN      VARCHAR2,  
    store_name    IN      VARCHAR2    DEFAULT NULL,  
    principal     IN      VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 47–76 UNLOCKPATH Procedure Parameters**

Parameter	Description
path	Name of path to file items
store_name	Name of store
principal	Agent (principal) invoking the current operation

## UNMOUNTSTORE Procedure

This procedure unmounts a registered store, either by name or by mount point.

### Syntax

```
DBMS_DBFS_CONTENT.UNMOUNTSTORE (  
    store_name      IN      VARCHAR2  DEFAULT NULL,  
    store_mount     IN      VARCHAR2  DEFAULT NULL,  
    ignore_unknown  IN      BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 47-77 UNMOUNTSTORE Procedure Parameters**

Parameter	Description
store_name	Name of store
store_mount	Location at which the store instance is mounted
ignore_unknown	If TRUE, attempts to unregister unknown stores will not raise an exception.

### Usage Notes

- Single stores can be unmounted only by store name (since they have no mount-points).
- Attempting to unmount a store by name unmounts all mount points associated with the store.
- Once unmounted all access to the store (or mount point) are not guaranteed to work.
- If the `ignore_unknown` argument is TRUE, attempts to unregister unknown stores or mounts do not raise an exception.



## UNREGISTERSTORE Procedure

This procedure unregisters a previously registered store (invalidating all mount points associated with it).

### Syntax

```
DBMS_DBFS_CONTENT.UNREGISTERSTORE (
  store_name      IN      VARCHAR2,
  ignore_unknown  IN      BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 47–78 UNREGISTERSTORE Procedure Parameters**

Parameter	Description
store_name	Name of store
ignore_unknown	

### Usage Notes

- Once unregistered all access to the store (and its mount points) are not guaranteed to work
- If the ignore\_unknown argument is TRUE, attempts to unregister unknown stores do not raise an exception.



---

---

## DBMS\_DBFS\_CONTENT\_SPI

The `DBMS_DBFS_CONTENT_SPI` package provides the Application Programming Interface (API) specification for [DBMS\\_DBFS\\_CONTENT](#) service providers. Application designers can create PL/SQL packages conforming to this API to extend `DBMS_CONTENT` to utilize custom service providers.

**See Also:**

- *Oracle Database SecureFiles and Large Objects Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_DBFS\\_CONTENT\\_SPI](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_DBFS\\_CONTENT\\_SPI Subprograms](#)

---

## Using DBMS\_DBFS\_CONTENT\_SPI

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)

## Overview

The `DBMS_DBFS_CONTENT_SPI` package describes an internal contract between the implementation of the `DBMS_DBFS_CONTENT` interface and individual service providers, and whichever package contains their code.

Since PL/SQL does not allow a compile-time, declarative type-conformation between package signatures, store providers should informally conform to the SPI, which is to say, they should implement the SPI by means of a package that contains all of the methods specified in package `DBMS_DBFS_CONTENT_SPI`, with the same method signatures and semantics.

Obviously, these provider packages can implement other methods and expose other interfaces, however, these interfaces are not to be used by the `DBMS_CONTENT` interface itself.

Since the provider SPI is merely a contract specification, there is no package body for `DBMS_DBFS_CONTENT_SPI`, and it is not possible to actually invoke any methods using this package.

The SPI references various elements (constants, types, exceptions) defined by the `DBMS_CONTENT` interface.

Additionally, there is an almost one-to-one correspondence between the client API exported by the `DBMS_CONTENT` interface and the provider interface that the `DBMS_CONTENT` interface itself expects to work against.

The main distinction in the method naming conventions is that all path name references are always store-qualified. That is, the notion of mount-points and full-absolute path names have been normalized and converted to store-qualified path names by the `DBMS_CONTENT` interface before it invokes any of the provider SPI methods.

Since the interconnection of the `DBMS_DBFS_CONTENT` interface and the provider SPI is a 1-to-many pluggable architecture, and the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

## Security Model

Implementations of the `DBMS_DBFS_CONTENT_SPI` package should be created as `AUTHID CURRENT_USER`.

## Operational Notes

- [Implementation](#)
- [Path Names](#)
- [Creation Operations](#)
- [Deletion Operations](#)
- [Get \(Retrieve\) and Put \(Insert\) Operations](#)
- [Rename and Move Operations](#)
- [Directory Navigation and Search](#)
- [Locking Operations](#)
- [Access Check Operation](#)

### Implementation

Since the interconnection of the [DBMS\\_DBFS\\_CONTENT](#) interface and the provider SPI is a 1-to-many pluggable architecture, the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

There are no explicit `INIT` or `FINI` methods to indicate when the `DBMS_DBFS_CONTENT` interface plugs or unplugs a particular provider SPI. Provider SPIs must be willing to auto-initialize themselves at any SPI entry-point.

All operations performed by a service provider are "stateless" in that they are complete operations unto themselves. If state is necessary to be maintained for some reason, then the state must be maintained in data structures such as auxiliary tables that can be queried as needed.

### Path Names

All path names used in the provider SPI are store-qualified in pair form (`store_name`, `pathname`) where the path name is rooted within the store namespace.

Stores and their providers that support contentID-based access (see `FEATURE_CONTENT_ID` in [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)) also support a form of addressing that is not based on path names. Content items are identified by an explicit store name, a `NULL` path name, and possibly a contentID specified as a parameter or by way of the `OPT_CONTENT_ID` (see [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)) property.

Not all operations are supported with contentID-based access, and applications should depend only on the simplest create or delete functionality being available.

### Creation Operations

The provider SPI must allow the DBFS content API to create directory, file, link, and reference elements subject to store feature support.

All of the creation subprograms require a valid path name, but note the special exemption for contentID-based access. Creation subprograms can optionally specify properties to be associated with the path name as it is created. It is also possible for clients to return item properties after the creation completes so that automatically generated properties, such as `STD_CREATION_TIME` (see [DBMS\\_DBFS\\_CONTENT Constants - Standard Properties](#)) are immediately available to clients. The exact set of properties fetched back is controlled by the various `PROP_XXX` bitmasks in the `prop_flags` parameter.

Links and references require an additional path name to associate with the primary path name.

File path names can optionally specify a BLOB value to use to initially populate the underlying file content (the provided BLOB may be any valid LOB). On creation, the underlying LOB is returned to the client, provided that `PROP_DATA` is specified in the `prop_flags` parameter.

Non-directory path names require that their parent directory be created first. Directory path names themselves can be recursively created with the path name hierarchy leading up to a directory created in one call.

Attempts to create paths that already exist is an error; the sole exception is path names that are "soft-deleted" (as discussed in the context of [Deletion Operations](#)). In these cases, the soft-deleted item is implicitly purged, and the new item creation is attempted.

Stores and their providers that support contentID-based access accept an explicit store name and a `NULL` path to create a new content element. The contentID generated for this element is available by means of the `OPT_CONTENT_ID` property (see [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)), contentID-based creation being automatically implied by `PROP_OPT` property in the `prop_flags` parameter).

The newly created element may also have an internally generated path name if `FEATURE_LAZY_PATH` property is not supported (see [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)) and this path is available by way of the `STD_CANONICAL_PATH` property (see [DBMS\\_DBFS\\_CONTENT Constants - Standard Properties](#)).

Only file elements are candidates for contentID-based access.

### Deletion Operations

The provider SPI must allow the DBFS content API to delete directory, file, link, and reference elements (subject to store feature support).

By default, the deletions are permanent, removing the successfully deleted items on transaction commit, but stores may also support "soft-delete" features. If requested by the client, soft-deleted items are retained by the store, although they are not typically visible in normal listings or searches.

Soft-deleted items can be restored, or explicitly purged.

Directory path names can be recursively deleted, with the path name hierarchy below a directory deleted in one call. Non-recursive deletions can be performed only on empty directories. Recursive soft-deletions apply the soft-delete to all of the items being deleted.

Individual path names, as well as all soft-deleted path names under a directory, can be restored or purged by means of the various restore and purge subprograms.

Providers that support filtering can use the provider filter to identify subsets of items to delete. This makes most sense for bulk operations such as the [DELETEDIRECTORY Procedure](#), [PURGEALL Procedure](#), and [RESTOREALL Procedure](#), but all of the deletion-related operations accept a `filter` argument.

Stores and their providers that support contentID-based access can also allow file items to be deleted by specifying their contentID.

### Get (Retrieve) and Put (Insert) Operations

Existing path items can be accessed (for query or for update) and modified using simple get and put subprograms. All path names allow their metadata (properties) to



be read and modified. On completion of the call, the client can request specific properties to be fetched by means of the `prop_flags` parameter.

File path names allow their data (content) to be read and modified. On completion of the call, the client can use the `PROP_DATA` bitmasks in the `prop_flags` parameter to request a new `BLOB` locator to continue data access.

Files can also be read or written without using `BLOB` locators by explicitly specifying logical offsets or buffer-amounts and a suitably sized buffer.

Update accesses must specify the `forUpdate` flag. Access to link path names can be implicitly and internally de-referenced by stores (subject to feature support) if the `deref` flag is specified, however, this may have undetermined outcomes since symbolic links are not always resolvable.

The read methods, such as the [GETPATH Procedures](#) where `forUpdate` is specified as 0, also accepts a valid `asof` timestamp in the `ctx` parameter that can be used by stores to implement "as of" style flashback queries. Mutating versions of the [GETPATH Procedures](#) and the [PUTPATH Procedures](#) methods do not support "as of" modes of operation.

The [GETPATHNOWAIT Procedure](#) implies that the operation is for an update, and, if implemented (see `FEATURE_NOWAIT` in [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)), this allows providers to return an exception (ORA-00054) rather than wait for row locks.

### Rename and Move Operations

Path names can be renamed or moved, possibly across directory hierarchies and mount-points, but within the same store.

Non-directory path names previously accessible by way of specifying the `oldPath` parameter are renamed as a single item subsequently accessible by specifying `newPath`, assuming that `newPath` does not already exist.

If `newPath` exists and is not a directory, the action of renaming implicitly deletes the existing item before renaming `oldPath`. If the `newPath` exists and is a directory, `oldPath` is moved into the target directory.

Directory path names previously accessible by way of `oldPath` are renamed by moving the directory and all of its children to `newPath` (if it does not already exist) or as children of `newPath` (if it exists and is a directory).

Stores and their providers that support `contentID`-based access and lazy path name binding also support the [SETPATH Procedure](#) that associates an existing `contentID` with a new "path".

### Directory Navigation and Search

The `DBMS_CONTENT` interface can list or search the contents of directory path names, with the option of doing so recursively into sub-directories, optionally seeing soft-deleted items, optionally using flashback "as of" a provided timestamp, and optionally filtering items in or out within the store based on list or search predicates.

### Locking Operations

Clients of the `DBMS_CONTENT` interface can apply user-level locks to any valid path name (subject to store feature support), associate the lock with user-data, and subsequently unlock these path names.

The status of locked items is available using various optional properties (note the previous discussion regarding `opt_lock`).

It is the responsibility of the store (assuming it supports user-defined lock checking) to ensure that lock and unlock operations are performed in a consistent manner.

**Access Check Operation**

This operation ascertains if a given path name (`store_name`, `path`, `pathtype`) can be manipulated by operation (see the various `DBMS_CONTENT.OP_XXX` opcodes in [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)) by the user acting on the store utilizing the `principal` parameter. This is a convenience function for the `DBMS_CONTENT` interface; a store that supports access control still internally performs these checks to guarantee security.

---

## Summary of DBMS\_DBFS\_CONTENT\_SPI Subprograms

**Table 48–1 DBMS\_DBFS\_CONTENT\_SPI Package Subprograms**

Subprogram	Description
<a href="#">CHECKACCESS Function</a> on page 48-11	Reports if the user ( <code>principal</code> ) can perform the specified operation on the given path
<a href="#">CREATEDIRECTORY Procedure</a> on page 48-12	Creates a directory
<a href="#">CREATEFILE Procedure</a> on page 48-13	Creates a file
<a href="#">CREATELINK Procedure</a> on page 48-14	Creates a physical link to an already existing file system element
<a href="#">CREATEREFERENCE Procedure</a> on page 48-15	Creates a new reference to the source file system element
<a href="#">DELETECONTENT Procedure</a> on page 48-16	Deletes the file specified by the given <code>contentID</code>
<a href="#">DELETEDIRECTORY Procedure</a> on page 48-17	Deletes a directory
<a href="#">DELETEFILE Procedure</a> on page 48-18	Deletes a file
<a href="#">GETFEATURES Function</a> on page 48-19	Returns the features of a store
<a href="#">GETPATH Procedures</a> on page 48-20	Returns existing path items (such as files and directories)
<a href="#">GETPATHBYSTOREID Function</a> on page 48-22	If the underlying <code>GUID</code> is found in the underlying store, returns the store-qualified path name
<a href="#">GETPATHNOWAIT Procedure</a> on page 48-23	Implies that the operation is for an update, and, if implemented, allows providers to return an exception ( <code>ORA-00054</code> ) rather than wait for row locks.
<a href="#">GETSTOREID Function</a> on page 48-24	Returns the ID of a store
<a href="#">GETVERSION Function</a> on page 48-25	Returns the version associated with a store
<a href="#">LIST Function</a> on page 48-26	Lists the contents of a directory path name
<a href="#">LOCKPATH Procedure</a> on page 48-27	Applies user-level locks to the given valid path name
<a href="#">PURGEALL Procedure</a> on page 48-28	Purges all soft-deleted entries matching the path and optional filter criteria
<a href="#">PURGEPATH Procedure</a> on page 48-29	Purges any soft-deleted versions of the given path item
<a href="#">PUTPATH Procedures</a> on page 48-30	Creates a new path item
<a href="#">RENAMEPATH Procedure</a> on page 48-32	Renames or moves a path
<a href="#">RESTOREALL Procedure</a> on page 48-33	Restores all soft-deleted path items meeting the path and filter criteria

**Table 48-1 (Cont.) DBMS\_DBFS\_CONTENT\_SPI Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">RESTOREPATH Procedure</a> on page 48-34	Restores all soft-deleted path items that match the given path and filter criteria
<a href="#">SEARCH Function</a> on page 48-35	Searches for path items matching the given path and filter criteria
<a href="#">SETPATH Procedure</a> on page 48-36	Assigns a path name to a path item represented by contentID
<a href="#">SPACEUSAGE Procedure</a> on page 48-37	Queries file system space usage statistics
<a href="#">UNLOCKPATH Procedure</a> on page 48-38	Unlocks path items that were previously locked with the <a href="#">LOCKPATH Procedure</a>

## CHECKACCESS Function

This function reports if the user (`principal`) can perform the specified operation on the given path. This enables verifying the validity of an operation without attempting to perform the operation. If `CHECKACCESS` returns 0, then the subprogram invoked to implement that operation should fail with an error.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.CHECKACCESS (
  store_name  IN    VARCHAR2  DEFAULT NULL,
  path        IN    VARCHAR2,
  pathtype    IN    INTEGER,
  operation   IN    VARCHAR2,
  principal   IN    VARCHAR2)
RETURN INTEGER;
```

### Parameters

**Table 48–2** *CHECKACCESS Procedure Parameters*

Parameter	Description
<code>store_name</code>	Name of store
<code>path</code>	Name of path to check for access
<code>pathtype</code>	Type of object <code>path</code> represents (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
<code>operation</code>	Operation to be checked (see <a href="#">DBMS_DBFS_CONTENT Constants - Optional Properties</a> )
<code>principal</code>	File system user for whom the access check is made

### Usage Notes

Whether or not the user invokes this function, a store that supports access control internally performs these checks to guarantee security.

## CREATEDIRECTORY Procedure

This procedure creates a directory.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATEDIRECTORY (
  store_name IN          VARCHAR2,
  path       IN          VARCHAR2,
  properties IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  prop_flags IN          INTEGER,
  recurse   IN          INTEGER,
  ctx       IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–3** *CREATEDIRECTORY Procedure Parameters*

Parameter	Description
store_name	Name of store
path	Name of path to the directory
properties	One or more properties and their values to be set, returned, or both, depending on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting PROP_SPC (see <a href="#">DBMS_DBFS_CONTENT Constants - Property Access Flags</a> ), and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
recurse	If 0, do not execute recursively; otherwise, recursively create the directories above the given directory
ctx	Context with which to create the directory (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## CREATEFILE Procedure

This procedure creates a file.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATEFILE (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  content     IN OUT NOCOPY BLOB,
  prop_flags  IN          INTEGER,
  ctx         IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–4 CREATEFILE Procedure Parameters**

Parameter	Description
store_name	Name of store
path	Name of path to the file
properties	One or more properties and their values to be set, returned or both depending, or both on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data with which to populate the file (optional)
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
ctx	Context with which to create the file (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## CREATELINK Procedure

This procedure creates a physical link to an already existing file system element (such as file or directory). The resulting entry shares the same metadata structures as the value of the `srcPath` parameter, and so is similar to incrementing a reference count on the file system element. This is analogous to a UNIX file system hard link.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATELINK (
  store_name      IN          VARCHAR2,
  srcPath         IN          VARCHAR2,
  dstPath        IN          VARCHAR2,
  properties     IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  prop_flags     IN          INTEGER,
  ctx            IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–5** *CREATELINK Procedure Parameters*

Parameter	Description
<code>store_name</code>	Name of store
<code>srcPath</code>	File system entry with which to link
<code>dstPath</code>	Path of the new link element to be created
<code>properties</code>	One or more properties and their values to be set, returned, or both, depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
<code>prop_flags</code>	Determines which properties are set, returned, or both. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
<code>ctx</code>	Context with which to create the link (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )



## CREATEREFERENCE Procedure

This procedure creates a new reference to the source file system element (such as a file, or directory). The resulting reference points to the source element but does not directly share metadata with the source element. This is analogous to a UNIX file system symbolic link.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATEREFERENCE (
  srcPath      IN          VARCHAR2,
  dstPath      IN          VARCHAR2,
  properties   IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  prop_flags   IN          INTEGER,
  store_name   IN          VARCHAR2,
  ctx          IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–6** *CREATEREFERENCE Procedure Parameters*

Parameter	Description
store_name	Name of store
srcPath	File system entry with which to link
dstPath	Path of the new link element to be created
properties	One or more properties and their values to be set, returned, or both, depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
prop_flags	Determines which properties are set, returned, or both. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
ctx	Context with which to create the reference (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## DELETECONTENT Procedure

This procedure deletes the file specified by the given contentID.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.DELETECONTENT (
  store_name      IN      VARCHAR2,
  contentID       IN      RAW,
  filter          IN      VARCHAR2,
  soft_delete     IN      INTEGER,
  ctx             IN      DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48-7 DELETECONTENT Procedure Parameters**

Parameter	Description
store_name	Name of store
contentID	Unique identifier for the file to be deleted
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see "Deletion Operations" on page 48-6).
ctx	Context with which to delete the file (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## DELETEDIRECTORY Procedure

This procedure deletes a directory. If `recurse` is nonzero, it recursively deletes all elements of the directory. A `filter`, if supplied, determines which elements of the directory are deleted.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.DELETEDIRECTORY (
  store_name  IN   VARCHAR2,
  path        IN   VARCHAR2,
  filter      IN   VARCHAR2,
  soft_delete IN   INTEGER,
  recurse     IN   INTEGER,
  ctx        IN   DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–8 DELETEDIRECTORY Procedure Parameters**

Parameter	Description
<code>store_name</code>	Name of store
<code>path</code>	Name of path to the directory
<code>filter</code>	A filter, if any, to be applied
<code>soft_delete</code>	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see <a href="#">"Deletion Operations"</a> on page 48-6).
<code>recurse</code>	If 0, do not execute recursively. Otherwise, recursively delete the directories and files below the given directory.
<code>ctx</code>	Context with which to delete the directory (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## DELETEDFILE Procedure

This procedure deletes the specified file.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.DELETEDFILE (  
  store_name    IN    VARCHAR2,  
  path          IN    VARCHAR2,  
  filter        IN    VARCHAR2,  
  soft_delete   IN    BOOLEAN,  
  ctx           IN    DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–9 DELETEDFILE Procedure Parameters**

Parameter	Description
store_name	Name of store
path	Name of path to the file
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see "Deletion Operations" on page 48-6).
ctx	Context with which to delete the file (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T</a> Object Type)

## GETFEATURES Function

This function returns the features of a store.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.GETFEATURES (  
    store_name      IN      VARCHAR2)  
RETURN INTEGER;
```

### Parameters

**Table 48–10** *GETFEATURES Function Parameters*

Parameter	Description
store_name	Name of store

### Return Values

DBMS\_CONTENT.FEATURE\_\* features supported by the Service Provider

## GETPATH Procedures

This procedure returns existing path items (such as files and directories). This includes both data and metadata (properties).

The client can request (using `prop_flags`) that specific properties be returned. File path names can be read either by specifying a BLOB locator using the `prop_data` bitmask in `prop_flags` (see [DBMS\\_DBFS\\_CONTENT Constants - Property Access Flags](#)) or by passing one or more RAW buffers.

When `forUpdate` is 0, this procedure also accepts a valid "as of" timestamp parameter as part of `ctx` that can be used by stores to implement "as of" style flashback queries. Mutating versions of the GETPATH Procedures do not support these modes of operation.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.GETPATH (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  content     OUT         NOCOPY BLOB,
  item_type   OUT         INTEGER,
  prop_flags  IN          INTEGER,
  forUpdate   IN          INTEGER,
  deref       IN          INTEGER,
  ctx         IN          DBMS_CONTENT_CONTEXT_T);
```

```
DBMS_DBFS_CONTENT_SPI.GETPATH (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  amount     IN OUT      NUMBER,
  offset      IN          NUMBER,
  buffer      OUT         NOCOPY RAW,
  prop_flags  IN          INTEGER,
  ctx         IN          DBMS_CONTENT_CONTEXT_T);
```

```
DBMS_DBFS_CONTENT_SPI.GETPATH (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  amount     IN OUT      NUMBER,
  offset      IN          NUMBER,
  buffers     OUT         NOCOPY DBMS_CONTENT_RAW_T,
  prop_flags  IN          INTEGER,
  ctx         IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–11** GETPATH Procedure Parameters

Parameter	Description
<code>store_name</code>	Name of store
<code>path</code>	Name of path to path items

**Table 48–11 (Cont.) GETPATH Procedure Parameters**

Parameter	Description
properties	One or more properties and their values to be returned depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
amount	On input, number of bytes to be read. On output, number of bytes read
offset	Byte offset from which to begin reading
buffer	Buffer to which to write
buffers	Buffers to which to write
prop_flags	Determines which properties are set, returned, or both. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
forUpdate	Specifies that a lock should be taken to signify exclusive write access to the path item
deref	If nonzero, attempts to resolve the given path item to actual data provided it is a reference (symbolic link)
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## GETPATHBYSTOREID Function

If the underlying GUID is found in the underlying store, this function returns the store-qualified path name.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.GETPATHBYSTOREID (  
  store_name      IN      VARCHAR2,  
  guid            IN      INTEGER)  
RETURN VARCHAR2;
```

### Parameters

**Table 48–12** *GETPATHBYSTOREID Function Parameters*

Parameter	Description
store_name	Name of store
guid	Unique ID representing the desired path item

### Return Values

Store-qualified path name represented by the GUID

### Usage Notes

If the `STD_GUID` is unknown, a `NULL` value is returned. Clients are expected to handle this as appropriate.



## GETPATHNOWAIT Procedure

This procedure implies that the operation is for an update, and, if implemented (see `FEATURE_NOWAIT` in [DBMS\\_DBFS\\_CONTENT Constants - Store Features](#)), allows providers to return an exception (ORA-00054) rather than wait for row locks.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.GETPATHNOWAIT (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  content     OUT         NOCOPY BLOB,
  item_type   OUT         INTEGER,
  prop_flags  IN          INTEGER,
  deref       IN          INTEGER,
  ctx        IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–13** *GETPATHNOWAIT Procedure Parameters*

Parameter	Description
store_name	Name of store
path	Name of path to path items
properties	One or more properties and their values to be returned depending on <code>prop_flags</code> (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
prop_flags	Determines which properties are returned. Default is <code>PROP_STD</code> . Specify properties to be returned by setting <code>prop_spec</code> , and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
deref	If nonzero, attempts to resolve the given path item to actual data provided it is a reference (symbolic link)
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## GETSTOREID Function

This function returns the ID of a store.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.GETSTOREID (  
    store_name          IN          VARCHAR2)  
    RETURN NUMBER;
```

### Parameters

**Table 48–14** *GETSTOREID Function Parameters*

Parameter	Description
store_name	Name of store

### Return Values

ID of the Store

### Usage Notes

A store ID identifies a provider-specific store, across registrations and mounts, but independent of changes to the store contents. For this reason, changes to the store table or tables should be reflected in the store ID, but re-initialization of the same store table or tables should preserve the store ID.

## GETVERSION Function

This function returns the version associated with a store.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.GETVERSION (  
    store_name      IN      VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 48–15** *GETVERSION Function Parameters*

Parameter	Description
store_name	Name of store

### Return Values

A "version" (either specific to a provider package, or to an individual store) based on a standard *a.b.c* naming convention (for *major*, *minor*, and *patch* components)

## LIST Function

This function lists the contents of a directory path name.

The invoker of the subprogram has the option to investigate recursively into sub-directories, to make soft-deleted items visible, to use a flashback "as of" a specified timestamp, and to filter items within the store based on list predicates.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.LIST (
  store_name  IN   VARCHAR2,
  path        IN   VARCHAR2,
  filter      IN   VARCHAR2,
  recurse     IN   INTEGER,
  ctx         IN   DBMS_CONTENT_CONTEXT_T)
RETURN DBMS_CONTENT_LIST_ITEMS_T PIPELINED;
```

### Parameters

**Table 48–16** LIST Function Parameters

Parameter	Description
store_name	Name of respository
path	Name of path to directories
filter	A filter, if any, to be applied
recurse	If 0, do not execute recursively. Otherwise, recursively list the contents of directories and files below the given directory.
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

### Return Values

Path items found that match the path, filter and criteria for executing recursively (see [DBMS\\_DBFS\\_CONTENT\\_LIST\\_ITEMS\\_T Table Type](#))

### Usage Notes

This function returns only list items; the client is expected to explicitly use one of the [GETPATH Procedures](#) to access the properties or content associated with an item.

## LOCKPATH Procedure

This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.LOCKPATH (
  store_name    IN    VARCHAR2,
  path          IN    VARCHAR2,
  lock_type     IN    INTEGER,
  lock_data     IN    VARCHAR2,
  ctx           IN    DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–17** LOCKPATH Procedure Parameters

Parameter	Description
store_name	Name of store
path	Path name of items to be locked
lock_type	One of the available lock types (see <a href="#">DBMS_DBFS_CONTENT Constants - Lock Types</a> )
lock_data	Optional user data to be associated with the lock
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

### Usage Notes

- It is the responsibility of the store and its providers (assuming it supports user-defined lock checking) to ensure that lock and unlock operations are performed in a consistent manner.
- The status of locked items is available by means of various optional properties (see OPT\_LOCK\* in [DBMS\\_DBFS\\_CONTENT Constants - Optional Properties](#)).

## PURGEALL Procedure

This procedure purges all soft-deleted entries matching the path and optional filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.PURGEALL (  
  store_name    IN      VARCHAR2,  
  path          IN      VARCHAR2,  
  filter        IN      VARCHAR2,  
  ctx           IN      DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–18** *PURGEALL Procedure Parameters*

Parameter	Description
store_name	Name of store
path	Name of path to file items
filter	A filter, if any, to be applied based on specified criteria
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## PURGEPATH Procedure

This procedure purges any soft-deleted versions of the given path item.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.PURGEPATH (
  path          IN      VARCHAR2,
  filter        IN      VARCHAR2,
  store_name    IN      VARCHAR2,
  ctx           IN      DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–19** *PURGEPATH Procedure Parameters*

Parameter	Description
store_name	Name of store
path	Name of path to file items
filter	A filter, if any, to be applied
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## PUTPATH Procedures

This procedure creates a new path item.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.PUTPATH (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  content     IN OUT NOCOPY BLOB,
  item_type   OUT         INTEGER,
  prop_flags  IN          INTEGER,
  ctx        IN          DBMS_CONTENT_CONTEXT_T);
```

```
DBMS_DBFS_CONTENT_SPI.PUTPATH (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  amount     IN          NUMBER,
  offset     IN          NUMBER,
  buffer     IN          RAW,
  prop_flags  IN          INTEGER,
  ctx        IN          DBMS_CONTENT_CONTEXT_T);
```

```
DBMS_DBFS_CONTENT_SPI.PUTPATH (
  store_name  IN          VARCHAR2,
  path        IN          VARCHAR2,
  properties  IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  written    OUT         NUMBER,
  offset     IN          NUMBER,
  buffers    IN          DBMS_CONTENT_RAW_T,
  prop_flags  IN          INTEGER,
  ctx        IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–20 PUTPATH Procedure Parameters**

Parameter	Description
store_name	Name of store
path	Path name of item to be put
properties	One or more properties and their values to be set depending on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )
amount	Number of bytes to be read
written	Number of bytes written
offset	Byte offset from which to begin reading
buffer	Buffer to which to write
buffers	Buffers to which to write



**Table 48–20 (Cont.) PUTPATH Procedure Parameters**

Parameter	Description
prop_flags	Determines which properties are set. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> with properties whose values are of interest.
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## Usage Notes

- All path names allow their metadata (properties) to be read and modified. On completion of the call, the client can access specific properties using prop\_flags (see [DBMS\\_DBFS\\_CONTENT Constants - Property Access Flags](#)).
- On completion of the call, the client can request a new BLOB locator that can be used to continue data access using the prop\_data bitmask in prop\_flags (see [DBMS\\_DBFS\\_CONTENT Constants - Property Access Flags](#)).
- Files can also be written without using BLOB locators, by explicitly specifying logical offsets or buffer-amounts, and a suitably sized buffer.

## RENAMEPATH Procedure

This procedure renames or moves a path. This operation can be performed across directory hierarchies and mount-points as long as it is within the same store.

---



---

**Note:** See ["Rename and Move Operations"](#) on page 48-7

---



---

### Syntax

```
DBMS_DBFS_CONTENT_SPT.RENAMEPATH (
  store_name      IN          VARCHAR2,
  oldPath         IN          VARCHAR2,
  newPath         IN          VARCHAR2,
  properties      IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  ctx             IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–21** *RENAMEPATH Procedure Parameters*

Parameter	Description
store_name	Name of store, must be unique
oldPath	Name of path prior to renaming
newPath	Name of path after renaming
properties	One or more properties and their values to be set depending on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## RESTOREALL Procedure

This procedure restores all soft-deleted path items meeting the path and optional filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.RESTOREALL (
  store_name  IN    VARCHAR2,
  path        IN    VARCHAR2,
  filter      IN    VARCHAR2,
  ctx         IN    DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–22** *RESTOREALL Procedure Parameters*

Parameter	Description
store_name	Name of store
path	Name of path to path items
filter	A filter, if any, to be applied
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## RESTOREPATH Procedure

This procedure restores all soft-deleted path items that match the given path and optional filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.RESTOREPATH (  
  store_name      IN      VARCHAR2,  
  path            IN      VARCHAR2,  
  filter          IN      VARCHAR2,  
  ctx             IN      DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–23** *RESTOREPATH Procedure Parameters*

Parameter	Description
store_name	Name of store
path	Name of path to path items
filter	A filter, if any, to be applied
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## SEARCH Function

This function searches for path items matching the given path and filter criteria.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.SEARCH (
  store_name  IN    VARCHAR2,
  path        IN    VARCHAR2,
  filter      IN    VARCHAR2,
  recurse     IN    INTEGER,
  ctx         IN    DBMS_CONTENT_CONTEXT_T)
RETURN DBMS_CONTENT_LIST_ITEMS_T PIPELINED;
```

### Parameters

**Table 48–24** LIST Function Parameters

Parameter	Description
store_name	Name of store
path	Name of path to the path items
filter	A filter, if any, to be applied
recurse	If 0, do not execute recursively. Otherwise, recursively search the contents of directories and files below the given directory.
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

### Return Values

Path items matching the given path and filter criteria (see [DBMS\\_DBFS\\_CONTENT\\_LIST\\_ITEMS\\_T Table Type](#))

## SETPATH Procedure

This procedure assigns a path name to a path item represented by contentID.

Stores and their providers that support contentID-based access and lazy path name binding also support the `SETPATH` Procedure that associates an existing contentID with a new path.

---



---

**Note:** See ["Rename and Move Operations"](#) on page 48-7

---



---

### Syntax

```
DBMS_DBFS_CONTENT_SPI.SETPATH (
  store_name    IN          VARCHAR2,
  contentID     IN          RAW,
  path          IN          VARCHAR2,
  properties    IN OUT NOCOPY DBMS_CONTENT_PROPERTIES_T,
  ctx           IN          DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–25** *SETPATH Procedure Parameters*

Parameter	Description
store_name	Name of the store
contentID	Unique identifier for the item to be associated
path	Name of path to path item
properties	One or more properties and their values to be set depending on prop_flags (see <a href="#">DBMS_DBFS_CONTENT_PROPERTIES_T Table Type</a> )
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )

## SPACEUSAGE Procedure

This procedure queries file system space usage statistics. Providers are expected to support this subprogram for their stores and to make a best effort determination of space usage, especially if the store consists of multiple tables, indexes, LOBs, and so on.

### Syntax

```
DBMS_DBFS_CONTENT_SPI.SPACEUSAGE (
  store_name  IN      VARCHAR2,
  blksize    OUT     INTEGER,
  tbytes     OUT     INTEGER,
  fbytes     OUT     INTEGER,
  nfile      OUT     INTEGER,
  ndir       OUT     INTEGER,
  nlink      OUT     INTEGER,
  nref       OUT     INTEGER);
```

### Parameters

**Table 48–26** SPACEUSAGE Procedure Parameters

Parameter	Description
store_name	Name of store
blksize	Natural tablespace blocksize that holds the store. If multiple tablespaces with different blocksizes are used, any valid blocksize is acceptable.
tbytes	Total size of the store in bytes computed over all segments that comprise the store
fbytes	Free or unused size of the store in bytes computed over all segments that comprise the store
nfile	Number of currently available files in the store
ndir	Number of currently available directories in the store
nlink	Number of currently available links in the store
nref	Number of currently available references in the store

### Usage Notes

- A space usage query on the top-level root directory returns a combined summary of the space usage of all available distinct stores under it (if the same store is mounted multiple times, it is still counted only once).
- Since database objects are dynamically expandable, it is not easy to estimate the division between "free" space and "used" space.

## UNLOCKPATH Procedure

This procedure unlocks path items that were previously locked with the [LOCKPATH Procedure](#).

### Syntax

```
DBMS_DBFS_CONTENT_SPI.UNLOCKPATH (  
  store_name      IN      VARCHAR2,  
  path            IN      VARCHAR2,  
  ctx             IN      DBMS_CONTENT_CONTEXT_T);
```

### Parameters

**Table 48–27 UNLOCKPATH Procedure Parameters**

Parameter	Description
store_name	Name of store
path	Name of path to the path items
ctx	Context with which to access the path items (see <a href="#">DBMS_DBFS_CONTENT_CONTEXT_T Object Type</a> )



The Oracle Database File System Hierarchical Store is implemented in the `DBMS_DBFS_HS` package. This package provides users the ability to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management for their database tables.

**See Also:**

- *Oracle Database SecureFiles and Large Objects Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_DBFS\\_HS](#)
  - Overview
  - Security Model
  - Constants
  - Operational Notes
- [Summary of DBMS\\_DBFS\\_HS Subprograms](#)

## Using DBMS\_DBFS\_HS

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Operational Notes](#)

## Overview

The DBMS\_DBFS\_HS package is a service provider underneath the DBMS\_DBFS\_CONTENT package that enables use of tape or Amazon S3 Web service as storage for data.

The data on tape or Amazon S3 Web service is part of the Oracle Database and can be accessed through all standard interfaces, but only through the database. The package allows users to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management of their content.

The package initially stores all content files in level-1 cache. As the level-1 cache fills up, content files are moved to level-2 cache and then to an external storage device using bulk writes.

## Security Model

The `DBMS_DBFS_HS` package runs with invoker's rights.

## Constants

The DBMS\_DBFS\_HS package uses the constants shown in following tables:

- [DBMS\\_DBFS\\_HS Constants - Used by the CREATESTORE Procedure](#)
- [DBMS\\_DBFS\\_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function](#)
- [DBMS\\_DBFS\\_HS Constants - Used by the REGISTERSTORECOMMAND Function](#)
- [DBMS\\_DBFS\\_HS Constants - Failure/Success/Error](#)

**Table 49–1 DBMS\_DBFS\_HS Constants - Used by the CREATESTORE Procedure**

Constant	Type	Value	Description
STORETYPE_TAPE	VARCHAR2 (50)	'HS_TAPE'	Use tape as a storage tier
STORETYPE_AMAZONS3	VARCHAR2 (50)	'HS_S3'	Use Amazon S3 Web service as a storage tier

**Table 49–2 DBMS\_DBFS\_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

Constant	Type	Value	Description
PROPNAME_BUCKET	VARCHAR2 (50)	'BUCKET'	<p>Specifies the AWS bucket to be used as a storage tier by the Hierarchical Store.</p> <p>Restrictions on bucket name are:</p> <ol style="list-style-type: none"> <li>1) Bucket names can only contain lowercase letters, numbers, periods (.) and dashes(-). Note that underscores ( ) are invalid.</li> <li>2) Bucket names must start with a number or letter.</li> <li>3) Bucket names cannot be in an IP address style (192.168.5.4).</li> <li>4) Bucket names must be between 3 and 63 characters long.</li> <li>5) Bucket names should not end with a dash.</li> <li>6) Dashes cannot appear next to periods. For example, my-.bucket.com is invalid.</li> </ol>
PROPNAME_CACHESIZE	VARCHAR2 (50)	'CACHE_SIZE'	<p>Specifies the cumulative cache size used for the Hierarchical Store. This property is set by the <a href="#">CREATESTORE Procedure</a> and can be modified by the <a href="#">RECONFIGCACHE Procedure</a>. It cannot be modified by the <a href="#">SETSTOREPROPERTY Procedure</a>, though its value can be queried by the <a href="#">GETSTOREPROPERTY Function</a>.</p>

**Table 49–2 (Cont.) DBMS\_DBFS\_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

Constant	Type	Value	Description
PROPNAME_ COMPRESSLEVEL	VARCHAR2 (50)	'COMPRESSION _LEVEL'	Use to enable compression of files stored in the DBFS hierarchical store. It specifies the compression level to be used for compressing the files
PROPNAME_ ENABLECLEANUPO NDELETE	VARCHAR2 (50)	'ENABLE_ CLEANUP_ON_ DELETE'	If this property is set to 'TRUE', whenever the user invokes the <a href="#">DELETEFILE Procedure</a> in the <a href="#">DBMS_DBFS_CONTENT</a> interface on a file residing in the DBMS_DBFS_HS store, the DBMS_DBFS_HS removes the file on the external storage that contains this user file provided that the file has no other useful data. By default, the property is set to 'TRUE' for STORETYPE_AMAZONS3 and 'FALSE' for STORETYPE_TAPE.
PROPNAME_ HTTPPROXY	VARCHAR2 (50)	'HTTP_PROXY'	Specifies the DNS name of the HTTP proxy, if any, that is needed to access the Amazon S3 storage service
PROPNAME_ LICENSEID	VARCHAR2 (50)	'LICENSE_ID'	Specifies the license ID associated with the library libosbws11.so.
PROPNAME_ LOBCACHE_QUOTA	VARCHAR2 (50)	'LOBCACHE_ QUOTA'	Specifies fraction of the cache_size which is allocated for level 1 cache. The default value of this parameter is NULL which means that 0.8 (= 80%) of the cache_size is used for level 1 cache.  This property cannot be modified by the <a href="#">SETSTOREPROPERTY Procedure</a> though its value can be queried by the <a href="#">GETSTOREPROPERTY Function</a> . Its value is set by <a href="#">CREATESTORE Procedure</a> and can be modified by the <a href="#">RECONFIGCACHE Procedure</a> .
PROPNAME_ MEDIAPOOL	VARCHAR2 (50)	'MEDIA_POOL'	Specifies the media pool number to use for storing the content
PROPVAL_ COMPLVL_NONE	VARCHAR2 (50)	'NONE'	Indicates no compression
PROPVAL_ COMPLVL_LOW	VARCHAR2 (50)	'LOW'	Use to set the compression level to LOW. This is expected to have the best performance while still providing a good compression ratio.
PROPVAL_ COMPLVL_MEDIUM	VARCHAR2 (50)	'MEDIUM'	Use to set the compression level to MEDIUM. This compression level is expected to provide better compression ratio than LOW but the time required for compression will be higher than compression level LOW.

**Table 49–2 (Cont.) DBMS\_DBFS\_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

Constant	Type	Value	Description
PROPVAL_ COMPLVL_HIGH	VARCHAR2 (50)	'HIGH'	Use to set the compression level to HIGH. This compression level is expected to provide the best compression ratio but compression time will in general be highest among the 3 compression levels.
PROPNAME_ OPTTARBALLSIZE	VARCHAR2 (50)	'OPTIMAL_ TARBALL_ SIZE'	Specifies <code>optimal_tarball_size</code> as the maximum possible size of an archive file.  Multiple content files are bundled together into one archive file and then the archive file is transferred to tape or Amazon S3. This is because creating one file on tape or Amazon S3 for every content file in the store is a prohibitively expensive operation.  This property cannot be modified by the <a href="#">SETSTOREPROPERTY Procedure</a> though its value can be queried by the <a href="#">GETSTOREPROPERTY Function</a> . Its value is set by <a href="#">CREATESTORE Procedure</a> and can be modified by the <a href="#">RECONFIGCACHE Procedure</a> .
PROPNAME_ READCHUNKSIZE	VARCHAR2 (50)	'READ_CHUNK_ SIZE'	Specifies the size used by the SBT protocol to transfer data from tape or S3. This chunk is allocated in memory per transaction for retrieval of content files from an archive store, so the value of this property should be conservative. The default size of 1MB is typically good for most users.
PROPNAME_ S3HOST	VARCHAR2 (50)	'S3_HOST'	Specifies the HOST name of the Amazon S3 storage service. It must be <code>s3.amazonaws.com</code> .
PROPNAME_ SBT_ LIBRARY	VARCHAR2 (50)	'SBT_ LIBRARY'	Specifies the path of the shared library used by RMAN to communicate with Amazon S3. It is named <code>libosbws11.so</code> and is available in <code>rdbms/lib</code> directory.
PROPNAME_ STREAMABLE	VARCHAR2 (50)	'STREAMABLE'	Indicates whether buffer-based PUT or GET should be done on this store. Valid values for are TRUE and FALSE. The default value of this property is TRUE.

**Table 49–2 (Cont.) DBMS\_DBFS\_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

Constant	Type	Value	Description
PROPNAME_ WALLET	VARCHAR2 (50)	'WALLET'	<p>The value of this property should be of the form:</p> <p>LOCATION=file:filename CREDENTIAL_ ALIAS=access/secret_alias PROXY_AUTH_ ALIAS=proxyusername/password alias</p> <p>Defines the Oracle Wallet which contains the credentials of the Amazon S3 account associated with the store under consideration.</p> <p>LOCATION: The directory path that contains the Oracle wallet. The format is file:directory-path The format of wallet_path in Windows is, for example: file:c:\WINNT\Profiles\username\WALLETS In UNIX or Linux it is, for example: file:/home/username/wallets</p> <p>When the package is executed in the Oracle database server, the wallet is accessed from the database server.</p> <p>PASSWORD: Defines the wallet password. If auto-login is enabled in wallet (this can be changed using the OWM utility), this parameter does not have to be specified. By default, the mkstore utility enables auto-login.</p> <p>CREDENTIAL_ALIAS: Defines the credential alias for ACCESS_KEY and SECRET_KEY</p>
PROPNAME_ WRITECHUNKSIZ	VARCHAR2 (50)	'WRITE_ CHUNK_SIZE'	<p>Specifies the size used by the SBT protocol to transfer data to tape or S3.</p> <p>This chunk is allocated in memory per transaction for PUT of Content Files to an archive store so the value should be conservative.</p> <p>The default size of 1MB is typically good for most users.</p>

**Table 49–3 DBMS\_DBFS\_HS Constants - Used by the REGISTERSTORECOMMAND Function**

Constant	Type	Value	Description
BEFORE_PUT	NUMBER	'1'	Specified operation must be performed before writing a SECUREFILE to the remote store



**Table 49–3 (Cont.) DBMS\_DBFS\_HS Constants - Used by the REGISTERSTORECOMMAND Function**

Constant	Type	Value	Description
BEFORE_GET	NUMBER	'2'	Specified operation must be performed before a retrieval operation such as reading a SECUREFILE from the remote device

**Table 49–4 DBMS\_DBFS\_HS Constants - Failure/Success/Error**

Constant	Type	Value	Description
FAIL	NUMBER	'0'	Procedure or function did not execute successfully
SUCCESS	NUMBER	'1'	Procedure or function completed successfully
ERROR	NUMBER	'2'	Procedure or function returned an error

## Operational Notes

When the `DBMS_DBFS_HS` package is executed in the Oracle database server, the wallet is accessed from the database server.

---

## Summary of DBMS\_DBFS\_HS Subprograms

**Table 49–5 DBMS\_DBFS\_HS Package Subprograms**

Subprogram	Description
<a href="#">CLEANUPUNUSEDBACKUPFILES Procedure</a> on page 49-12	Removes files created on the external storage device that hold no currently used data
<a href="#">CREATEBUCKET Procedure</a> on page 49-13	Creates an AWS bucket, associated with a store of type <code>STORETYPE_AMAZONS3</code> into which the Hierarchical Store can then move data
<a href="#">CREATESTORE Procedure</a> on page 49-14	Creates a new hierarchical store
<a href="#">DEREGSTORECOMMAND Function</a> on page 49-16	Removes a command that had been previously associated with a store through the <a href="#">RECONFIGCACHE Procedure</a>
<a href="#">DROPSTORE Procedure</a> on page 49-16	Deletes a previously created hierarchical store
<a href="#">FLUSHCACHE Procedure</a> on page 49-18	Flushes (writes out) dirty contents from the level-1 cache.
<a href="#">GETSTOREPROPERTY Function</a> on page 49-19	Retrieves the values of a property of a store
<a href="#">RECONFIGCACHE Procedure</a> on page 49-20	Reconfigures the parameters of the database cache used by the store
<a href="#">REGISTERSTORECOMMAND Procedure</a> on page 49-21	Registers commands for a store with the Hierarchical Store to be sent to the Media Manager for the external storage device associated with the store
<a href="#">SENDCOMMAND Procedures</a> on page 49-22	Sends a command to be executed on the external storage device's Media Manager
<a href="#">SETSTOREPROPERTY Procedure</a> on page 49-23	Stores properties of a store in the database
<a href="#">STOREPUSH Procedure</a> on page 49-26	Pushes locally staged data to the remote storage

## CLEANUPUNUSEDBACKUPFILES Procedure

This procedure removes files created on the external storage device that hold no currently used data in them.

### Syntax

```
DBMS_DBFS_HS.CLEANUPUNUSEDBACKUPFILES (  
    store_name    IN    VARCHAR2);
```

### Parameters

**Table 49–6** *CLEANUPUNUSEDBACKUPFILES Procedure Parameters*

Parameter	Description
store_name	Name of store

### Usage Notes

- The action of removing files from external storage device can not be rolled back.
- This method can be executed periodically to clear space on the external storage device. Asynchronously deleting content from the external storage device is useful because it has minimal impact on the OLTP performance. The periodic scheduling can be accomplished using the DBMS\_SCHEDULER package.

## CREATEBUCKET Procedure

This procedure creates an AWS bucket, associated with a store of type `STORETYPE_AMAZONS3` into which the Hierarchical Store can then move data.

### Syntax

```
DBMS_DBFS_HS.CREATEBUCKET (  
    store_name    IN    VARCHAR2);
```

### Parameters

**Table 49–7** *CREATEBUCKET Procedure Parameters*

Parameter	Description
store_name	Name of store

### Usage Notes

- The `PROPNAME_BUCKET` property of the store should be set before this subprogram is called.
- Once this procedure has successfully created a bucket in Amazon S3, the bucket can only be deleted using out-of-band methods, such as logging-in to S3 and deleting data (directories, files, and other items) for the bucket.

## CREATESTORE Procedure

This procedure creates a new hierarchical store `store_name` of type `STORE_TYPE` (`STORETYPE_TAPE` or `STORETYPE_AMAZONS3`) in schema `schema_name` (defaulting to current schema) under the ownership of the invoking session user.

### Syntax

```
DBMS_DBFS_HS.CREATESTORE (
  store_name          IN    VARCHAR2,
  store_type          IN    VARCHAR2,
  tbl_name            IN    VARCHAR2,
  tbs_name            IN    VARCHAR2,
  cache_size          IN    NUMBER,
  lob_cache_quota     IN    NUMBER DEFAULT NULL,
  optimal_tarball_size IN    NUMBER DEFAULT NULL,
  schema_name         IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 49–8 CREATESTORE Procedure Parameters**

Parameter	Description
<code>store_name</code>	Name of store
<code>store_type</code>	<code>STORETYPE_TAPE</code> or <code>STORETYPE_AMAZONS3</code>
<code>tbl_name</code>	Table for store entries
<code>tbs_name</code>	Tablespace for the store
<code>cache_size</code>	Amount of space used by the store to cache content in given tablespace
<code>lob_cache_quota</code>	Fraction of the <code>cache_size</code> which is allocated for level 1 cache. The default value of this parameter is <code>NULL</code> which means that 0.8 (= 80%) of the <code>cache_size</code> is used for level 1 cache.
<code>optimal_tarball_size</code>	Maximum possible size of the archive file.  Multiple content files are bundled together into one archive file, and then the archive file is transferred to tape or Amazon S3. This is because creating one file on tape or Amazon S3 for every content file in the store is a prohibitively expensive operation.  The value of is set by default to 10GB for tape and to 100MB for Amazon S3.
<code>schema_name</code>	Schema for the store

### Usage Notes

- Store names must be unique for an owner. The same store names can be used for different stores owned by different owners.
- Once a table space has been specified to store the store's content in a database, it cannot be changed later.
- This subprogram will execute like a DDL statement, performing an automatic `COMMIT` before and after execution.
- Stores using `DBMS_DBFS_HS` must not use singleton mount. This means that the singleton parameter should be `FALSE` and the `store_mount` parameter should

have a non-NULL value in a call to the DBMS\_DBFS\_CONTENT.MOUNTSTORE Procedure.

## DEREGSTORECOMMAND Function

This procedure removes a command that had been previously associated with a store through the [RECONFIGCACHE Procedure](#).

### Syntax

```
DBMS_DBFS_HS.DEREGSTORECOMMAND (  
    store_name      IN      VARCHAR2,  
    message         IN      VARCHAR2);
```

### Parameters

**Table 49–9** *DEREGSTORECOMMAND Procedure Parameters*

Parameter	Description
store_name	Name of store
message	Message to be deregistered

### Usage Notes

If this subprogram successfully executes, its actions cannot be rolled back by the user. If the user wants to restore the previous state, the user must call the [REGISTERSTORECOMMAND Procedure](#).



## DROPSTORE Procedure

This procedure deletes a previously created hierarchical store specified by name and owned by the invoking session user.

### Syntax

```
DBMS_DBFS_HS.DROPSTORE (
  store_name  IN  VARCHAR2,
  opt_flags   IN  INTEGER DEFAULT 0);
```

### Parameters

**Table 49–10** *DROPSTORE Procedure Parameters*

Parameter	Description
store_name	Name of store owned by the invoking session user
opt_flags	User can specify optional flags. If <code>DISABLE_CLEANUPBACKUPFILES</code> is specified as one of the optional flags, the call to the <a href="#">CLEANUPUNUSEDBACKUPFILES Procedure</a> is not issued. By default, when this flag is not set, the procedure implicitly cleans-up all unused backup files.

### Usage Notes

- The procedure executes like a DDL in that it auto-commits before and after its execution.
- If `CLEANUPBACKUPFILES` is disabled during the procedure, the user must resort to out-of-band techniques to cleanup unused backup files. No further invocations of `CLEANUPBACKFILES` for a dropped store are possible through hierarchical store.
- This subprogram will un-register the store from `DBMS_DBFS_CONTENT` package. All files in the given store are deleted from the store (Tape or Amazon S3 Web service). The database table holding the store's entries in the database, is also dropped by this subprogram.

## FLUSHCACHE Procedure

This procedure flushes out dirty contents from level-1 cache, which can be locked, to level-2 cache, thereby freeing-up space in level 1 cache.

### Syntax

```
DBMS_DBFS_HS.FLUSHCACHE (  
    store_name    IN    VARCHAR2);
```

### Parameters

**Table 49–11** *FLUSHCACHE Procedure Parameters*

Parameter	Description
store_name	Name of store

## GETSTOREPROPERTY Function

This function retrieves the values of a property.

### Syntax

```
DBMS_DBFS_HS.GETSTOREPROPERTY (
  store_name      IN      VARCHAR2,
  property_name   IN      VARCHAR2,
  noexcp          IN      BOOLEAN DEFAULT FALSE) RETURN VARCHAR2;
```

### Parameters

**Table 49–12** *GETSTOREPROPERTY Function Parameters*

Parameter	Description
store_name	Name of store
property_name	Name of property
noexcp	If set to FALSE, raises an exception if the property does not exist in the database. If noexcp is set to TRUE, returns NULL if the property does not exist.

### Return Values

The values of a property.

### Usage Notes

The specified store must already have been created.

## RECONFIGCACHE Procedure

This procedure reconfigures the parameters of the database cache being used by the store.

### Syntax

```
DBMS_DBFS_HS.RECONFIGCACHE (
  store_name          IN    VARCHAR2,
  cache_size         IN    NUMBER DEFAULT NULL,
  lobcache_quota     IN    NUMBER DEFAULT NULL,
  optimal_tarball_size IN  NUMBER DEFAULT NULL);
```

### Parameters

**Table 49–13 RECONFIGCACHE Procedure Parameters**

Parameter	Description
store_name	Name of store
cache_size	Cumulative cache size used for the Hierarchical Store
lobcache_quota	Fraction of the cache size that are assigned to level 1 cache
optimal_tarball_size	Maximum possible size of an archive file. Since creating one file for every content file in the store is a prohibitively expensive operation, multiple content files are bundled together into one archive file for transfer to tape or Amazon S3.

### Usage Notes

- The specified store must already have been created before reconfiguration.
- The Hierarchical Store uses a level 1 cache and a level 2 cache. The level 1 cache subsumes most of the working set and the level 2 cache is used to perform bulk writes to the backend device.
- If any of the last 3 parameters is NULL, its value specified during store creation is used. If the parameter was NULL when the call to the [CREATESTORE Procedure](#) was issued, the DBMS\_DBFS\_HS package assigns a default value.

The DBMS\_DBFS\_HS package optimistically tries to allocate more than 1 tarball's worth of size for level 2 cache to facilitate concurrency, though a minimum of 1 tarball size is necessary for level 2 cache.

The values for cumulative cache size and LOB cache quota decide allocation of space for the two caches. If values are not provided, a user might see an `INSUFFICIENT_CACHE` exception. In that case, it is better to revise the cache parameters in order to have a working store.

- If this subprogram successfully executes, its actions cannot be rolled back by the user. In that case, the user should call RECONFIGCACHE again with new or modified parameters.

## REGISTERSTORECOMMAND Procedure

This procedure registers commands for a store with the Hierarchical Store. These commands are sent to the Media Manager for the external storage device associated with the store.

### Syntax

```
DBMS_DBFS_HS.REGISTERSTORECOMMAND (
  store_name      IN      VARCHAR2,
  message         IN      VARCHAR2,
  flags           IN      NUMBER);
```

### Parameters

**Table 49–14 REGISTERSTORECOMMAND Procedure Parameters**

Parameter	Description
store_name	Name of store
message	Message to be sent to the Media Manager of the external store
flags	Valid values: <ul style="list-style-type: none"> <li>▪ BEFORE_PUT CONSTANT NUMBER := 1;</li> <li>▪ BEFORE_GET CONSTANT NUMBER := 2;</li> </ul>

### Usage Notes

- These commands are sent before the next read or write of content. When the Hierarchical Store wants to push (or get) data to (or from) the storage device, it begins a session (to communicate with the device). After beginning the session, it sends all registered commands for the to the relevant device before writing (or getting) any data.
- If this method successfully executes, its actions cannot be rolled back by the user. To restore the previous state the user must call the [DEREGSTORECOMMAND Function](#).

## SENDERCOMMAND Procedures

This procedure sends a command to be executed on the external storage device's Media Manager.

### Syntax

```
DBMS_DBFS_HS.SENDERCOMMAND (  
  store_name  IN    VARCHAR2,  
  message     IN    VARCHAR2);
```

### Parameters

**Table 49–15** *SENDERCOMMAND Procedure Parameters*

Parameter	Description
store_name	Name of store
message	Message string to be executed

## SETSTOREPROPERTY Procedure

This procedure stores properties of a store in the database as name-value pairs.

### Syntax

```
DBMS_DBFS_HS.SETSTOREPROPERTY (
  store_name      IN   VARCHAR2,
  property_name   IN   VARCHAR2,
  property_value  IN   VARCHAR2);
```

### Parameters

**Table 49–16 SETSTOREPROPERTY Procedure Parameters**

Parameter	Description
store_name	Name of store
property_name	For a store using Tape device, there are three properties whose values must be set by the user, and four properties that have default values. Stores of type STORETYPE_AMAZONS3 have properties with default values. The various options for both types of stores are detailed under property_value.
property_value	<p><b>Stores using a Tape Device</b></p> <p>The values for the following properties must be set by the user:</p> <ul style="list-style-type: none"> <li>■ PROPNAME_SBTLIBRARY - This should point to the shared library used by RMAN to communicate with the external tape device. It is usually named libobk.so.</li> <li>■ PROPNAME_MEDIAPool - Media pool number for storing content</li> <li>■ PROPNAME_CACHE_SIZE - Amount of space, in bytes, used for the cache of this store</li> </ul> <p>The following properties, which have default values assigned to them when a store is created, benefit from tuning:</p> <ul style="list-style-type: none"> <li>■ PROPNAME_READCHUNKSIZE and PROPNAME_WRITECHUNKSIZE - These are the sizes used by the SBT protocol to transfer data to and from the tapes. These chunks are allocated in memory per transaction, so the values should be conservative. The default size is 1MB.</li> <li>■ PROPNAME_STREAMABLE - Indicates whether DBFS_LINKS can perform read operations (for example SELECT or DBMS_LOB.READ) directly from the store, or if the data must be copied back into the database before it can be read</li> <li>■ PROPNAME_ENABLECLEANUPONDELETE - Indicates if unused files on the external storage device should be deleted by DBMS_DBFS_HS. Valid values for this property are 'FALSE' for STORETYPE_TAPE.</li> <li>■ PROPNAME_COMPRESSLEVEL - Describes how files written to Tape should be compressed. It can be set to PROPVAL_COMPLVL_NONE, PROPVAL_COMPLVL_LOW, PROPVAL_COMPLVL_MEDIUM or PROPVAL_COMPLVL_HIGH. By default it is set to PROPVAL_COMPLVL_NONE.</li> </ul>

**Table 49–16 (Cont.) SETSTOREPROPERTY Procedure Parameters**

Parameter	Description
(cont) property_ value	<p><b>Stores of type STORETYPE_AMAZONS3</b></p> <p>It is mandatory that the following properties have assigned values, and default values are provided:</p> <ul style="list-style-type: none"> <li>■ PROPNAME_SBTLIBRARY - Specifies the path of the shared library used by RMAN to communicate with Amazon S3. It is named <code>libosbws11.so</code> and is available in <code>rdbms/lib</code> directory.</li> <li>■ PROPNAME_S3HOST - Defines the HOST name of the Amazon S3 storage service. It must be <code>s3.amazonaws.com</code>.</li> <li>■ PROPNAME_BUCKET - Defines the AWS bucket used as a storage tier by the Hierarchical Store. Restrictions on bucket names are: <ul style="list-style-type: none"> <li>-- Bucket names can only contain lowercase letters, numbers, periods (.) and dashes (-). Use of an underscore (_) is invalid.</li> <li>-- Bucket names must start with a number or letter</li> <li>-- Bucket names cannot be in an IP address style ("192.168.5.4")</li> <li>-- Bucket names must be between 3 and 63 characters in length</li> <li>-- Bucket names should not end with a dash</li> <li>-- Dashes cannot appear next to periods. For example, "my-.bucket.com" is invalid.</li> </ul> </li> <li>■ PROPNAME_LICENSEID - Specifies the license ID associated with the library <code>libosbws11.so</code>.</li> <li>■ PROPNAME_WALLET - Has the form: <pre>'LOCATION=file:&lt;filename&gt; CREDENTIAL_ ALIAS=&lt;access/secret_alias&gt; PROXY_AUTH_ ALIAS=&lt;proxyusername/password alias&gt;'</pre> <ul style="list-style-type: none"> <li>-- LOCATION - Directory path that contains the Oracle wallet. The format is <code>file:directory-path</code>. See <a href="#">Examples</a> for variations in format.</li> <li>-- PASSWORD - Defines the wallet password. If auto-login is enabled in the wallet (this can be changed using the user's own utility), and does not have to be specified. By default, the <code>mkstore</code> utility enables auto-login.</li> <li>-- CREDENTIAL_ALIAS - Defines the credential alias for ACCESS_KEY and SECRET_KEY</li> <li>-- PROXY_AUTH_ALIAS - Defines authentication credentials for the proxy server, if applicable.</li> </ul> </li> </ul>



**Table 49–16 (Cont.) SETSTOREPROPERTY Procedure Parameters**

Parameter	Description
(property_value (contd.))	<p>The following properties are optional:</p> <ul style="list-style-type: none"> <li>■ PROPNAME_HTTPPROXY - Defines the DNS name of the HTTP proxy, if any, that is needed to access the Amazon S3 storage service.</li> <li>■ PROPNAME_STREAMABLE – Indicates whether buffer-based PUT or GET operation should be done on this store. Valid values for this property are TRUE (default) and FALSE.</li> <li>■ PROPNAME_ENABLECLEANUPONDELETE - Indicates if unused files on the external storage device should be deleted by DBMS_DBFS_HS. Valid values for this property are 'FALSE' for STORETYPE_TAPE.</li> <li>■ PROPNAME_COMPRESSLEVEL - Describes how files written to Tape should be compressed. It can be set to PROPVAL_COMPLVL_NONE, PROPVAL_COMPLVL_LOW, PROPVAL_COMPLVL_MEDIUM or PROPVAL_COMPLVL_HIGH. By default it is set to PROPVAL_COMPLVL_NONE.</li> </ul>

## Usage Notes

- The specified store must already have been created.
- If this subprogram successfully executes, its actions cannot be rolled back by the user.
- The same property can be set multiple times to the same or different values using this subprogram

## Examples

### Format

The format of `wallet_path` in Windows is, for example:

```
file:c:\WINNT\Profiles\\WALLETS
```

The format of `wallet_path` in UNIX or Linux is, for example:

```
file:/home/username/wallets
```

## STOREPUSH Procedure

This procedure pushes locally staged data to the remote storage.

### Syntax

```
DBMS_DBFS_HS.STOREPUSH (
  store_name  IN      VARCHAR2,
  path        IN      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 49–17 STOREPUSH Procedure Parameters**

Parameter	Description
store_name	Name of store whose content the client writes from local cache to the external store
path	A non-mount qualified (without mount point) path within the store. By default, its value is NULL which corresponds to the root path of the store.

### Usage Notes

- The Hierarchical Store caches the content files locally in database tables. When enough content is amassed in the cache to make it efficient to write to the external storage device (or the cache is completely filled), the Hierarchical Store creates a tarball out of the local content and writes these tarballs as files on the external device. The size of the tarball created by the Hierarchical Store is controlled by the store property `PROPNAME_OPTTARBALLSIZE`.
- When the amount of free space in the cache is such that the caching of a content file will push the space used above `cache_size`, the Hierarchical Store will internally call `STOREPUSH`. The `STOREPUSH` Procedure creates tarball(s) out of the existing dirty or modified content files in the cache and writes them out to the external device. A `STOREPUSH` call is not guaranteed to write all the dirty content from local cache to the external storage, since some files may be locked by other sessions.
- `STOREPUSH` has a built-in ability feature allowing it to automatically resume operation. If a `STOREPUSH` call is interrupted (say by a network outage) after it has transferred some tarballs to the external device, it can be restarted after the outage and will then resume transferring data from the point it was interrupted. In other words, work done before the outage is not lost. `STOREPUSH` can safely be restarted and the effect is such as if the outage never occurred.
- If this method successfully executes, its actions cannot be rolled back by the user.
- By default, when `path` is NULL, all files in the store are candidates for `STOREPUSH`. If `path` has a valid input value, all files which are under the namespace of given path are written from the local cache to the external store. If a given path is an existing file, it is pushed out again to the remote store.

---

---

## DBMS\_DBFS\_SFS

The DBMS\_DBFS\_SFS package provides an interface to operate a SecureFile-based store (SFS) for the content interface described in the DBMS\_DBFS\_CONTENT package.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_DBFS\\_SFS](#)
  - Overview
  - Security Model
  - Constants
- [Summary of DBMS\\_DBFS\\_SFS Subprograms](#)

## Using DBMS\_DBFS\_SFS

- [Overview](#)
- [Security Model](#)
- [Constants](#)

## Overview

The DBMS\_DBFS\_SFS package is a sample implementation of a package that implements and extends the DBMS\_DBFS\_CONTENT\_SFI interface. It provides a POSIX-compliant file system stored in the RDBMS.

## Security Model

The `DBMS_DBFS_SFS` package runs with `AUTHID CURRENT_USER`.

## Constants

The DBMS\_DBFS\_SFS package uses the constants shown in following tables:

- [DBMS\\_DBFS\\_SFS Constants - Compression Levels](#)
- [DBMS\\_DBFS\\_SFS Constants - Used by the encryption Parameter](#)
- [DBMS\\_DBFS\\_SFS Constants - Used by the npartitions Parameter](#)
- [DBMS\\_DBFS\\_SFS Constants - Used by the partition\\_key Parameter](#)

**Table 50–1 DBMS\_DBFS\_SFS Constants - Compression Levels**

Constant	Type	Value	Description
COMPRESSION_ DEFAULT	VARCHAR2 (32)	' '	Use the default SecureFile compression level
COMPRESSION_LOW	VARCHAR2 (32)	'LOW'	Use compression level 'LOW'
COMPRESSION_ MEDIUM	VARCHAR2 (32)	'MEDIUM'	Use compression level 'MEDIUM'
COMPRESSION_HIGH	VARCHAR2 (32)	'HIGH'	Use compression level 'HIGH'

**Table 50–2 DBMS\_DBFS\_SFS Constants - Used by the encryption Parameter**

Constant	Type	Value	Description
ENCRYPTION_ DEFAULT	VARCHAR2 (32)	' '	Use the default SecureFile encryption algorithm
ENCRYPTION_ 3DES168	VARCHAR2 (32)	'3DES168'	Use encryption 3DES 168 bit
ENCRYPTION_ AES128	VARCHAR2 (32)	'AES128'	Use encryption AES 128 bit
ENCRYPTION_ AES192	VARCHAR2 (32)	'AES192'	Use encryption AES 192 bit
ENCRYPTION_ AES256	VARCHAR2 (32)	'AES256'	Use encryption AES 256 bit

**Table 50–3 DBMS\_DBFS\_SFS Constants - Used by the npartitions Parameter**

Constant	Type	Value	Description
DEFAULT_ PARTITIONS	INTEGER	16	Default to 16 partitions

**Table 50–4 DBMS\_DBFS\_SFS Constants - Used by the partition\_key Parameter**

Constant	Type	Value	Description
PARTITION_BY_ ITEM	INTEGER	1	Use a hash of the item name for the partition key
PARTITION_BY_ PATH	INTEGER	2	Use a hash of the path name for the partition key
PARTITION_BY_ GUID	INTEGER	3	Use a hash of the GUID as the partition key

## Summary of DBMS\_DBFS\_SFS Subprograms

**Table 50–5** *DBMS\_DBFS\_SFS Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">CREATEFILESYSTEM Procedure</a> on page 50-7	Creates a file system store
<a href="#">CREATESTORE Procedure</a> on page 50-9	Creates a new DBFS SFS store
<a href="#">DROPFILESYSTEM Procedures</a> on page 50-10	Drops the DBFS SFS store
<a href="#">INITFS Procedure</a> on page 50-11	Initializes a POSIX file system store



## CREATEFILESYSTEM Procedure

This procedure creates a file system store.

### Syntax

```
DBMS_DBFS_SFS.CREATEFILESYSTEM (
  store_name          IN      VARCHAR2,
  schema_name        IN      VARCHAR2  DEFAULT NULL,
  tbl_name           IN      VARCHAR2  DEFAULT NULL,
  tbl_tbs           IN      VARCHAR2  DEFAULT NULL,
  lob_tbs           IN      VARCHAR2  DEFAULT NULL,
  use_bf            IN      BOOLEAN   DEFAULT FALSE,
  properties         IN      DBMS_DBFS_CONTENT_PROPERTIES_T DEFAULT NULL,
  create_only       IN      BOOLEAN   FALSE,
  use_objects       IN      BOOLEAN   DEFAULT FALSE,
  with_grants      IN      BOOLEAN   DEFAULT FALSE,
  do_dedup         IN      BOOLEAN   DEFAULT FALSE,
  do_compress      IN      BOOLEAN   DEFAULT FALSE,
  compression      IN      VARCHAR2  DEFAULT COMPRESSION_DEFAULT,
  do_encrypt       IN      BOOLEAN   DEFAULT FALSE,
  encryption       IN      VARCHAR2  DEFAULT ENCRYPTION_DEFAULT,
  do_partition     IN      BOOLEAN   DEFAULT FALSE,
  npartitions      IN      NUMBER     DEFAULT DEFAULT_PARTITIONS,
  partition_key    IN      NUMBER     DEFAULT PARTITION_BY_ITEM,
  partition_guidi  IN      BOOLEAN   DEFAULT FALSE,
  partition_pathi  IN      BOOLEAN   DEFAULT FALSE,
  partition_prop   IN      BOOLEAN   DEFAULT TRUE);
```

### Parameters

**Table 50-6 CREATEFILESYSTEM Procedure Parameters**

Parameter	Description
store_name	Name of store
schema_name	Schema for the store, defaulting to the current schema
tbl_name	Table for store entries. If not specified, an internally generated name is used.
tbl_tbs	Tablespace for the store, defaulting to the schema's default tablespace
lob_tbs	Tablespace in which to create the LOB segment. It defaults to the user's default tablespace.
use_bf	If TRUE, a BasicFile LOB is used; otherwise a SecureFile LOB is used.
properties	Table of (name, value, typecode) tuples used to configure the store properties. Currently no such properties are defined or used.
create_only	If TRUE, the file system is created, but not registered with the current user
use_objects	If TRUE, a single base-table with an object-type column (using a nested table) is created to backup the new file system. Otherwise, a pair of (parent, child) tables is used to backup the file system. In both cases, the object type nested table or the child table is used only for user-defined properties.

**Table 50–6 (Cont.) CREATEFILESYSTEM Procedure Parameters**

Parameter	Description
with_grants	If TRUE, DML and query access permissions are granted to the DBFS_ROLE as part of creating the file system. Otherwise, explicit grants (or existing permissions) are required to access the file system.
do_dedup	If TRUE, do deduplication the underlying SecureFile column
do_compress	If TRUE, do compression the underlying SecureFile column
compression	Compression algorithm to use (see <a href="#">DBMS_DBFS_SFS Constants - Compression Levels</a> )
do_encrypt	If TRUE, encrypt the underlying SecureFile column
encryption	encryption algorithm to use (see <a href="#">DBMS_DBFS_SFS Constants - Used by the encryption Parameter</a> )
do_partition	If TRUE, partition the table used for storage
npartitions	Number of partitions to create for the table (see <a href="#">DBMS_DBFS_SFS Constants - Used by the npartitions Parameter</a> ).
partition_key	How to partition the table: by item name, by path name, or by GUID (see <a href="#">DBMS_DBFS_SFS Constants - Used by the partition_key Parameter</a> ).
partition_guidi	If TRUE, build an index on GUID
partition_pathi	If TRUE, build an index on path name
partition_prop	If TRUE, partition the properties table

## Usage Notes

The procedure executes like a DDL in that it auto-commits before and after its execution.

## CREATESTORE Procedure

This procedure creates a new DBFS SFS store owned by the invoking session user.

### Syntax

```
DBMS_DBFS_SFS.CREATESTORE (
  store_name      IN      VARCHAR2,
  tbl_name       IN      VARCHAR2 DEFAULT NULL,
  tbs_name       in      VARCHAR2 DEFAULT NULL,
  use_bf        in      BOOLEAN  DEFAULT FALSE,
  stgopts       in      VARCHAR2 DEFAULT '');
```

### Parameters

**Table 50–7 CREATESTORE Procedure Parameters**

Parameter	Description
store_name	Name of store
store_type	STORETYPE_TAPE or STORETYPE_AMAZONS3
tbl_name	Placeholder for the store content cached in database
tbs_name	Named tablespace
use_bf	If TRUE, a BasicFile LOB is used; otherwise a SecureFile LOB is used.
stgopts	Currently non-operational, reserved for future use

## DROPFILESYSTEM Procedures

This procedure drops the DBFS SFS store, purging all dictionary information associated with the store, and dropping the underlying file system table.

### Syntax

```
DBMS_DBFS_SFS.DROPFILESYSTEM (
  schema_name  IN      VARCHAR2 DEFAULT NULL,
  tbl_name     IN      INTEGER);

DBMS_DBFS_SFS.DROPFILESYSTEM (
  store_name   IN      VARCHAR2);
```

### Parameters

**Table 50–8** *DROPFILESYSTEM Procedure Parameters*

Parameter	Description
schema_name	Name of schema
tbl_name	Name of tablespace
store_name	Name of store path

### Usage Notes

- If the specified store table is registered by the current user, it will be unregistered from the content interface described in the `DBMS_DBFS_CONTENT` package and the POSIX metadata tables.
- Subsequent to unregistration, an attempt will be made to store table(s). This operation may fail if other users are currently using this store table.
- The user attempting a drop of the tables underlying the store must actually have the privileges to complete the drop operation (either as the owner of the tables, or as a sufficiently privileged user for cross-schema operations).
- The procedure executes like a DDL in that it auto-commits before and after its execution.

## INITFS Procedure

This procedure initialize a POSIX file system store. The table associated with the POSIX file system store `store_name` is truncated and reinitialized with a single "root" directory entry.

### Syntax

```
DBMS_DBFS_SFS.INITFS (  
    store_name      IN      VARCHAR2);
```

### Parameters

**Table 50–9** *INITFS Procedure Parameters*

Parameter	Description
<code>store_name</code>	Name of store

### Usage Notes

The procedure executes like a DDL in that it auto-commits before and after its execution.



---

---

## DBMS\_DB\_VERSION

The DBMS\_DB\_VERSION package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.

**See Also:** *PL/SQL Users Guide and Reference* regarding conditional compilation

This package contains the following topics

- [Using DBMS\\_DB\\_VERSION](#)
  - Overview
  - Constants
  - Examples

## Using DBMS\_DB\_VERSION

- [Overview](#)
- [Constants](#)



## Overview

The DBMS\_DB\_VERSION package specifies the Oracle version numbers and other information useful for simple conditional compilation selections based on Oracle versions.

The package for the Oracle Database 11g Release 1 version is shown below.

```
PACKAGE DBMS_DB_VERSION IS
  VERSION CONSTANT PLS_INTEGER := 11; -- RDBMS version number
  RELEASE CONSTANT PLS_INTEGER := 1; -- RDBMS release number
  ver_le_9_1    CONSTANT BOOLEAN := FALSE;
  ver_le_9_2    CONSTANT BOOLEAN := FALSE;
  ver_le_9      CONSTANT BOOLEAN := FALSE;
  ver_le_10_1   CONSTANT BOOLEAN := FALSE;
  ver_le_10_2   CONSTANT BOOLEAN := FALSE;
  ver_le_10     CONSTANT BOOLEAN := FALSE;
  ver_le_11_1   CONSTANT BOOLEAN := TRUE;
  ver_le_11     CONSTANT BOOLEAN := TRUE;
END DBMS_DB_VERSION;
```

The boolean constants follow a naming convention. Each constant gives a name for a boolean expression. For example:

- VER\_LE\_9\_1 represents version <= 9 and release <= 1
- VER\_LE\_10\_2 represents version <= 10 and release <= 2
- VER\_LE\_10 represents version <= 10

A typical usage of these boolean constants is:

```
$IF DBMS_DB_VERSION.VER_LE_10 $THEN
  version 10 and earlier code
$ELSIF DBMS_DB_VERSION.VER_LE_11 $THEN
  version 11 code
$ELSE
  version 12 and later code
$END
```

This code structure will protect any reference to the code for hypothetical version 12. It also prevents the controlling package constant DBMS\_DB\_VERSION.VER\_LE\_11 from being referenced when the program is compiled under version 10. A similar observation applies to version 11. This scheme works even though the static constant VER\_LE\_11 is not defined in version 10 database because conditional compilation protects the \$ELSIF from evaluation if DBMS\_DB\_VERSION.VER\_LE\_10 is TRUE.

## Constants

The `DBMS_DB_VERSION` package contains different constants for different Oracle Database releases. The Oracle Database 11g Release 1 version of the `DBMS_DB_VERSION` package uses the constants shown in [Table 51-1](#).

**Table 51-1** *DBMS\_DB\_VERSION Constants*

Name	Type	Value	Description
<code>VERSION</code>	<code>PLS_INTEGER</code>	10	Current version
<code>RELEASE</code>	<code>PLS_INTEGER</code>	2	Current release
<code>VER_LE_9</code>	<code>BOOLEAN</code>	<code>FALSE</code>	Version <= 9
<code>VER_LE_9_1</code>	<code>BOOLEAN</code>	<code>FALSE</code>	Version <= 9 and release <= 1
<code>VER_LE_9_2</code>	<code>BOOLEAN</code>	<code>FALSE</code>	Version <= 9 and release <= 2
<code>VER_LE_10</code>	<code>BOOLEAN</code>	<code>TRUE</code>	Version <= 10
<code>VER_LE_10_1</code>	<code>BOOLEAN</code>	<code>FALSE</code>	Version <= 10 and release <= 1
<code>VER_LE_10_2</code>	<code>BOOLEAN</code>	<code>TRUE</code>	Version <=10 and release <= 2
<code>VER_LE_11</code>	<code>BOOLEAN</code>	<code>FALSE</code>	Version <= 11
<code>VER_LE_11_1</code>	<code>BOOLEAN</code>	<code>TRUE</code>	Version <=11 and release <= 1

## Examples

This example uses conditional compilation to guard new features.

```

CREATE OR REPLACE PROCEDURE whetstone IS

  -- Notice that conditional compilation constructs
  -- can interrupt a regular PL/SQL statement.
  -- You can locate a conditional compilation directive anywhere
  -- there is whitespace in the regular statement.

  SUBTYPE my_real IS
    $IF DBMS_DB_VERSION.VER_LE_9 $THEN NUMBER
                                $ELSE BINARY_DOUBLE
    $END;

  t  CONSTANT my_real := $IF DBMS_DB_VERSION.VER_LE_9 $THEN 0.499975
                                $ELSE 0.499975d
    $END;

  t2 CONSTANT my_real := $if DBMS_DB_VERSION.VER_LE_9 $THEN 2.0
                                $ELSE 2.0d
    $END;

  x  CONSTANT my_real := $IF DBMS_DB_VERSION.VER_LE_9 $THEN 1.0
                                $ELSE 1.0d
    $END;

  y  CONSTANT my_real := $IF DBMS_DB_VERSION.VER_LE_9 $THEN 1.0
                                $ELSE 1.0d
    $END;

  z  MY_REAL;

  PROCEDURE P(x IN my_real, y IN my_real, z OUT NOCOPY my_real) IS
    x1 my_real;
    y1 my_real;
  BEGIN
    x1 := x;
    y1 := y;
    x1 := t * (x1 + y1);
    y1 := t * (x1 + y1);
    z := (x1 + y1)/t2;
  END P;
BEGIN
  P(x, y, z);
  DBMS_OUTPUT.PUT_LINE ('z = ' || z);
END whetstone;
/

```



DBMS\_DEBUG is a PL/SQL interface to the PL/SQL debugger layer, Probe, in the Oracle server.

This API is primarily intended to implement server-side debuggers and it provides a way to debug server-side PL/SQL program units.

---

---

**Note:** The term *program unit* refers to a PL/SQL program of any type (procedure, function, package, package body, trigger, anonymous block, object type, or object type body).

---

---

This chapter contains the following topics:

- [Using DBMS\\_DEBUG](#)
  - Overview
  - Constants
  - Variables
  - Exceptions
  - Operational Notes
- [Data Structures](#)
  - RECORD Types
  - TABLE Types
- [Summary of DBMS\\_DEBUG Subprograms](#)

## Using DBMS\_DEBUG

- [Overview](#)
- [Constants](#)
- [Variables](#)
- [Exceptions](#)
- [Operational Notes](#)

## Overview

To debug server-side code, you must have two database sessions: one session to run the code in debug mode (the target session), and a second session to supervise the target session (the debug session).

The target session becomes available for debugging by making initializing calls with `DBMS_DEBUG`. This marks the session so that the PL/SQL interpreter runs in debug mode and generates debug events. As debug events are generated, they are posted from the session. In most cases, debug events require return notification: the interpreter pauses awaiting a reply.

Meanwhile, the debug session must also initialize itself using `DBMS_DEBUG`: This tells it which target session to supervise. The debug session may then call entry points in `DBMS_DEBUG` to read events that were posted from the target session and to communicate with the target session.

The following subprograms are run in the target session (the session that is to be debugged):

- [SYNCHRONIZE Function](#)
- [DEBUG\\_ON Procedure](#)
- [DEBUG\\_OFF Procedure](#)

`DBMS_DEBUG` does not provide an interface to the PL/SQL compiler, but it does depend on debug information optionally generated by the compiler. Without debug information, it is not possible to examine or modify the values of parameters or variables.

## Constants

A breakpoint status may have the following value:

- `breakpoint_status_unused`—breakpoint is not in use

Otherwise, the status is a mask of the following values:

- `breakpoint_status_active`—a line breakpoint
- `breakpoint_status_disabled`—breakpoint is currently disabled
- `breakpoint_status_remote`—a shadow breakpoint (a local representation of a remote breakpoint)



## Variables

The DBMS\_DEBUG uses the variables shown in [Table 52-1](#).

**Table 52-1 DBMS\_DEBUG Variables**

Variable	Description
default_timeout	The timeout value (used by both sessions).The smallest possible timeout is 1 second. If this value is set to 0, then a large value (3600) is used.

## Exceptions

These values are returned by the various functions called in the debug session (SYNCHRONIZE, CONTINUE, SET\_BREAKPOINT, and so on). If PL/SQL exceptions worked across client/server and server/server boundaries, then these would all be exceptions rather than error codes.

Value	Description
success	Normal termination

Statuses returned by GET\_VALUE and SET\_VALUE:

Status	Description
error_bogus_frame	No such entypoint on the stack
error_no_debug_info	Program was compiled without debug symbols
error_no_such_object	No such variable or parameter
error_unknown_type	Debug information is unreadable
error_indexed_table	Returned by GET_VALUE if the object is a table, but no index was provided
error_illegal_index	No such element exists in the collection
error_nullcollection	Table is atomically NULL
error_nullvalue	Value is NULL

Statuses returned by SET\_VALUE:

Status	Description
error_illegal_value	Constraint violation
error_illegal_null	Constraint violation
error_value_malformed	Unable to decipher the given value
error_other	Some other error
error_name_incomplete	Name did not resolve to a scalar

Statuses returned by the breakpoint functions:

Status	Description
error_no_such_breakpt	No such breakpoint
error_idle_breakpt	Cannot enable or disable an unused breakpoint
error_bad_handle	Unable to set breakpoint in given program (nonexistent or security violation)

General error codes (returned by many of the DBMS\_DEBUG subprograms):

<b>Status</b>	<b>Description</b>
error_unimplemented	Functionality is not yet implemented
error_deferred	No program running; operation deferred
error_exception	An exception was raised in the DBMS_DEBUG or Probe packages on the server
error_communication	Some error other than a timeout occurred
error_timeout	Timeout occurred

<b>Exception</b>	<b>Description</b>
illegal_init	DEBUG_ON was called prior to INITIALIZE

The following exceptions are raised by procedure SELF\_CHECK:

<b>Exception</b>	<b>Description</b>
pipe_creation_failure	Could not create a pipe
pipe_send_failure	Could not write data to the pipe
pipe_receive_failure	Could not read data from the pipe
pipe_datatype_mismatch	Datatype in the pipe was wrong
pipe_data_error	Data got garbled in the pipe

## Operational Notes

There are two ways to ensure that debug information is generated: through a session switch, or through individual recompilation.

To set the session switch, enter the following statement:

```
ALTER SESSION SET PLSQL_DEBUG = true;
```

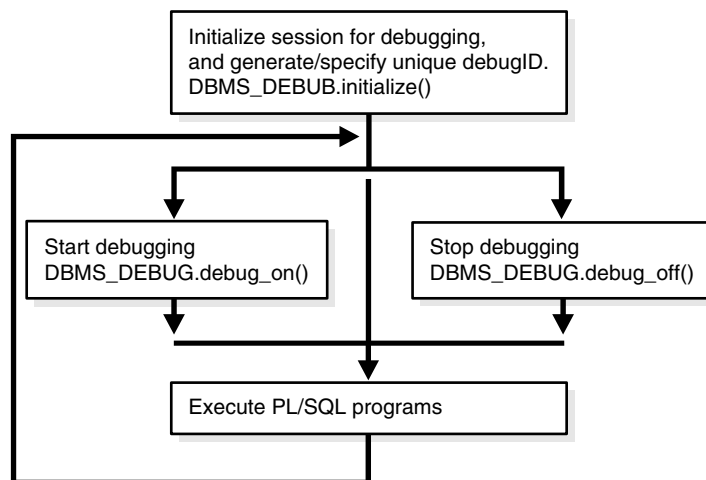
This instructs the compiler to generate debug information for the remainder of the session. It does not recompile any existing PL/SQL.

To generate debug information for existing PL/SQL code, use one of the following statements (the second recompiles a package or type body):

```
ALTER [PROCEDURE | FUNCTION | PACKAGE | TRIGGER | TYPE] <name> COMPILE DEBUG;
ALTER [PACKAGE | TYPE] <name> COMPILE DEBUG BODY;
```

Figure 52–1 and Figure 52–2 illustrate the flow of operations in the session to be debugged and in the debugging session.

**Figure 52–1 Target Session**

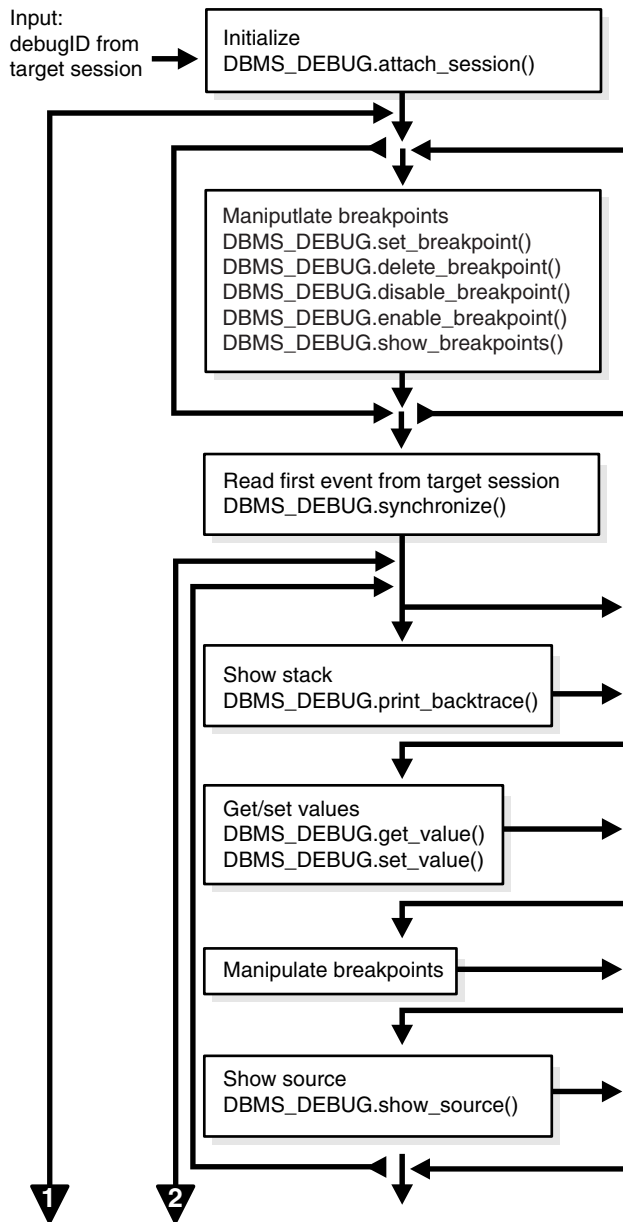


This illustration shows the first steps in the target session:

First, initialize the session for debugging, and generate or specify the unique debugID. `DBMS_DEBUG.initialize()`

Then, start debugging using `DBMS_DEBUG.debug_on()` or stop debugging using `DBMS_DEBUG.debug_off()`. Then, execute the PL/SQL program.

\*\*\*\*\*

**Figure 52-2 Debug Session**

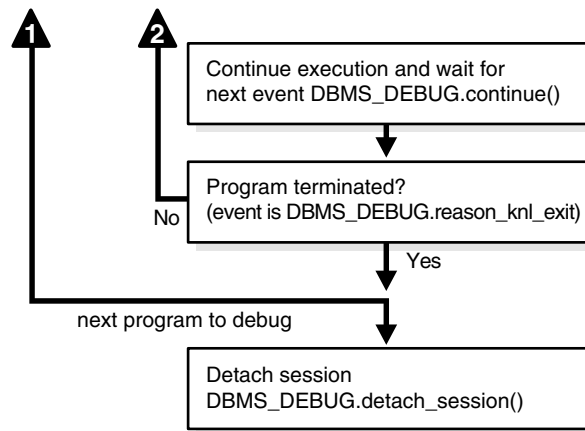
This illustration shows the continuation of steps in the target session:

- Initialize using `DBMS_DEBUG.attach_session()`
- Manipulate the breakpoints using:
  - `DBMS_DEBUG.set_breakpoint()`
  - `DBMS_DEBUG.delete_breakpoint()`
  - `DBMS_DEBUG.disable_breakpoint()`
  - `DBMS_DEBUG.enable_breakpoint()`
  - `DBMS_DEBUG.show_breakpoint()`
- Read first event from the target session using `DBMS_DEBUG.synchronize()`
- Show the stack using `DBMS_DEBUG.print_backtrace()`

- Get/set values using:
  - DBMS\_DEBUG.get\_value()
  - 
  - DBMS\_DEBUG.set\_value()
- Manipulate the breakpoints
- Show the source by using DBMS\_DEBUG.show\_source().

\*\*\*\*\*

**Figure 52–3 Debug Session (Cont.)**



This illustration shows the final steps in the target session:

- Continue execution and wait for the next event DBMS\_DEBUG.continue()
- If the program terminated, detach the session using DBMS\_DEBUG.detach\_session()
- If it did not terminate, resume the process by showing the stack.

\*\*\*\*\*

## Control of the Interpreter

The interpreter pauses execution at the following times:

1. At startup of the interpreter so any deferred breakpoints may be installed prior to execution.
2. At any line containing an enabled breakpoint.
3. At any line where an *interesting* event occurs. The set of interesting events is specified by the flags passed to DBMS\_DEBUG.CONTINUE in the *breakflags* parameter.

## Session Termination

There is no event for session termination. Therefore, it is the responsibility of the debug session to check and make sure that the target session has not ended. A call to DBMS\_DEBUG.SYNCHRONIZE after the target session has ended causes the debug session to hang until it times out.

## Deferred Operations

The diagram suggests that it is possible to set breakpoints prior to having a target session. This is true. In this case, Probe caches the breakpoint request and transmits it to the target session at first synchronization. However, if a breakpoint request is deferred in this fashion, then:

- SET\_BREAKPOINT does not set the breakpoint number (it can be obtained later from SHOW\_BREAKPOINTS if necessary).
- SET\_BREAKPOINT does not validate the breakpoint request. If the requested source line does not exist, then an error silently occurs at synchronization, and no breakpoint is set.

## Diagnostic Output

To debug Probe, there are *diagnostics* parameters to some of the calls in DBMS\_DEBUG. These parameters specify whether to place diagnostic output in the RDBMS tracefile. If output to the RDBMS tracefile is disabled, these parameters have no effect.

## Common and Debug Session Sections

- [Common Section](#)
- [Target Session](#)
- [Debug Session Section](#)

### Common Section

The following subprograms may be called in either the target or the debug session:

- [PROBE\\_VERSION Procedure](#)
- [SELF\\_CHECK Procedure](#)
- [SET\\_TIMEOUT Function](#)

### Target Session

The following subprograms may be called only in the target session:

- [INITIALIZE Function](#)
- [DEBUG\\_ON Procedure](#)
- [SET\\_TIMEOUT\\_BEHAVIOUR Procedure](#)
- [GET\\_TIMEOUT\\_BEHAVIOUR Function](#)

### Debug Session Section

The following subprograms should be run in the debug session only:

- [ATTACH\\_SESSION Procedure](#)
- [SYNCHRONIZE Function](#)
- [SHOW\\_FRAME\\_SOURCE Procedure](#)
- [SHOW\\_SOURCE Procedures](#)
- [GET\\_MORE\\_SOURCE Procedure](#)
- [PRINT\\_BACKTRACE Procedure](#)
- [CONTINUE Function](#)
- [SET\\_BREAKPOINT Function](#)

- [DELETE\\_BREAKPOINT Function](#)
- [SET\\_OER\\_BREAKPOINT Function](#)
- [DELETE\\_OER\\_BREAKPOINT Function](#)
- [ENABLE\\_BREAKPOINT Function](#)
- [DISABLE\\_BREAKPOINT Function](#)
- [SHOW\\_BREAKPOINTS Procedures](#)
- [SET\\_VALUE Functionn](#)
- [GET\\_VALUE Function](#)
- [TARGET\\_PROGRAM\\_RUNNING Procedure](#)
- [DETACH\\_SESSION Procedure](#)
- [GET\\_RUNTIME\\_INFO Function](#)
- [PRINT\\_INSTANTIATIONS Procedure](#)
- [PING Procedure](#)
- [GET\\_LINE\\_MAP Function](#)
- [GET\\_RUNTIME\\_INFO Function](#)
- [GET\\_INDEXES Function](#)
- [EXECUTE Procedure](#)

## OER Breakpoints

Exceptions that are declared in PL/SQL programs are known as user-defined exceptions. In addition, there are Oracle Errors (OERs) that are returned from the Oracle kernel. To tie the two mechanisms together, PL/SQL provides the `exception_init` pragma that turns a user-defined exception into an OER, so that a PL/SQL handler may be used for it, and so that the PL/SQL engine can return OERs to the Oracle kernel. As of the current release, the only information available about an OER is its number. If two user-defined exceptions are `exception_init'd` to the same OER, they are indistinguishable.

## Namespaces

Program units on the server reside in different namespaces. When setting a breakpoint, specify the desired namespace.

1. `Namespace_cursor` contains cursors (anonymous blocks).
2. `Namespace_pgkspec_or_toplevel` contains:
  - Package specifications.
  - Procedures and functions that are not nested inside other packages, procedures, or functions.
  - Object types.
3. `Namespace_pkg_body` contains package bodies and type bodies.
4. `Namespace_trigger` contains triggers.



## Libunit Types

These values are used to disambiguate among objects in a given namespace. These constants are used in PROGRAM\_INFO when Probe is giving a stack backtrace.

- LibunitType\_cursor
- LibunitType\_procedure
- LibunitType\_function
- LibunitType\_package
- LibunitType\_package\_body
- LibunitType\_trigger
- LibunitType\_Unknown

## Breakflags

These are values to use for the breakflags parameter to CONTINUE, in order to tell Probe what events are of interest to the client. These flags may be combined.

Value	Description
break_next_line	Break at next source line (step over calls)
break_any_call	Break at next source line (step into calls)
break_any_return	Break after returning from current entrypoint (skip over any entrypoints called from the current routine)
break_return	Break the next time an entrypoint gets ready to return. (This includes entrypoints called from the current one. If interpreter is running Proc1, which calls Proc2, then break_return stops at the end of Proc2.)
break_exception	Break when an exception is raised
break_handler	Break when an exception handler is executed
abort_execution	Stop execution and force an 'exit' event as soon as DBMS_DEBUG.CONTINUE is called.

## Information Flags

These are flags which may be passed as the info\_requested parameter to SYNCHRONIZE, CONTINUE, and GET\_RUNTIME\_INFO.

Flag	Description
info_getStackDepth	Get the current depth of the stack
info_getBreakpoint	Get the breakpoint number
info_getLineinfo	Get program unit information

## Reasons for Suspension

After CONTINUE is run, the program either runs to completion or breaks on some line.

Reason	Description
reason_none	-

<b>Reason</b>	<b>Description</b>
reason_interpreter_starting	Interpreter is starting
reason_breakpoint	Hit a breakpoint
reason_enter	Procedure entry
reason_return	Procedure is about to return
reason_finish	Procedure is finished
reason_line	Reached a new line
reason_interrupt	An interrupt occurred
reason_exception	An exception was raised
reason_exit	Interpreter is exiting (old form)
reason_knl_exit	Kernel is exiting
reason_handler	Start exception-handler
reason_timeout	A timeout occurred
reason_instantiate	Instantiation block
reason_abort	Interpreter is aborting

## Data Structures

The DBMS\_DEBUG package defines RECORD types and TABLE types.

### RECORD Types

- [BREAKPOINT\\_INFO](#) Record Type
- [PROGRAM\\_INFO](#) Record Type
- [RUNTIME\\_INFO](#) Record Type

### TABLE Types

- [BACKTRACE\\_TABLE](#) Table Type
- [BREAKPOINT\\_TABLE](#) Table Type
- [INDEX\\_TABLE](#) Table Type
- [VC2\\_TABLE](#) Table Type

## BREAKPOINT\_INFO Record Type

This type gives information about a breakpoint, such as its current status and the program unit in which it was placed.

### Syntax

```
TYPE breakpoint_info IS RECORD (  
    name          VARCHAR2(30),  
    owner         VARCHAR2(30),  
    dblink        VARCHAR2(30),  
    line#         BINARY_INTEGER,  
    libunittype   BINARY_INTEGER,  
    status        BINARY_INTEGER);
```

### Fields

**Table 52–2** BREAKPOINT\_INFO Fields

Field	Description
name	Name of the program unit
owner	Owner of the program unit
dblink	Database link, if remote
line#	Line number
libunittype	NULL, unless this is a nested procedure or function
status	See <a href="#">Constants</a> on page 52-4 for values of breakpoint_status_*

## PROGRAM\_INFO Record Type

This type specifies a program location. It is a line number in a program unit. This is used for stack backtraces and for setting and examining breakpoints. The read-only fields are currently ignored by Probe for breakpoint operations. They are set by Probe only for stack backtraces.

### Syntax

```
TYPE program_info IS RECORD(
  -- The following fields are used when setting a breakpoint
  namespace      BINARY_INTEGER,
  name            VARCHAR2(30),
  owner          VARCHAR2(30),
  dblink         VARCHAR2(30),
  line#          BINARY_INTEGER,
  -- Read-only fields (set by Probe when doing a stack backtrace)
  libunittype    BINARY_INTEGER,
  entrypointname VARCHAR2(30));
```

### Fields

**Table 52-3 PROGRAM\_INFO Fields**

Field	Description
namespace	See <a href="#">Namespaces</a> on page 52-12
name	Name of the program unit
owner	Owner of the program unit
dblink	Database link, if remote
line#	Line number
libunittype	A read-only field, NULL, unless this is a nested procedure or function
entrypointname	A read-only field, to disambiguate among objects that share the same namespace (for example, procedure and package specifications). See the <a href="#">Libunit Types</a> on page 52-13 for more information.

## RUNTIME\_INFO Record Type

This type gives context information about the running program.

### Syntax

```
TYPE runtime_info IS RECORD(  
    line#           BINARY_INTEGER,  
    terminated     binary_integer,  
    breakpoint     binary_integer,  
    stackdepth    BINARY_INTEGER,  
    interpreterdepth BINARY_INTEGER,  
    reason        BINARY_INTEGER,  
    program       program_info);
```

### Fields

**Table 52–4** *RUNTIME\_INFO Fields*

Field	Description
line#	Duplicate of program.line#
terminated	Whether the program has terminated
breakpoint	Breakpoint number
stackdepth	Number of frames on the stack
interpreterdepth	[A reserved field]
reason	Reason for suspension
program	Source location

## **BACKTRACE\_TABLE Table Type**

This type is used by PRINT\_BACKTRACE.

### **Syntax**

```
TYPE backtrace_table IS TABLE OF program_info INDEX BY BINARY_INTEGER;
```

## **BREAKPOINT\_TABLE Table Type**

This type is used by SHOW\_BREAKPOINTS.

### **Syntax**

```
TYPE breakpoint_table IS TABLE OF breakpoint_info INDEX BY BINARY_INTEGER;
```



## INDEX\_TABLE Table Type

This type is used by `GET_INDEXES` to return the available indexes for an indexed table.

### Syntax

```
TYPE index_table IS table of BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

## VC2\_TABLE Table Type

This type is used by SHOW\_SOURCE.

### Syntax

```
TYPE vc2_table IS TABLE OF VARCHAR2(90) INDEX BY BINARY_INTEGER;
```

## Summary of DBMS\_DEBUG Subprograms

**Table 52–5 DBMS\_DEBUG Package Subprograms**

Subprogram	Description
<a href="#">ATTACH_SESSION Procedure</a> on page 52-25	Notifies the debug session about the target debugID
<a href="#">CONTINUE Function</a> on page 52-26	Continues execution of the target program
<a href="#">DEBUG_OFF Procedure</a> on page 27	Turns debug-mode off
<a href="#">DEBUG_ON Procedure</a> on page 52-28	Turns debug-mode on
<a href="#">DELETE_BREAKPOINT Function</a> on page 52-29	Deletes a breakpoint
<a href="#">DELETE_OER_BREAKPOINT Function</a> on page 52-30	Deletes an OER breakpoint
<a href="#">DETACH_SESSION Procedure</a> on page 52-31	Stops debugging the target program
<a href="#">DISABLE_BREAKPOINT Function</a> on page 52-32	Disables a breakpoint
<a href="#">ENABLE_BREAKPOINT Function</a> on page 52-33	Activates an existing breakpoint
<a href="#">EXECUTE Procedure</a> on page 52-34	Executes SQL or PL/SQL in the target session
<a href="#">GET_INDEXES Function</a> on page 52-36	Returns the set of indexes for an indexed table
<a href="#">GET_MORE_SOURCE Procedure</a> on page 52-37	Provides additional source in the event of buffer overflow when using SHOW_SOURCE
<a href="#">GET_LINE_MAP Function</a> on page 52-38	Returns information about line numbers in a program unit
<a href="#">GET_RUNTIME_INFO Function</a> on page 52-39	Returns information about the current program
<a href="#">GET_TIMEOUT_BEHAVIOUR Function</a> on page 52-40	Returns the current timeout behavior
<a href="#">GET_VALUE Function</a> on page 52-41	Gets a value from the currently-running program
<a href="#">INITIALIZE Function</a> on page 52-43	Sets debugID in target session
<a href="#">PING Procedure</a> on page 52-45	Pings the target session to prevent it from timing out
<a href="#">PRINT_BACKTRACE Procedure</a> on page 52-46	Prints a stack backtrace
<a href="#">PRINT_INSTANTIATIONS Procedure</a> on page 52-47	Prints a stack backtrace
<a href="#">PROBE_VERSION Procedure</a> on page 52-48	Returns the version number of DBMS_DEBUG on the server

**Table 52–5 (Cont.) DBMS\_DEBUG Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SELF_CHECK Procedure</a> on page 52-49	Performs an internal consistency check
<a href="#">SET_BREAKPOINT Function</a> on page 52-50	Sets a breakpoint in a program unit
<a href="#">SET_OER_BREAKPOINT Function</a> on page 52-51	Sets an OER breakpoint
<a href="#">SET_TIMEOUT Function</a> on page 52-52	Sets the timeout value
<a href="#">SET_TIMEOUT_BEHAVIOUR Procedure</a> on page 52-53	Tells Probe what to do with the target session when a timeout occurs
<a href="#">SET_VALUE Function</a> on page 52-54	Sets a value in the currently-running program
<a href="#">SHOW_BREAKPOINTS Procedures</a> on page 52-56	Returns a listing of the current breakpoints
<a href="#">SHOW_FRAME_SOURCE Procedure</a> on page 52-57	Fetches the frame source
<a href="#">SHOW_SOURCE Procedures</a> on page 52-58	Fetches program source
<a href="#">SYNCHRONIZE Function</a> on page 52-60	Waits for program to start running
<a href="#">TARGET_PROGRAM_RUNNING Procedure</a> on page 52-61	Returns TRUE if the target session is currently executing a stored procedure, or FALSE if it is not

## ATTACH\_SESSION Procedure

This procedure notifies the debug session about the target program.

### Syntax

```
DBMS_DEBUG.ATTACH_SESSION (  
    debug_session_id IN VARCHAR2,  
    diagnostics      IN BINARY_INTEGER := 0);
```

### Parameters

**Table 52–6** *ATTACH\_SESSION Procedure Parameters*

Parameter	Description
debug_session_id	Debug ID from a call to INITIALIZE in target session
diagnostics	Generate diagnostic output if nonzero

## CONTINUE Function

This function passes the given breakflags (a mask of the events that are of interest) to Probe in the target process. It tells Probe to continue execution of the target process, and it waits until the target process runs to completion or signals an event.

If `info_requested` is not `NULL`, then calls `GET_RUNTIME_INFO`.

### Syntax

```
DBMS_DEBUG.CONTINUE (
    run_info      IN OUT runtime_info,
    breakflags   IN      BINARY_INTEGER,
    info_requested IN      BINARY_INTEGER := NULL)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–7** CONTINUE Function Parameters

Parameter	Description
<code>run_info</code>	Information about the state of the program
<code>breakflags</code>	Mask of events that are of interest (see " <a href="#">Breakflags</a> " on page 52-13)
<code>info_requested</code>	Which information should be returned in <code>run_info</code> when the program stops (see " <a href="#">Information Flags</a> " on page 52-13)

### Return Values

**Table 52–8** CONTINUE Function Return Values

Return	Description
<code>success</code>	
<code>error_timeout</code>	Timed out before the program started running
<code>error_communication</code>	Other communication error

## DEBUG\_OFF Procedure

---

---

**Caution:** There must be a debug session waiting if immediate is TRUE.

---

---

This procedure notifies the target session that debugging should no longer take place in that session. It is not necessary to call this function before ending the session.

### Syntax

```
DBMS_DEBUG.DEBUG_OFF;
```

### Usage Notes

The server does not handle this entrypoint specially. Therefore, it attempts to debug this entrypoint.

## DEBUG\_ON Procedure

This procedure marks the target session so that all PL/SQL is run in debug mode. This must be done before any debugging can take place.

### Syntax

```
DBMS_DEBUG.DEBUG_ON (  
    no_client_side_plsql_engine BOOLEAN := TRUE,  
    immediate                   BOOLEAN := FALSE);
```

### Parameters

**Table 52–9** *DEBUG\_ON Procedure Parameters*

Parameter	Description
<code>no_client_side_plsql_engine</code>	Should be left to its default value unless the debugging session is taking place from a client-side PL/SQL engine
<code>immediate</code>	If this is <code>TRUE</code> , then the interpreter immediately switches itself into debug-mode, instead of continuing in regular mode for the duration of the call.



## DELETE\_BREAKPOINT Function

This function deletes a breakpoint.

### Syntax

```
DBMS_DEBUG.DELETE_BREAKPOINT (
    breakpoint IN BINARY_INTEGER)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–10** *DELETE\_BREAKPOINT Function Parameters*

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT

### Return Values

**Table 52–11** *DELETE\_BREAKPOINT Function Return Values*

Return	Description
success	
error_no_such_breakpt	No such breakpoint exists
error_idle_breakpt	Cannot delete an unused breakpoint
error_stale_breakpt	The program unit was redefined since the breakpoint was set

## DELETE\_OER\_BREAKPOINT Function

This function deletes an OER breakpoint.

### Syntax

```
DBMS_DEBUG.DELETE_OER_BREAKPOINT (  
    oer IN PLS_INTEGER)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 52–12** *DELETE\_OER\_BREAKPOINT Function Parameters*

Parameter	Description
oer	The OER (positive 4-byte number) to delete

## **DETACH\_SESSION Procedure**

This procedure stops debugging the target program. This procedure may be called at any time, but it does not notify the target session that the debug session is detaching itself, and it does not terminate execution of the target session. Therefore, care should be taken to ensure that the target session does not hang itself.

### **Syntax**

```
DBMS_DEBUG.DETACH_SESSION;
```

## DISABLE\_BREAKPOINT Function

This function makes an existing breakpoint inactive but leaves it in place.

### Syntax

```
DBMS_DEBUG.DISABLE_BREAKPOINT (  
    breakpoint IN BINARY_INTEGER)  
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–13** *DISABLE\_BREAKPOINT Function Parameters*

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT

### Return Values

**Table 52–14** *DISABLE\_BREAKPOINT Function Return Values*

Returns	Description
success	
error_no_such_ breakpt	No such breakpoint exists
error_idle_breakpt	Cannot disable an unused breakpoint

## ENABLE\_BREAKPOINT Function

This function is the reverse of disabling. This enables a previously disabled breakpoint.

### Syntax

```
DBMS_DEBUG.ENABLE_BREAKPOINT (
    breakpoint IN BINARY_INTEGER)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–15** *ENABLE\_BREAKPOINT Function Parameters*

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT

### Return Values

**Table 52–16** *ENABLE\_BREAKPOINT Function Return Values*

Return	Description
success	Success
error_no_such_breakpt	No such breakpoint exists
error_idle_breakpt	Cannot enable an unused breakpoint

## EXECUTE Procedure

This procedure executes SQL or PL/SQL code in the target session. The target session is assumed to be waiting at a breakpoint (or other event). The call to `DBMS_DEBUG.EXECUTE` occurs in the debug session, which then asks the target session to execute the code.

### Syntax

```
DBMS_DEBUG.EXECUTE (
  what          IN VARCHAR2,
  frame#        IN BINARY_INTEGER,
  bind_results  IN BINARY_INTEGER,
  results       IN OUT NOCOPY dbms_debug_vc2coll,
  errm          IN OUT NOCOPY VARCHAR2);
```

### Parameters

**Table 52–17 EXECUTE Procedure Parameters**

Parameter	Description
<code>what</code>	SQL or PL/SQL source to execute
<code>frame#</code>	The context in which to execute the code. Only -1 (global context) is supported at this time.
<code>bind_results</code>	Whether the source wants to bind to <code>results</code> in order to return values from the target session: 0 = No 1 = Yes
<code>results</code>	Collection in which to place results, if <code>bind_results</code> is not 0
<code>errm</code>	Error message, if an error occurred; otherwise, NULL

### Examples

#### Example 1

This example executes a SQL statement. It returns no results.

```
DECLARE
  coll sys.dbms_debug_vc2coll; -- results (unused)
  errm VARCHAR2(100);
BEGIN
  dbms_debug.execute('insert into emp(ename,empno,deptno) ' ||
                    'values(''LJE'', 1, 1)',
                    -1, 0, coll, errm);
END;
```

#### Example 2

This example executes a PL/SQL block, and it returns no results. The block is an autonomous transaction, which means that the value inserted into the table becomes visible in the debug session.

```
DECLARE
  coll sys.dbms_debug_vc2coll;
  errm VARCHAR2(100);
BEGIN
```

```

dbms_debug.execute(
  'DECLARE PRAGMA autonomous_transaction; ' ||
  'BEGIN ' ||
  '  insert into emp(ename, empno, deptno) ' ||
  '  values(''LJE'', 1, 1); ' ||
  ' COMMIT; ' ||
  'END;',
  -1, 0, coll, errm);
END;

```

### Example 3

This example executes a PL/SQL block, and it returns some results.

```

DECLARE
  coll sys.dbms_debug_vc2coll;
  errm VARCHAR2(100);
BEGIN
  dbms_debug.execute(
    'DECLARE ' ||
    '  pp SYS.dbms_debug_vc2coll := SYS.dbms_debug_vc2coll(); ' ||
    '  x PLS_INTEGER; ' ||
    '  i PLS_INTEGER := 1; ' ||
    'BEGIN ' ||
    '  SELECT COUNT(*) INTO x FROM emp; ' ||
    '  pp.EXTEND(x * 6); ' ||
    '  FOR c IN (SELECT * FROM emp) LOOP ' ||
    '    pp(i) := ''Ename: '' || c.ename; i := i+1; ' ||
    '    pp(i) := ''Empno: '' || c.empno; i := i+1; ' ||
    '    pp(i) := ''Job:   '' || c.job;   i := i+1; ' ||
    '    pp(i) := ''Mgr:   '' || c.mgr;   i := i+1; ' ||
    '    pp(i) := ''Sal:   '' || c.sal;   i := i+1; ' ||
    '    pp(i) := null;          i := i+1; ' ||
    '  END LOOP; ' ||
    '  :1 := pp; ' ||
    'END;',
    -1, 1, coll, errm);
  each := coll.FIRST;
  WHILE (each IS NOT NULL) LOOP
    dosomething(coll(each));
    each := coll.NEXT(each);
  END LOOP;
END;

```

## GET\_INDEXES Function

Given a name of a variable or parameter, this function returns the set of its indexes, if it is an indexed table. An error is returned if it is not an indexed table.

### Syntax

```
DBMS_DEBUG.GET_INDEXES (
    varname   IN  VARCHAR2,
    frame#    IN  BINARY_INTEGER,
    handle    IN  program_info,
    entries   OUT index_table)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–18** GET\_INDEXES Function Parameters

Parameter	Description
varname	Name of the variable to get index information about
frame#	Number of frame in which the variable or parameter resides; NULL for a package variable
handle	Package description, if object is a package variable
entries	1-based table of the indexes: if non-NULL, then entries(1) contains the first index of the table, entries(2) contains the second index, and so on.

### Return Values

**Table 52–19** GET\_INDEXES Function Return Values

Return	Description
error_no_such_object	One of the following: <ul style="list-style-type: none"> <li>- The package does not exist</li> <li>- The package is not instantiated</li> <li>- The user does not have privileges to debug the package</li> <li>- The object does not exist in the package</li> </ul>



## GET\_MORE\_SOURCE Procedure

When source does not fit in the buffer provided by that version of the [SHOW\\_SOURCE Procedures](#) which produce a formatted buffer, this procedure provides additional source.

### Syntax

```
DBMS_DEBUG.GET_MORE_SOURCE (
  buffer          IN OUT VARCHAR2,
  buflen         IN BINARY_INTEGER,
  piece#         IN BINARY_INTEGER);
```

### Parameters

**Table 52–20** *GET\_MORE\_SOURCE Procedure Parameters*

Parameter	Description
buffer	The buffer
buflen	The length of the buffer
piece#	A value between 2 and the value returned in the parameter pieces from the call to the relevant version of the <a href="#">SHOW_SOURCE Procedures</a>

### Usage Notes

This procedure should be called only after the version of SHOW\_SOURCE that returns a formatted buffer.

## GET\_LINE\_MAP Function

This function finds line and entrypoint information about a program so that a debugger can determine the source lines at which it is possible to place breakpoints.

### Syntax

```
DBMS_DEBUG.GET_LINE_MAP (
  program          IN  program_info,
  maxline          OUT BINARY_INTEGER,
  number_of_entry_points OUT BINARY_INTEGER,
  linemap         OUT  RAW)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–21** GET\_LINE\_MAP Function Parameters

Parameter	Description
program	A top-level program unit (procedure / package / function / package body, and so on). Its Namespace, Name, and Owner fields must be initialized, the remaining fields are ignored.
maxline	The largest source code line number in 'program'
number_of_entry_points	The number of subprograms in 'program'
linemap	A bitmap representing the executable lines of 'program'. If line number N is executable, bit number N MOD 8 will be set to 1 at linemap position N / 8. The length of returned linemap is either maxline divided by 8 (plus one if maxline MOD 8 is not zero) or 32767 in the unlikely case of maxline being larger than 32767 * 8.

### Return Values

**Table 52–22** GET\_LINE\_MAP Function Return Values

Return	Description
success	A successful completion
error_no_debug_info	The program unit exists, but has no debug info
error_bad_handle	No such program unit exists

## GET\_RUNTIME\_INFO Function

This function returns information about the current program. It is only needed if the `info_requested` parameter to `SYNCHRONIZE` or `CONTINUE` was set to 0.

---



---

**Note:** This is currently only used by client-side PL/SQL.

---



---

### Syntax

```
DBMS_DEBUG.GET_RUNTIME_INFO (
  info_requested IN BINARY_INTEGER,
  run_info      OUT runtime_info)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–23** *GET\_RUNTIME\_INFO Function Parameters*

Parameter	Description
<code>info_requested</code>	Which information should be returned in <code>run_info</code> when the program stops (see " <a href="#">Information Flags</a> " on page 52-13)
<code>run_info</code>	Information about the state of the program

## GET\_TIMEOUT\_BEHAVIOUR Function

This procedure returns the current timeout behavior. This call is made in the target session.

### Syntax

```
DBMS_DEBUG.GET_TIMEOUT_BEHAVIOUR
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–24** GET\_TIMEOUT\_BEHAVIOUR Function Parameters

Parameter	Description
oer	The OER (a 4-byte positive number)

### Return Values

**Table 52–25** GET\_TIMEOUT\_BEHAVIOUR Function Return Values

Return	Description
success	A successful completion

### Information Flags

```
info_getOerInfo CONSTANT PLS_INTEGER:= 32;
```

### Usage Notes

Less functionality is supported on OER breakpoints than on code breakpoints. In particular, note that:

- No "breakpoint number" is returned - the number of the OER is used instead. Thus it is impossible to set duplicate breakpoints on a given OER (it is a no-op).
- It is not possible to disable an OER breakpoint (although clients are free to simulate this by deleting it).
- OER breakpoints are deleted using `delete_oer_breakpoint`.

## GET\_VALUE Function

This function gets a value from the currently-running program. There are two overloaded GET\_VALUE functions.

### Syntax

```
DBMS_DEBUG.GET_VALUE (
  variable_name IN VARCHAR2,
  frame#       IN BINARY_INTEGER,
  scalar_value OUT VARCHAR2,
  format      IN VARCHAR2 := NULL)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–26** GET\_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter
frame#	Frame in which it lives; 0 means the current procedure
scalar_value	Value
format	Optional date format to use, if meaningful

### Return Values

**Table 52–27** GET\_VALUE Function Return Values

Return	Description
success	A successful completion
error_bogus_frame	Frame does not exist
error_no_debug_info	Entrypoint has no debug information
error_no_such_object	variable_name does not exist in frame#
error_unknown_type	The type information in the debug information is illegible
error_nullvalue	Value is NULL
error_indexed_table	The object is a table, but no index was provided

This form of GET\_VALUE is for fetching package variables. Instead of a frame#, it takes a handle, which describes the package containing the variable.

### Syntax

```
DBMS_DEBUG.GET_VALUE (
  variable_name IN VARCHAR2,
  handle       IN program_info,
  scalar_value OUT VARCHAR2,
  format      IN VARCHAR2 := NULL)
RETURN BINARY_INTEGER;
```

## Parameters

**Table 52–28** GET\_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter
handle	Description of the package containing the variable
scalar_value	Value
format	Optional date format to use, if meaningful

## Return Values

**Table 52–29** GET\_VALUE Function Return Values

Return	Description
error_no_such_object	One of the following: <ul style="list-style-type: none"> <li>- Package does not exist</li> <li>- Package is not instantiated</li> <li>- User does not have privileges to debug the package</li> <li>- Object does not exist in the package</li> </ul>
error_indexed_table	The object is a table, but no index was provided

## Examples

This example illustrates how to get the value with a given package PACK in schema SCOTT, containing variable VAR:

```

DECLARE
    handle    dbms_debug.program_info;
    resultbuf VARCHAR2(500);
    retval    BINARY_INTEGER;
BEGIN
    handle.Owner      := 'SCOTT';
    handle.Name       := 'PACK';
    handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
    retval           := dbms_debug.get_value('VAR', handle, resultbuf, NULL);
END;
```

## INITIALIZE Function

This function initializes the target session for debugging.

### Syntax

```
DBMS_DEBUG.INITIALIZE (
  debug_session_id IN VARCHAR2      := NULL,
  diagnostics      IN BINARY_INTEGER := 0)
RETURN VARCHAR2;
```

### Parameters

**Table 52–30 INITIALIZE Function Parameters**

Parameter	Description
debug_session_id	Name of session ID. If NULL, then a unique ID is generated.
diagnostics	Indicates whether to dump diagnostic output to the tracefile: 0 = (default) no diagnostics 1 = print diagnostics

### Return Values

The newly-registered debug session ID (debugID)

### Usage Notes

You cannot use DBMS\_DEBUG and the JDWP-based debugging interface simultaneously. This call will either fail with an ORA-30677 error if the session is currently being debugged with the JDWP-based debugging interface or, if the call succeeds, any further use of the JDWP-based interface to debug this session will be disallowed.

Calls to DBMS\_DEBUG will succeed only if either the caller or the specified debug role carries the DEBUG CONNECT SESSION privilege. Failing that, an ORA-1031 error will be raised. Other exceptions are also possible if a debug role is specified but the password does not match, or if the calling user has not been granted the role, or the role is application-enabled and this call does not originate from within the role-enabling package.

The CREATE ANY PROCEDURE privilege does not affect the visibility of routines through the debugger. A privilege DEBUG for each object has been introduced with a corresponding DEBUG ANY PROCEDURE variant. These are required in order to see routines owned by users other than the session's login user.

Authentication of the debug role and the check for DEBUG CONNECT SESSION privilege will be done in the context of the caller to this routine. If the caller is a definer's rights routine or has been called from one, only privileges granted to the defining user, the debug role, or PUBLIC will be used to check for DEBUG CONNECT SESSION. If this call is from within a definer's rights routine, the debug role, if specified, must be one that has been granted to that definer, but it need not also have been granted to the session login user or be enabled in the calling session at the time the call is made.

The checks made by the debugger after this call is made looking for the DEBUG privilege on individual procedures will be done in the context of the session's login user, the roles that were enabled at session level at the moment this call was made

(even if those roles were not available within a definer's rights environment of the call), and the debug role.



## PING Procedure

This procedure pings the target session to prevent it from timing out. Use this procedure when execution is suspended in the target session, for example at a breakpoint.

If the `timeout_behaviour` is set to `retry_on_timeout` then this procedure is not necessary.

### Syntax

```
DBMS_DEBUG.PING;
```

### Exceptions

Oracle will display the `no_target_program` exception if there is no target program or if the target session is not currently waiting for input from the debug session.

### Usage Notes

Timeout options for the target session are registered with the target session by calling `set_timeout_behaviour`:

- `retry_on_timeout` - Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
- `continue_on_timeout` - Continue execution, using same event flags.
- `nodebug_on_timeout` - Turn debug-mode OFF (in other words, call `debug_off`) and then continue execution. No more events will be generated by this target session unless it is re-initialized by calling `debug_on`.
- `abort_on_timeout` - Continue execution, using the `abort_execution` flag, which should cause the program to terminate immediately. The session remains in debug-mode.

```
retry_on_timeout CONSTANT BINARY_INTEGER:= 0;  
continue_on_timeout CONSTANT BINARY_INTEGER:= 1;  
nodebug_on_timeout CONSTANT BINARY_INTEGER:= 2;  
abort_on_timeout CONSTANT BINARY_INTEGER:= 3;
```

## PRINT\_BACKTRACE Procedure

This procedure prints a backtrace listing of the current execution stack. This should only be called if a program is currently running.

There are two overloaded PRINT\_BACKTRACE procedures.

### Syntax

```
DBMS_DEBUG.PRINT_BACKTRACE (  
    listing IN OUT VARCHAR2);  
  
DBMS_DEBUG.PRINT_BACKTRACE (  
    backtrace OUT backtrace_table);
```

### Parameters

**Table 52–31 PRINT\_BACKTRACE Procedure Parameters**

Parameter	Description
listing	A formatted character buffer with embedded newlines
backtrace	1-based indexed table of backtrace entries. The currently-running procedure is the last entry in the table (that is, the frame numbering is the same as that used by GET_VALUE). Entry 1 is the oldest procedure on the stack.

## PRINT\_INSTANTIATIONS Procedure

This procedure returns a list of the packages that have been instantiated in the current session.

### Syntax

```
DBMS_DEBUG.PRINT_INSTANTIATIONS (
  pkgs   IN OUT NOCOPY backtrace_table,
  flags  IN BINARY_INTEGER);
```

### Parameters

**Table 52–32 PRINT\_INSTANTIATIONS Procedure Parameters**

Parameter	Description
pkgs	The instantiated packages
flags	Bitmask of options: <ul style="list-style-type: none"> <li>■ 1 - show specs</li> <li>■ 2 - show bodies</li> <li>■ 4 - show local instantiations</li> <li>■ 8 - show remote instantiations (NYI)</li> <li>■ 16 - do a fast job. The routine does not test whether debug information exists or whether the libunit is shrink-wrapped.</li> </ul>

### Exceptions

`no_target_program` - target session is not currently executing

### Usage Notes

On return, `pkgs` contains a `program_info` for each instantiation. The valid fields are: `Namespace`, `Name`, `Owner`, and `LibunitType`.

In addition, `Line#` contains a bitmask of:

- 1 - the libunit contains debug info
- 2 - the libunit is shrink-wrapped

## PROBE\_VERSION Procedure

This procedure returns the version number of DBMS\_DEBUG on the server.

### Syntax

```
DBMS_DEBUG.PROBE_VERSION (
    major out BINARY_INTEGER,
    minor out BINARY_INTEGER);
```

### Parameters

**Table 52-33** *PROBE\_VERSION Procedure Parameters*

Parameter	Description
major	Major version number
minor	Minor version number: increments as functionality is added

## SELF\_CHECK Procedure

This procedure performs an internal consistency check. SELF\_CHECK also runs a communications test to ensure that the Probe processes are able to communicate.

If SELF\_CHECK does not return successfully, then an incorrect version of DBMS\_DEBUG was probably installed on this server. The solution is to install the correct version (pblload.sql loads DBMS\_DEBUG and the other relevant packages).

### Syntax

```
DBMS_DEBUG.SELF_CHECK (
    timeout IN binary_integer := 60);
```

### Parameters

**Table 52–34 SELF\_CHECK Procedure Parameters**

Parameter	Description
timeout	The timeout to use for the communication test. Default is 60 seconds.

### Exceptions

**Table 52–35 SELF\_CHECK Procedure Exceptions**

Exception	Description
OER-6516	Probe version is inconsistent
pipe_creation_failure	Could not create a pipe
pipe_send_failure	Could not write data to the pipe
pipe_receive_failure	Could not read data from the pipe
pipe_datatype_mismatch	Datatype in the pipe was wrong
pipe_data_error	Data got garbled in the pipe

All of these exceptions are fatal. They indicate a serious problem with Probe that prevents it from working correctly.

## SET\_BREAKPOINT Function

This function sets a breakpoint in a program unit, which persists for the current session. Execution pauses if the target program reaches the breakpoint.

### Syntax

```
DBMS_DEBUG.SET_BREAKPOINT (
  program      IN program_info,
  line#        IN BINARY_INTEGER,
  breakpoint#  OUT BINARY_INTEGER,
  fuzzy        IN BINARY_INTEGER := 0,
  iterations   IN BINARY_INTEGER := 0)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–36** SET\_BREAKPOINT Function Parameters

Parameter	Description
program	Information about the program unit in which the breakpoint is to be set. (In version 2.1 and later, the namespace, name, owner, and dblink may be set to NULL, in which case the breakpoint is placed in the currently-running program unit.)
line#	Line at which the breakpoint is to be set
breakpoint#	On successful completion, contains the unique breakpoint number by which to refer to the breakpoint
fuzzy	Only applicable if there is no executable code at the specified line: 0 means return <code>error_illegal_line</code> 1 means search forward for an adjacent line at which to place the breakpoint -1 means search backward for an adjacent line at which to place the breakpoint
iterations	Number of times to wait before signalling this breakpoint

### Return Values

---



---

**Note:** The `fuzzy` and `iterations` parameters are not yet implemented

---



---

**Table 52–37** SET\_BREAKPOINT Function Return Values

Return	Description
success	A successful completion
error_illegal_line	Cannot set a breakpoint at that line
error_bad_handle	No such program unit exists

## SET\_OER\_BREAKPOINT Function

This function sets an OER breakpoint.

### Syntax

```
DBMS_DEBUG.SET_OER_BREAKPOINT (
    oer IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

### Parameters

**Table 52–38 SET\_OER\_BREAKPOINT Function Parameters**

Parameter	Description
oer	The OER (positive 4-byte number) to set

### Return Values

**Table 52–39 SET\_OER\_BREAKPOINT Function Return Values**

Return	Description
success	A successful completion
error_no_such_breakpt	No such OER breakpoint exists

## SET\_TIMEOUT Function

This function sets the timeout value and returns the new timeout value.

### Syntax

```
DBMS_DEBUG.SET_TIMEOUT (  
    timeout BINARY_INTEGER)  
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–40** *SET\_TIMEOUT Function Parameters*

Parameter	Description
timeout	The timeout to use for communication between the target and debug sessions



## SET\_TIMEOUT\_BEHAVIOUR Procedure

This procedure tells Probe what to do with the target session when a timeout occurs. This call is made in the target session.

### Syntax

```
DBMS_DEBUG.SET_TIMEOUT_BEHAVIOUR (
    behaviour IN PLS_INTEGER);
```

### Parameters

**Table 52–41 SET\_TIMEOUT\_BEHAVIOUR Procedure Parameters**

Parameter	Description
behaviour	One of the following:
retry_on_timeout	Retry. Timeout has no effect. This is like setting the timeout to an infinitely large value.
continue_on_timeout	Continue execution, using same event flags
nodebug_on_timeout	Turn debug-mode OFF (in other words, call <code>debug_off</code> ) and continue execution. No more events will be generated by this target session unless it is re-initialized by calling <code>debug_on</code> .
abort_on_timeout	Continue execution, using the <code>abort_execution</code> flag, which should cause the program to terminate immediately. The session remains in debug-mode.

### Exceptions

unimplemented - the requested behavior is not recognized

### Usage Notes

The default behavior (if this procedure is not called) is `continue_on_timeout`, since it allows a debugger client to reestablish control (at the next event) but does not cause the target session to hang indefinitely.

## SET\_VALUE Function

This function sets a value in the currently-running program. There are two overloaded SET\_VALUE functions.

### Syntax

```
DBMS_DEBUG.SET_VALUE (
    frame#           IN binary_integer,
    assignment_statement IN varchar2)
RETURN BINARY_INTEGER;
```

```
DBMS_DEBUG.SET_VALUE (
    handle           IN program_info,
    assignment_statement IN VARCHAR2)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–42 SET\_VALUE Function Parameters**

Parameter	Description
frame#	Frame in which the value is to be set; 0 means the currently executing frame.
handle	Description of the package containing the variable
assignment_statement	An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, 'x := 3;'. Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.

### Return Values

**Table 52–43 SET\_VALUE Function Return Values**

Return	Description
success	-
error_illegal_value	Not possible to set it to that value
error_illegal_null	Cannot set to NULL because object type specifies it as 'not NULL'
error_value_malformed	Value is not a scalar
error_name_incomplete	The assignment statement does not resolve to a scalar. For example, 'x := 3;', if x is a record.
error_no_such_object	One of the following: <ul style="list-style-type: none"> <li>- Package does not exist</li> <li>- Package is not instantiated</li> <li>- User does not have privileges to debug the package</li> <li>- Object does not exist in the package</li> </ul>

## Usage Notes

In some cases, the PL/SQL compiler uses temporaries to access package variables, and does not guarantee to update such temporaries. It is possible, although unlikely, that modification to a package variable using `SET_VALUE` might not take effect for a line or two.

## Examples

To set the value of `SCOTT.PACK.var` to 6:

```
DECLARE
    handle dbms_debug.program_info;
    retval BINARY_INTEGER;
BEGIN
    handle.Owner      := 'SCOTT';
    handle.Name       := 'PACK';
    handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
    retval            := dbms_debug.set_value(handle, 'var := 6;');
END;
```

## SHOW\_BREAKPOINTS Procedures

There are two overloaded procedures that return a listing of the current breakpoints. There are three overloaded SHOW\_BREAKPOINTS procedures.

### Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
    listing    IN OUT VARCHAR2);
```

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
    listing    OUT breakpoint_table);
```

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
    code_breakpoints OUT breakpoint_table,
    oer_breakpoints  OUT oer_table);
```

### Parameters

**Table 52-44** *SHOW\_BREAKPOINTS Procedure Parameters*

Parameter	Description
listing	A formatted buffer (including newlines) of the breakpoints.
	Indexed table of breakpoint entries. The breakpoint number is indicated by the index into the table. Breakpoint numbers start at 1 and are reused when deleted.
code_breakpoints	The indexed table of breakpoint entries, indexed by breakpoint number
oer_breakpoints	The indexed table of OER breakpoints, indexed by OER

## SHOW\_FRAME\_SOURCE Procedure

The procedure gets the source code. There are two overloaded SHOW\_SOURCE procedures.

### Syntax

```
DBMS_DEBUG.SHOW_FRAME_SOURCE (
  first_line IN          BINARY_INTEGER,
  last_line  IN          BINARY_INTEGER,
  source     IN OUT NOCOPY vc2_table,
  frame_num  IN          BINARY_INTEGER);
```

### Parameters

**Table 52–45** SHOW\_FRAME\_SOURCE Procedure Parameters

Parameter	Description
first_line	Line number of first line to fetch (PL/SQL programs always start at line 1 and have no holes)
last_line	Line number of last line to fetch. No lines are fetched past the end of the program.
source	The resulting table, which may be indexed by line#
frame_num	1-based frame number

### Usage Notes

- You use this function only when backtrace shows an anonymous unit is executing at a given frame position and you need to view the source in order to set a breakpoint.
- If frame number is top of the stack and it's an anonymous block then SHOW\_SOURCE can also be used.
- If it's a stored PLSQL package/function/procedure then use SQL as described in the [Usage Notes](#) to [SHOW\\_SOURCE Procedures](#).

## SHOW\_SOURCE Procedures

The procedure gets the source code. There are two overloaded SHOW\_SOURCE procedures.

### Syntax

```
DBMS_DEBUG.SHOW_SOURCE (
    first_line IN BINARY_INTEGER,
    last_line  IN BINARY_INTEGER,
    source     OUT vc2_table);

DBMS_DEBUG.SHOW_SOURCE (
    first_line IN BINARY_INTEGER,
    last_line  IN BINARY_INTEGER,
    window     IN BINARY_INTEGER,
    print_arrow IN BINARY_INTEGER,
    buffer     IN OUT VARCHAR2,
    buflen     IN BINARY_INTEGER,
    pieces     OUT BINARY_INTEGER);
```

### Parameters

**Table 52–46** SHOW\_SOURCE Procedure Parameters

Parameter	Description
first_line	Line number of first line to fetch (PL/SQL programs always start at line 1 and have no holes)
last_line	Line number of last line to fetch. No lines are fetched past the end of the program.
source	The resulting table, which may be indexed by line#
window	'Window' of lines (the number of lines around the current source line)
print_arrow	Nonzero means to print an arrow before the current line
buffer	Buffer in which to place the source listing
buflen	Length of buffer
pieces	Set to nonzero if not all the source could be placed into the given buffer

### Return Values

An indexed table of source-lines. The source lines are stored starting at first\_line. If any error occurs, then the table is empty.

### Usage Notes

The best way to get the source code (for a program that is being run) is to use SQL. For example:

```
DECLARE
    info DBMS_DEBUG.runtime_info;
BEGIN
    -- call DBMS_DEBUG.SYNCHRONIZE, CONTINUE,
    -- or GET_RUNTIME_INFO to fill in 'info'
    SELECT text INTO <buffer> FROM all_source
```

```
WHERE owner = info.Program.Owner
      AND name = info.Program.Name
      AND line = info.Line#;
END;
```

However, this does not work for nonpersistent programs (for example, anonymous blocks and trigger invocation blocks). For nonpersistent programs, call `SHOW_SOURCE`. There are two flavors: one returns an indexed table of source lines, and the other returns a packed (and formatted) buffer.

The second overloading of `SHOW_SOURCE` returns the source in a formatted buffer, complete with line-numbers. It is faster than the indexed table version, but it does not guarantee to fetch all the source.

If the source does not fit in `bufferlength (bufLen)`, then additional pieces can be retrieved using the `GET_MORE_SOURCE` procedure (`pieces` returns the number of additional pieces that need to be retrieved).

## SYNCHRONIZE Function

This function waits until the target program signals an event. If `info_requested` is not NULL, then it calls `GET_RUNTIME_INFO`.

### Syntax

```
DBMS_DEBUG.SYNCHRONIZE (
  run_info          OUT runtime_info,
  info_requested IN  BINARY_INTEGER := NULL)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 52–47 SYNCHRONIZE Function Parameters**

Parameter	Description
<code>run_info</code>	Structure in which to write information about the program. By default, this includes information about what program is running and at which line execution has paused.
<code>info_requested</code>	Optional bit-field in which to request information other than the default (which is <code>info_getStackDepth + info_getLineInfo</code> ). 0 means that no information is requested at all (see "Information Flags" on page 52-13).

### Return Values

**Table 52–48 SYNCHRONIZE Function Return Values**

Return	Description
<code>success</code>	A successful completion
<code>error_timeout</code>	Timed out before the program started execution
<code>error_communication</code>	Other communication error



## TARGET\_PROGRAM\_RUNNING Procedure

This procedure returns `TRUE` if the target session is currently executing a stored procedure, or `FALSE` if it is not.

### Syntax

```
DBMS_DEBUG.TARGET_PROGRAM_RUNNING  
RETURN BOOLEAN;
```



This package provides access to some SQL data definition language (DDL) statements from stored procedures. It also provides special administration operations that are not available as Data Definition Language statements (DDLs).

This chapter contains the following topics:

- [Using DBMS\\_DDL](#)
  - Deprecated Subprograms
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_DDL Subprograms](#)

## Using DBMS\_DDL

This section contains topics which relate to using the DBMS\_DDL package.

- [Deprecated Subprograms](#)
- [Security Model](#)
- [Operational Notes](#)

## Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only

The following subprograms are deprecated with release Release 10gR2:

- [ALTER\\_COMPILE Procedure](#)

## Security Model

This package runs with the privileges of the calling user, rather than the package owner `SYS`.

## Operational Notes

The `ALTER_COMPILE` procedure commits the current transaction, performs the operation, and then commits again.

---

## Summary of DBMS\_DDL Subprograms

**Table 53–1 DBMS\_DDL Package Subprograms**

Subprogram	Description
<a href="#">ALTER_COMPILE Procedure</a> on page 53-7	Compiles the PL/SQL object
<a href="#">ALTER_TABLE_NOT_REFERENCEABLE Procedure</a> on page 53-8	Reorganizes object tables
<a href="#">ALTER_TABLE_REFERENCEABLE Procedure</a> on page 53-9	Reorganizes object tables
<a href="#">CREATE_WRAPPED Procedures</a> on page 53-10	Takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body, generates a CREATE OR REPLACE statement with the PL/SQL source text obfuscated and executes the generated statement
<a href="#">IS_TRIGGER_FIRE_ONCE Function</a> on page 53-12	Returns TRUE if the specified DML or DDL trigger is set to fire once. Otherwise, returns FALSE
<a href="#">SET_TRIGGER_FIRING_PROPERTY Procedures</a> on page 53-13	Sets the specified DML or DDL trigger's firing property
<a href="#">WRAP Functions</a> on page 53-15	Takes as input a CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body and returns a CREATE OR REPLACE statement where the text of the PL/SQL unit has been obfuscated



## ALTER\_COMPILE Procedure

This procedure is equivalent to the following SQL statement:

```
ALTER PROCEDURE|FUNCTION|PACKAGE [<schema>.] <name> COMPILE [BODY]
```

---

**Note:** This procedure is deprecated in Release 10gR2. While the procedure remains available in the package, Oracle recommends using the DDL equivalent in a dynamic SQL statement.

---

### Syntax

```
DBMS_DDL.ALTER_COMPILE (
    type          VARCHAR2,
    schema        VARCHAR2,
    name          VARCHAR2
    reuse_settings BOOLEAN := FALSE);
```

### Parameters

**Table 53–2 ALTER\_COMPILE Procedure Parameters**

Parameter	Description
type	Must be either PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY or TRIGGER
schema	Schema name If NULL, then use current schema (case-sensitive)
name	Name of the object (case-sensitive)
reuse_settings	Indicates whether the session settings in the objects should be reused, or whether the current session settings should be adopted instead

### Exceptions

**Table 53–3 ALTER\_COMPILE Procedure Exceptions**

Exception	Description
ORA-20000:	Insufficient privileges or object does not exist
ORA-20001:	Remote object, cannot compile
ORA-20002:	Bad value for object type: should be either PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, or TRIGGER

## ALTER\_TABLE\_NOT\_REFERENCEABLE Procedure

This procedure alters the given object table `table_schema.table_name` so it becomes not the default referenceable table for the schema `affected_schema`. This is equivalent to SQL

```
ALTER TABLE [<table_schema>.<table_name> NOT REFERENCEABLE FOR <affected_schema>
```

which is currently not supported or available as a DDL statement.

### Syntax

```
DBMS_DDL.ALTER_TABLE_NOT_REFERENCEABLE (
    table_name      IN          VARCHAR2,
    table_schema    IN  DEFAULT NULL,
    affected_schema IN  DEFAULT NULL);
```

### Parameters

**Table 53–4 ALTER\_TABLE\_NOT\_REFERENCEABLE Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the table to be altered. Cannot be a synonym. Must not be NULL. Case sensitive.
<code>table_schema</code>	Name of the schema owning the table to be altered. If NULL then the current schema is used. Case sensitive.
<code>affected_schema</code>	Name of the schema affected by this alteration. If NULL then the current schema is used. Case sensitive.

### Usage Notes

This procedure simply reverts for the affected schema to the default table referenceable for PUBLIC; that is, it simply undoes the previous `ALTER_TABLE_REFERENCEABLE` call for this specific schema. The affected schema must a particular schema (cannot be PUBLIC).

The user that executes this procedure must own the table (that is, the schema is the same as the user), and the affected schema must be the same as the user.

If the user executing this procedure has `ALTER ANY TABLE` and `SELECT ANY TABLE` and `DROP ANY TABLE` privileges, the user doesn't have to own the table and the affected schema can be any valid schema.

## ALTER\_TABLE\_REFERENCEABLE Procedure

This procedure alters the given object table `table_schema.table_name` so it becomes the referenceable table for the given schema `affected_schema`. This is equivalent to SQL

```
ALTER TABLE [<table_schema>.<table_name>] REFERENCEABLE FOR <affected_schema>
```

which is currently not supported or available as a DDL statement.

### Syntax

```
DBMS_DDL.ALTER_TABLE_REFERENCEABLE
  table_name      IN  VARCHAR2,
  table_schema    IN  DEFAULT NULL,
  affected_schema IN  DEFAULT NULL);
```

### Parameters

**Table 53–5 ALTER\_TABLE\_REFERENCEABLE Procedure Parameters**

Parameter	Description
<code>table_name</code>	Name of the table to be altered. Cannot be a synonym. Must not be NULL. Case sensitive.
<code>table_schema</code>	Name of the schema owning the table to be altered. If NULL then the current schema is used. Case sensitive.
<code>affected_schema</code>	Name of the schema affected by this alteration. If NULL then the current schema is used. Case sensitive.

### Usage Notes

When you create an object table, it automatically becomes referenceable, unless you use the `OID AS` clause when creating the table. The `OID AS` clause makes it possible for you to create an object table and to assign to the new table the same EOID as another object table of the same type. After you create a new table using the `OID AS` clause, you end up with two object table with the same EOID; the new table is not referenceable, the original one is. All references that used to point to the objects in the original table still reference the same objects in the same original table.

If you execute this procedure on the new table, it makes the new table the referenceable table replacing the original one; thus, those references now point to the objects in the new table instead of the original table.

## CREATE\_WRAPPED Procedures

The procedure takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body. It then generates a CREATE OR REPLACE statement with the PL/SQL source text obfuscated and executes the generated statement. In effect, this procedure bundles together the operations of wrapping the text and creating the PL/SQL unit.

**See Also:** [WRAP Functions](#) on page 53-15

This procedure has 3 overloads. Each of the three functions provides better performance than using a combination of individual [WRAP Functions](#) and DBMS\_SQL.PARSE (or EXECUTE IMMEDIATE) calls. The different functionality of each form of syntax is presented with the definition.

### Syntax

Is a shortcut for EXECUTE IMMEDIATE SYS.DBMS\_DDL.WRAP(*ddl*):

```
DBMS_DDL.CREATE_WRAPPED (
    ddl    VARCHAR2);
```

Is a shortcut for DBMS\_SQL.PARSE(cursor, SYS.DBMS\_DDL.WRAP (input, lb, ub)):

```
DBMS_DDL.CREATE_WRAPPED (
    ddl    DBMS_SQL.VARCHAR2A,
    lb     PLS_INTEGER,
    ub     PLS_INTEGER);
```

Is a shortcut for DBMS\_SQL.PARSE(cursor, SYS.DBMS\_DDL.WRAP (input, lb, ub)):

```
DBMS_DDL.CREATE_WRAPPED (
    ddl    DBMS_SQL.VARCHAR2S,
    lb     PLS_INTEGER,
    ub     PLS_INTEGER);
```

### Parameters

**Table 53–6 CREATE\_WRAPPED Procedure Parameters**

Parameter	Description
ddl	A CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body
lb	Lower bound for indices in the string table that specify the CREATE OR REPLACE statement
ub	Upper bound for indices in the string table that specify the CREATE OR REPLACE statement.

### Usage Notes

- The CREATE OR REPLACE statement is executed with the privileges of the user invoking DBMS\_DDL.CREATE\_WRAPPED.
- Any PL/SQL code that attempts to call these interfaces should use the fully qualified package name SYS.DBMS\_DDL to avoid the possibility that the name

DBMS\_DDL is captured by a locally-defined unit or by redefining the DBMS\_DDL public synonym.

- Each invocation of any accepts only a single PL/SQL unit. By contrast, the PL/SQL `wrap` utility accepts a entire SQL\*Plus file and obfuscates the PL/SQL units within the file leaving all other text as-is. These interfaces are intended to be used in conjunction with or as a replacement for PL/SQL's dynamic SQL interfaces (`EXECUTE IMMEDIATE` and `DBMS_SQL.PARSE`). Since these dynamic SQL interfaces only accept a single unit at a time (and do not understand the SQL\*Plus `"/"` termination character), both the [CREATE\\_WRAPPED Procedures](#) and the [WRAP Functions](#) require input to be a single unit.

## Exceptions

ORA-24230: If the input is not a CREATE OR REPLACE statement specifying a PL/SQL unit, exception `DBMS_DDL.MALFORMED_WRAP_INPUT` is raised.

## Examples

```
DECLARE
    ddl VARCHAR2(32767);
BEGIN
    ddl := GENERATE_PACKAGE(...);
    SYS.DBMS_DDL.CREATE_WRAPPED(ddl); -- Instead of EXECUTE IMMEDIATE ddl
END;
```

## IS\_TRIGGER\_FIRE\_ONCE Function

This function returns `TRUE` if the specified DML or DDL trigger is set to fire once. Otherwise, it returns `FALSE`.

A fire once trigger fires in a user session but does not fire in the following cases:

- For changes made by a Streams apply process
- For changes made by executing one or more Streams apply errors using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package
- For changes made by a Logical Standby apply process

---



---

**Note:** Only DML and DDL triggers can be fire once. All other types of triggers always fire.

---



---

**See Also:** ["SET\\_TRIGGER\\_FIRING\\_PROPERTY Procedures"](#) on page 53-13

### Syntax

```
DBMS_DDL.IS_TRIGGER_FIRE_ONCE
  trig_owner      IN VARCHAR2,
  trig_name       IN VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

**Table 53–7 IS\_TRIGGER\_FIRE\_ONCE Function Parameters**

Parameter	Description
<code>trig_owner</code>	Schema of trigger
<code>trig_name</code>	Name of trigger

## SET\_TRIGGER\_FIRING\_PROPERTY Procedures

This procedure sets the specified DML or DDL trigger's firing property whether or not the property is set for the trigger. Use this procedure to control a DML or DDL trigger's firing property for changes:

- Applied by a Streams apply process
- Made by executing one or more Streams apply errors using the EXECUTE\_ERROR or EXECUTE\_ALL\_ERRORS procedure in the DBMS\_APPLY\_ADM package.
- Applied by a Logical Standby apply process

### Syntax

```
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY (
    trig_owner      IN  VARCHAR2,
    trig_name       IN  VARCHAR2,
    fire_once       IN  BOOLEAN);
```

```
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY (
    trig_owner      IN  VARCHAR2,
    trig_name       IN  VARCHAR2,
    property        IN  INTEGER,
    setting         IN  BOOLEAN);
```

### Parameters

**Table 53–8 SET\_TRIGGER\_FIRING\_PROPERTY Procedure Parameters**

Parameter	Description
trig_owner	Schema of the trigger to set
trig_name	Name of the trigger to set
fire_once	<ul style="list-style-type: none"> <li>■ If TRUE, the trigger is set to fire once. By default, the fire_once parameter is set to TRUE for DML and DDL triggers.</li> <li>■ If FALSE, the trigger is set to always fire unless apply_server_only property is set to TRUE, which overrides fire_once property setting.</li> </ul>
property	<ul style="list-style-type: none"> <li>■ DBMS_DDL.fire_once to set the fire_once property of the trigger</li> <li>■ DBMS_DDL.apply_server_only to indicate whether trigger fires only in the context of SQL apply processes maintaining a logical standby database or Streams apply processes</li> </ul>
setting	Value of property being set

### Usage Notes

DML triggers created on a table have their fire-once property set to TRUE. In this case, the triggers only fire when the table is modified by an user process, and they are automatically disabled inside Oracle processes maintaining either a logical standby database (SQL Apply) or Oracle processes doing replication (Streams Apply) processes, and thus do not fire when a SQL Apply or a Streams Apply process modifies the table. There are two ways for a user to fire a trigger as a result of SQL Apply or a Streams Apply process making a change to a maintained table: (a) setting the fire-once property of a trigger to FALSE, which allows it fire both in the context of a user process or a SQL or Streams Apply process, or (b) by setting the

apply-server-only property to TRUE and thus making the trigger fire only in the context of a SQL Apply or a Streams Apply process and not in the context of a user process.

- FIRE\_ONCE=TRUE, APPLY\_SERVER\_ONLY=FALSE

This is the default property setting for a DML trigger. The trigger only fires when user process modifies the base table.

- FIRE\_ONCE=TRUE or FALSE, APPLY\_SERVER\_ONLY=TRUE

The trigger only fires when SQL Apply or Streams Apply process modifies the base table. The trigger does not fire when a user process modifies the base table. Thus the apply-server-only property overrides the fire-once property of a trigger.

---

---

**Note:**

- If you dequeue an error transaction from the error queue and execute it without using the DBMS\_APPLY\_ADM package, then relevant changes resulting from this execution cause a trigger to fire, regardless of the trigger firing property.
  - Only DML and DDL triggers can be fire once. All other types of triggers always fire.
- 
- 

**See Also:** *Oracle Streams Concepts and Administration* for more information about the apply process and controlling a trigger's firing property



## WRAP Functions

This function takes as input a single CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body and returns a CREATE OR REPLACE statement where the text of the PL/SQL unit has been obfuscated.

The function has 3 overloads to allow for the different ways in which DDL statements can be generated dynamically and presented to DBMS\_SQL or EXECUTE IMMEDIATE. The different functionality of each form of syntax is presented with the definition.

**See Also:** [CREATE\\_WRAPPED Procedures](#) on page 53-10

### Syntax

Provides basic functionality:

```
DBMS_DDL.WRAP (
    ddl      VARCHAR2)
RETURN VARCHAR2;
```

Provides the same functionality as the first form, but allows for larger inputs. This function is intended to be used with the [PARSE Procedures](#) in the [DBMS\\_SQL](#) package and its argument list follows the convention of DBMS\_SQL.PARSE:

```
DBMS_DDL.WRAP (
    ddl      DBMS_SQL.VARCHAR2S,
    lb       PLS_INTEGER,
    ub       PLS_INTEGER)
RETURN DBMS_SQL.VARCHAR2S;
```

Provides the same functionality as the second form and is provided for compatibility with multiple forms of the [PARSE Procedures](#) in the [DBMS\\_SQL](#) package:

```
DBMS_DDL.WRAP (
    ddl      DBMS_SQL.VARCHAR2A,
    lb       PLS_INTEGER,
    ub       PLS_INTEGER)
RETURN DBMS_SQL.VARCHAR2A;
```

### Parameters

**Table 53–9 WRAP Function Parameters**

Parameter	Description
ddl	A CREATE OR REPLACE statement that specifies creation of a PL/SQL package specification, package body, function, procedure, type specification or type body
lb	Lower bound for indices in the string table that specify the CREATE OR REPLACE statement
ub	Upper bound for indices in the string table that specify the CREATE OR REPLACE statement.

### Return Values

A CREATE OR REPLACE statement with the text obfuscated. In the case of the second and third form, the return value is a table of strings that need to be concatenated in order to construct the CREATE OR REPLACE string containing obfuscated source text.

## Usage Notes

- Any PL/SQL code that attempts to call these interfaces should use the fully qualified package name `SYS.DBMS_DDL` to avoid the possibility that the name `DBMS_DDL` is captured by a locally-defined unit or by redefining the `DBMS_DDL` public synonym.
- Each invocation of any accepts only a single PL/SQL unit. By contrast, the PL/SQL `wrap` utility accepts a full SQL file and obfuscates the PL/SQL units within the file leaving all other text as-is. These interfaces are intended to be used in conjunction with or as a replacement for PL/SQL's dynamic SQL interfaces (`EXECUTE IMMEDIATE` and `DBMS_SQL.PARSE`). Since these dynamic SQL interfaces only accept a single unit at a time (and do not understand the SQL\*Plus `"/` termination character), both the [CREATE\\_WRAPPED Procedures](#) and the [WRAP Functions](#) require input to be a single unit.

## Exceptions

ORA-24230: If the input is not a `CREATE OR REPLACE` statement specifying a PL/SQL unit, exception `DBMS_DDL.MALFORMED_WRAP_INPUT` is raised.

## Examples

```
DECLARE
    ddl VARCHAR2(32767);
BEGIN
    ddl := GENERATE_PACKAGE(...);
EXECUTE IMMEDIATE SYS.DBMS_DDL.WRAP(ddl); -- Instead of EXECUTE IMMEDIATE ddl
END;
```

---

---

## DBMS\_DEFER

DBMS\_DEFER is the user interface to a replicated transactional deferred remote procedure call facility. Replicated applications use the calls in this interface to queue procedure calls for later transactional execution at remote nodes.

These procedures are typically called from either after row triggers or application specified update procedures.

- [Documentation of DBMS\\_DEFER](#)

---

## Documentation of DBMS\_DEFER

For a complete description of this package within the context of Replication, see DBMS\_DEFER in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_DEFER\_QUERY

DBMS\_DEFER\_QUERY enables you to query the deferred transactions queue data that is not exposed through views.

- [Documentation of DBMS\\_DEFER\\_QUERY](#)

---

## Documentation of DBMS\_DEFER\_QUERY

For a complete description of this package within the context of Replication, see DBMS\_DEFER\_QUERY in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_DEFER\_SYS

DBMS\_DEFER\_SYS subprograms manage default replication node lists. This package is the system administrator interface to a replicated transactional deferred remote procedure call facility. Administrators and replication daemons can execute transactions queued for remote nodes using this facility, and administrators can control the nodes to which remote calls are destined.

- [Documentation of DBMS\\_DEFER\\_SYS](#)

---

## Documentation of DBMS\_DEFER\_SYS

For a complete description of this package within the context of Replication, see DBMS\_DEFER\_SYS in the *Oracle Database Advanced Replication Management API Reference*.



You can use the `DBMS_DESCRIBE` package to get information about a PL/SQL object. When you specify an object name, `DBMS_DESCRIBE` returns a set of indexed tables with the results. Full name translation is performed and security checking is also checked on the final object.

This chapter contains the following topics:

- [Using DBMS\\_DESCRIBE](#)
  - Overview
  - Security Model
  - Types
  - Exceptions
  - Examples
- [Summary of DBMS\\_DESCRIBE Subprograms](#)

## Using DBMS\_DESCRIBE

- [Overview](#)
- [Security Model](#)
- [Types](#)
- [Exceptions](#)
- [Examples](#)

## Overview

This package provides the same functionality as the Oracle Call Interface `OCIDescribeAny` call.

**See Also:** *Oracle Call Interface Programmer's Guide*

## Security Model

This package is available to `PUBLIC` and performs its own security checking based on the schema object being described.

## Types

The DBMS\_DESCRIBE package declares two PL/SQL table types, which are used to hold data returned by DESCRIBE\_PROCEDURE in its OUT parameters. The types are:

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(30)
    INDEX BY BINARY_INTEGER;
```

```
TYPE NUMBER_TABLE IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
```

## Exceptions

DBMS\_DESCRIBE can raise application errors in the range -20000 to -20004.

**Table 57-1 DBMS\_DESCRIBE Errors**

<b>Error</b>	<b>Description</b>
ORA-20000	ORU-10035: cannot describe a package ('X') only a procedure within a package.
ORA-20001	ORU-10032: procedure 'X' within package 'Y' does not exist.
ORA-20002	ORU-10033: object 'X' is remote, cannot describe; expanded name 'Y'.
ORA-20003	ORU-10036: object 'X' is invalid and cannot be described.
ORA-20004	Syntax error attempting to parse 'X'.

## Examples

One use of the DESCRIBE\_PROCEDURE procedure is as an external service interface.

For example, consider a client that provides an OBJECT\_NAME of SCOTT.ACCOUNT\_UPDATE, where ACCOUNT\_UPDATE is an overloaded function with specification:

```
TABLE account (account_no NUMBER, person_id NUMBER,
               balance NUMBER(7,2))
TABLE person  (person_id number(4), person_nm varchar2(10))

FUNCTION ACCOUNT_UPDATE (account_no  NUMBER,
                         person       person%rowtype,
                         amounts      DBMS_DESCRIBE.NUMBER_TABLE,
                         trans_date   DATE)
return account.balance%type;

FUNCTION ACCOUNT_UPDATE (account_no  NUMBER,
                         person       person%rowtype,
                         amounts      DBMS_DESCRIBE.NUMBER_TABLE,
                         trans_no     NUMBER)
return account.balance%type;
```

This procedure might look similar to the following output:

overload	position	argument	level	datatype	length	prec	scale	rad
1	0		0		2	22	7	2 10
1	1	ACCOUNT	0		2	0	0	0 0
1	2	PERSON	0		250	0	0	0 0
1	1	PERSON_ID	1		2	22	4	0 10
1	2	PERSON_NM	1		1	10	0	0 0
1	3	AMOUNTS	0		251	0	0	0 0
1	1		1		2	22	0	0 0
1	4	TRANS_DATE	0		12	0	0	0 0
2	0		0		2	22	7	2 10
2	1	ACCOUNT_NO	0		2	22	0	0 0
2	2	PERSON	0		2	22	4	0 10
2	3	AMOUNTS	0		251	22	4	0 10
2	1		1		2	0	0	0 0
2	4	TRANS_NO	0		2	0	0	0 0

The following PL/SQL procedure has as its parameters all of the PL/SQL datatypes:

```
CREATE OR REPLACE PROCEDURE p1 (
  pvc2   IN   VARCHAR2,
  pvc    OUT  VARCHAR,
  pstr   IN OUT STRING,
  plong  IN   LONG,
  prowid IN   ROWID,
  pchara IN   CHARACTER,
  pchar  IN   CHAR,
  praw  IN   RAW,
  plraw  IN   LONG RAW,
  pbinnt IN   BINARY_INTEGER,
  pplsint IN  PLS_INTEGER,
  pbool  IN   BOOLEAN,
  pnat   IN   NATURAL,
  ppos   IN   POSITIVE,
  pposn  IN   POSITIVEN,
```

```

    pnatn  IN    NATURALN,
    pnum   IN    NUMBER,
    pintgr IN    INTEGER,
    pint   IN    INT,
    psmall IN    SMALLINT,
    pdec   IN    DECIMAL,
    preal  IN    REAL,
    pfloat IN    FLOAT,
    pnumer IN    NUMERIC,
    pdp    IN    DOUBLE PRECISION,
    pdate  IN    DATE,
    pmls   IN    MLSLABEL) AS

```

```

BEGIN
    NULL;
END;

```

If you describe this procedure using the following:

```

CREATE OR REPLACE PACKAGE describe_it AS

    PROCEDURE desc_proc (name VARCHAR2);

END describe_it;

CREATE OR REPLACE PACKAGE BODY describe_it AS

    PROCEDURE prt_value(val VARCHAR2, isize INTEGER) IS
        n INTEGER;
    BEGIN
        n := isize - LENGTHB(val);
        IF n < 0 THEN
            n := 0;
        END IF;
        DBMS_OUTPUT.PUT(val);
        FOR i in 1..n LOOP
            DBMS_OUTPUT.PUT(' ');
        END LOOP;
    END prt_value;

    PROCEDURE desc_proc (name VARCHAR2) IS

        overload    DBMS_DESCRIBE.NUMBER_TABLE;
        position     DBMS_DESCRIBE.NUMBER_TABLE;
        c_level      DBMS_DESCRIBE.NUMBER_TABLE;
        arg_name     DBMS_DESCRIBE.VARCHAR2_TABLE;
        dtv          DBMS_DESCRIBE.NUMBER_TABLE;
        def_val      DBMS_DESCRIBE.NUMBER_TABLE;
        p_mode       DBMS_DESCRIBE.NUMBER_TABLE;
        length       DBMS_DESCRIBE.NUMBER_TABLE;
        precision    DBMS_DESCRIBE.NUMBER_TABLE;
        scale        DBMS_DESCRIBE.NUMBER_TABLE;
        radix        DBMS_DESCRIBE.NUMBER_TABLE;
        spare        DBMS_DESCRIBE.NUMBER_TABLE;
        idx          INTEGER := 0;

    BEGIN
        DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
            name,
            null,
            null,

```



```

        overload,
        position,
        c_level,
        arg_name,
        dty,
        def_val,
        p_mode,
        length,
        precision,
        scale,
        radix,
        spare);

DBMS_OUTPUT.PUT_LINE('Position      Name          DTY  Mode');
LOOP
    idx := idx + 1;
    prt_value(TO_CHAR(position(idx)), 12);
    prt_value(arg_name(idx), 12);
    prt_value(TO_CHAR(dty(idx)), 5);
    prt_value(TO_CHAR(p_mode(idx)), 5);
    DBMS_OUTPUT.NEW_LINE;
END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.NEW_LINE;
        DBMS_OUTPUT.NEW_LINE;

END desc_proc;
END describe_it;

```

Then the results list all the numeric codes for the PL/SQL datatypes:

Position	Name	Datatype_Code	Mode
1	PVC2	1	0
2	PVC	1	1
3	PSTR	1	2
4	PLONG	8	0
5	PROWID	11	0
6	PCHARA	96	0
7	PCHAR	96	0
8	PRAW	23	0
9	PLRAW	24	0
10	PBININT	3	0
11	PPLSINT	3	0
12	PBOOL	252	0
13	PNAT	3	0
14	PPOS	3	0
15	PPOSN	3	0
16	PNATN	3	0
17	PNUM	2	0
18	PINTGR	2	0
19	PINT	2	0
20	PSMALL	2	0
21	PDEC	2	0
22	PREAL	2	0
23	PFLOAT	2	0
24	PNUMER	2	0
25	PDP	2	0
26	PDATE	12	0
27	PMLS	106	0

## Summary of DBMS\_DESCRIBE Subprograms

**Table 57–2 DBMS\_DESCRIBE Package Subprograms**

Subprogram	Description
<a href="#">DESCRIBE_PROCEDURE</a> Procedure on page 57-11	Provides a brief description of a PL/SQL stored procedure

## DESCRIBE\_PROCEDURE Procedure

The procedure `DESCRIBE_PROCEDURE` provides a brief description of a PL/SQL stored procedure. It takes the name of a stored procedure and returns information about each parameter of that procedure.

### Syntax

```
DBMS_DESCRIBE.DESCRIBE_PROCEDURE (
  object_name          IN  VARCHAR2,
  reserved1            IN  VARCHAR2,
  reserved2            IN  VARCHAR2,
  overload             OUT NUMBER_TABLE,
  position             OUT NUMBER_TABLE,
  level                OUT NUMBER_TABLE,
  argument_name        OUT VARCHAR2_TABLE,
  datatype             OUT NUMBER_TABLE,
  default_value        OUT NUMBER_TABLE,
  in_out               OUT NUMBER_TABLE,
  length               OUT NUMBER_TABLE,
  precision            OUT NUMBER_TABLE,
  scale                OUT NUMBER_TABLE,
  radix                OUT NUMBER_TABLE,
  spare                OUT NUMBER_TABLE,
  include_string_constraints OUT BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 57-3 DBMS\_DESCRIBE.DESCRIBE\_PROCEDURE Parameters**

Parameter	Description
<code>object_name</code>	<p>Name of the procedure being described.</p> <p>The syntax for this parameter follows the rules used for identifiers in SQL. The name can be a synonym. This parameter is required and may not be null. The total length of the name cannot exceed 197 bytes. An incorrectly specified <code>OBJECT_NAME</code> can result in one of the following exceptions:</p> <p>ORA-20000 - A package was specified. You can only specify a stored procedure, stored function, packaged procedure, or packaged function.</p> <p>ORA-20001 - The procedure or function that you specified does not exist within the given package.</p> <p>ORA-20002 - The object that you specified is a remote object. This procedure cannot currently describe remote objects.</p> <p>ORA-20003 - The object that you specified is invalid and cannot be described.</p> <p>ORA-20004 - The object was specified with a syntax error.</p>
<code>reserved1</code> <code>reserved2</code>	Reserved for future use -- must be set to <code>NULL</code> or the empty string.
<code>overload</code>	<p>A unique number assigned to the procedure's signature.</p> <p>If a procedure is overloaded, then this field holds a different value for each version of the procedure.</p>
<code>position</code>	<p>Position of the argument in the parameter list.</p> <p>Position 0 returns the values for the return type of a function.</p>

**Table 57-3 (Cont.) DBMS\_DESCRIBE.DESCRIBE\_PROCEDURE Parameters**

Parameter	Description
level	If the argument is a composite type, such as record, then this parameter returns the level of the datatype. See the <i>Oracle Call Interface Programmer's Guide</i> for a description of the ODESSP call for an example.
argument_name	Name of the argument associated with the procedure that you are describing.
datatype	Oracle datatype of the argument being described. The datatypes and their numeric type codes are: 0 placeholder for procedures with no arguments 1 VARCHAR, VARCHAR2, STRING 2 NUMBER, INTEGER, SMALLINT, REAL, FLOAT, DECIMAL 3 BINARY_INTEGER, PLS_INTEGER, POSITIVE, NATURAL 8 LONG 11 ROWID 12 DATE 23 RAW 24 LONG RAW 58 OPAQUE TYPE 96 CHAR (ANSI FIXED CHAR), CHARACTER 106 MLSLABEL 121 OBJECT 122 NESTED TABLE 123 VARRAY 178 TIME 179 TIME WITH TIME ZONE 180 TIMESTAMP 181 TIMESTAMP WITH TIME ZONE 231 TIMESTAMP WITH LOCAL TIME ZONE 250 PL/SQL RECORD 251 PL/SQL TABLE 252 PL/SQL BOOLEAN
default_value	1 if the argument being described has a default value; otherwise, the value is 0.
in_out	Describes the mode of the parameter: 0 IN 1 OUT 2 IN OUT
length	For %rowtype formal arguments, the length constraint is returned, otherwise 0 is returned. If the include_string_constraints parameter is set to TRUE, the argument's formal length constraint is passed back if it is of the appropriate type. Those are the string types: 1;8;23;24;96
precision	If the argument being described is of datatype 2 (NUMBER), then this parameter is the precision of that number.
scale	If the argument being described is of datatype 2 (NUMBER), then this parameter is the scale of that number.
radix	If the argument being described is of datatype 2 (NUMBER), then this parameter is the radix of that number.
spare	Reserved for future functionality.
include_string_constraints	The default is FALSE. If the parameter is set to TRUE, the arguments' formal type constraints is passed back if it is of the appropriate type. Those are the string types: 1;8;23;24;96

**Return Values**

All values from DESCRIBE\_PROCEDURE are returned in its OUT parameters. The datatypes for these are PL/SQL tables, to accommodate a variable number of parameters.



The DBMS\_DG package allows applications to notify the primary database in an Oracle Data Guard broker environment to initiate a fast-start failover when the application encounters a condition that warrants a failover.

**See Also:** *Oracle Data Guard Broker* for more information about performing a fast-start failover in a broker configuration

This chapter contains the following topics:

- [Using DBMS\\_DG](#)
  - [Security Model](#)
- [Summary of the DBMS\\_DG Subprogram](#)

## Using DBMS\_DG

There are conditions detectable by applications running outside of the Oracle database that may warrant the Oracle Data Guard broker to perform a fast-start failover. Because the range of possible conditions is virtually unlimited, it is left to the applications to determine which conditions warrant a fast-start failover.

When such conditions occur, the application calls the `DBMS_DG.INITIATE_FS_FAILOVER` procedure to alert the primary database that the application wants a fast-start failover to occur immediately. The primary database then notifies the observer, which immediately initiates a fast-start failover as long as the standby database is in a valid fast-start failover state ("observed" and either "synchronized" or "within lag") to accept a failover.

If the configuration is not in a valid fast-start failover state, the `INITIATE_FS_FAILOVER` subprogram returns an ORA error message (it will not signal an exception) to inform the calling application that a fast-start failover could not be performed.



## Security Model

The DBMS\_DG package runs with invoker's rights and requires the SYSDBA privilege.

---

## Summary of the DBMS\_DG Subprogram

**Table 58–1 DBMS\_DG Package Subprogram**

Subprogram	Description
<a href="#">INITIATE_FS_FAILOVER Procedure</a> on page 58-5	Enables an application to notify the primary database that a fast-start failover is necessary when the application encounters conditions that warrant a failover.

## INITIATE\_FS\_FAILOVER Procedure

Use this procedure to specify a condition string that, when encountered by an application, allows the application to request the primary database to immediately invoke a fast-start failover.

### Syntax

```
DBMS_DG.INITIATE_FS_FAILOVER (
    condstr          IN VARCHAR2)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 58–2** *INITIATE\_FS\_FAILOVER Procedure Parameters*

Parameter	Description
condstr	Specifies the condition string for which a fast-start failover should be requested. If no condition string argument is supplied, the default string of "Application Failover Requested" will be logged in the broker log file and in the database alert log of the primary database.

### Usage Notes

- This procedure returns a binary integer.
- Query the V\$FS\_FAILOVER\_STATS view to see the time of the last fast-start failover and the reason it was performed.

### Errors

**Table 58–3** *INITIATE\_FS\_FAILOVER Procedure Errors*

Error	Description
ORA-00000: normal, successful completion	The request to initiate a fast-start failover has been posted to the observer.
ORA-16646: fast-start failover is disabled	Either a broker configuration does not exist or fast-start failover has not been enabled.
ORA-16666: unable to initiate fast-start failover on a standby database	DBMS_DG.INITIATE_FS_FAILOVER was invoked on a standby site.
ORA-16817: unsynchronized fast-start failover configuration	DBMS_DG.INITIATE_FS_FAILOVER was invoked in a maximum available fast-start failover configuration when the configuration was not synchronized.
ORA-16819: fast-start failover observer not started	DBMS_DG.INITIATE_FS_FAILOVER was invoked but an observer had not yet been started.
ORA-16820: fast-start failover observer is no longer observing this database	DBMS_DG.INITIATE_FS_FAILOVER was invoked but the configuration detects that the observer may not be running.
ORA-16829: lagging fast-start failover configuration	DBMS_DG.INITIATE_FS_FAILOVER was invoked in a maximum performance fast-start failover configuration when the configuration was not in the user-specified redo lag limit.

**Example**

In this example, the program attempts to initiate a fast-start failover when fast-start failover is disabled. To use this example, connect as user `SYS` with `SYDDBA` privileges.

```
set serveroutput on

declare
status integer;

begin
status := dbms_dg.initiate_fs_failover('Failover Requested');

dbms_output.put_line('Fast-Start Failover is disabled: Expected status =
ORA-16646');
dbms_output.put_line('                Actual Status = ORA-' || status);

end;
/
exit;
```

---

---

## DBMS\_DIMENSION

DBMS\_DIMENSION enables you to verify dimension relationships and provides an alternative to the Enterprise Manager Dimension Wizard for displaying a dimension definition.

**See Also:** *Oracle Database Data Warehousing Guide* for detailed conceptual and usage information about the DBMS\_DIMENSION package

This chapter contains the following topics:

- [Using DBMS\\_DIMENSION](#)
  - Security Model
- [Summary of DBMS\\_DIMENSION Subprograms](#)

---

## Using DBMS\_DIMENSION

This section contains topics which relate to using the DBMS\_DIMENSION package.

- [Security Model](#)

## Security Model

Security on this package can be controlled by granting `EXECUTE` to selected users or roles.

A user can validate or describe all the dimensions in his own schema. To validate or describe a dimension in another schema, you must have either an object privilege on the dimension or one of the following system privileges: `CREATE ANY DIMENSION`, `ALTER ANY DIMENSION`, and `DROP ANY DIMENSION`.

## Summary of DBMS\_DIMENSION Subprograms

**Table 59–1** *DBMS\_DIMENSION Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">DESCRIBE_DIMENSION Procedure</a> on page 59-5	Prints out the definition of the input dimension, including dimension owner and name, levels, hierarchies, and attributes
<a href="#">VALIDATE_DIMENSION Procedure</a> on page 59-6	Verifies that the relationships specified in a dimension are correct



## DESCRIBE\_DIMENSION Procedure

This procedure displays the definition of the dimension, including dimension name, levels, hierarchies, and attributes. It displays the output using the DBMS\_OUTPUT package.

### Syntax

```
DBMS_DIMENSION.DESCRIBE_DIMENSION (  
    dimension IN VARCHAR2);
```

### Parameters

**Table 59–2** DESCRIBE\_DIMENSION Procedure Parameter

Parameter	Description
dimension	The owner and name of the dimension in the format of owner.name.

## VALIDATE\_DIMENSION Procedure

This procedure verifies that the relationships specified in a dimension are valid. The rowid for any row that is found to be invalid will be stored in the table `DIMENSION_EXCEPTIONS` in the user's schema.

### Syntax

```
DBMS_DIMENSION.VALIDATE_DIMENSION (
    dimension          IN VARCHAR2,
    incremental        IN BOOLEAN := TRUE,
    check_nulls        IN BOOLEAN := FALSE,
    statement_id       IN VARCHAR2 := NULL );
```

### Parameters

**Table 59–3** *VALIDATE\_DIMENSION Procedure Parameters*

Parameter	Description
<code>dimension</code>	The owner and name of the dimension in the format of <code>owner.name</code> .
<code>incremental</code>	If <code>TRUE</code> , check only the new rows for tables of this dimension. If <code>FALSE</code> , check all the rows.
<code>check_nulls</code>	If <code>TRUE</code> , then all level columns are verified to be non-null. If <code>FALSE</code> , this check is omitted. Specify <code>FALSE</code> when non-NULLness is guaranteed by other means, such as <code>NOT NULL</code> constraints.
<code>statement_id</code>	A client-supplied unique identifier to associate output rows with specific invocations of the procedure.

The DBMS\_DST package provides an interface to apply the Daylight Saving Time (DST) patch to the Timestamp with Time Zone data type.

**See Also:**

- *Oracle Database Globalization Support Guide*
- *Oracle Database Reference*

This chapter contains the following topics:

- [Using DBMS\\_DST](#)
  - Overview
  - Security Model
  - Views
- [Summary of DBMS\\_DST Subprograms](#)

## Using DBMS\_DST

- [Overview](#)
- [Security Model](#)
- [Views](#)

## Overview

The transition period during which Daylight Saving Time comes into effect, or stops being in effect, has the potential for problems, such as data loss, when handling timestamps with time zone data. The `DBMS_DST` package enables working with these transitions in the context of a set of rules.

## Security Model

The DBMS\_DST package is an invoker's rights package.

**See Also:** *Oracle Database PL/SQL Language Reference* for more information about using Invoker Rights or Definer Rights

The execute privilege on the package is granted to the EXECUTE\_CATALOG\_ROLE role. This role is normally granted to selected users to allow EXECUTE privileges for packages and procedures in the data dictionary.

The user that invokes the package must have the following privileges:

- CREATE ANY TABLE
- ALTER ANY TABLE
- DROP ANY TABLE
- SELECT ANY TABLE
- LOCK ANY TABLE
- ALTER ANY INDEX
- ALTER ANY TRIGGER
- UPDATE ANY TABLE
- EXECUTE ANY TYPE

## Views

The DBMS\_DST package uses the views shown in [Table 60–1, "Views used by DBMS\\_DST"](#), further described in the *Oracle Database Reference*:

**Table 60–1 Views used by DBMS\_DST**

View	Description
DBA_TSTZ_TABLES	Displays information about all tables in the database, which have columns defined on <code>TIMESTAMP WITH TIME ZONE</code> data types or object types containing attributes of <code>TIMESTAMP WITH TIME ZONE</code> data types. Its columns are the same as those in <code>ALL_TSTZ_TABLES</code> .
USER_TSTZ_TABLES	Displays information about the tables owned by the current user, which have columns defined on <code>TIMESTAMP WITH TIME ZONE</code> data types or object types containing attributes of <code>TIMESTAMP WITH TIME ZONE</code> data types. Its columns (except for <code>OWNER</code> ) are the same as those in <code>ALL_TSTZ_TABLES</code> .
ALL_TSTZ_TABLES	Displays information about the tables accessible to the current user, which have columns defined on <code>TIMESTAMP WITH TIME ZONE</code> data types or object types containing attributes of <code>TIMESTAMP WITH TIME ZONE</code> data types.

---

## Summary of DBMS\_DST Subprograms

**Table 60–2 DBMS\_DST Package Subprograms**

Subprogram	Description
<a href="#">BEGIN_PREPARE Procedure</a> on page 60-7	Starts a prepare window
<a href="#">BEGIN_UPGRADE Procedure</a> on page 60-8	Starts an upgrade window
<a href="#">CREATE_AFFECTED_TABLE Procedure</a> on page 60-9	Creates a table that has the schema shown in the comments for the <a href="#">FIND_AFFECTED_TABLES Procedure</a>
<a href="#">CREATE_ERROR_TABLE Procedure</a> on page 60-10	Creates a log error table
<a href="#">CREATE_TRIGGER_TABLE Procedure</a> on page 60-11	Creates a table that is used to record active triggers disabled before performing upgrade on the table, having not been enabled due to fatal failure during the upgrading process
<a href="#">END_PREPARE Procedure</a> on page 60-12	Ends a prepare window
<a href="#">END_UPGRADE Procedure</a> on page 60-13	Ends an upgrade window
<a href="#">FIND_AFFECTED_TABLES Procedure</a> on page 60-14	Finds all the tables that have affected TSTZ data due to the new timezone version
<a href="#">UPGRADE_DATABASE Procedure</a> on page 60-15	Upgrades all tables in the database that have one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type
<a href="#">UPGRADE_SCHEMA Procedure</a> on page 60-17	Upgrades tables in a specified list of schemas that has one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type
<a href="#">UPGRADE_TABLE Procedure</a> on page 60-19	Upgrades a specified list of tables that has one or more columns defined on the TSTZ type or an ADT containing the TSTZ type



## BEGIN\_PREPARE Procedure

This procedure starts a prepare window. Once a prepare window is started successfully, the database property 'DST\_UPGRADE\_STATE' is set to 'PREPARE', and the database property 'SECONDARY\_TT\_VERSION' is set to a new timezone version.

The prepare window lets a DBA investigate data affected by the upgrade, and so judge when it is optimal to perform the upgrade. The prepare window can overlap normal database operation.

### Syntax

```
DBMS_DST.BEGIN_PREPARE (  
    new_version                IN BINARY_INTEGER);
```

### Parameters

**Table 60–3** *BEGIN\_PREPARE Procedure Parameters*

Parameter	Description
new_version	New timezone version to which the database is to be prepared to upgrade

## BEGIN\_UPGRADE Procedure

This procedure starts an upgrade window. When an upgraded window is started successfully, the TSTZ data in the dictionary tables is upgraded to reflect the new timezone version, and the database property 'DST\_UPGRADE\_STATE' is set to 'UPGRADE'. Once BEGIN\_UPGRADE has been performed successfully, the user must re-start the database. After a successful restart, the database property 'PRIMARY\_TT\_VERSION' is the new timezone version, and 'SECONDARY\_TT\_VERSION' is the old timezone version.

The procedure operates atomically, and upgrades all or none of the dictionary tables and the database properties. It must be called in the database in OPEN MIGRATE mode.

### Syntax

```
DBMS_DST.BEGIN_UPGRADE (
    new_version          IN BINARY_INTEGER,
    error_on_overlap_time IN BOOLEAN := FALSE,
    error_on_nonexisting_time IN BOOLEAN := FALSE);
```

### Parameters

**Table 60–4 BEGIN\_UPGRADE Procedure Parameters**

Parameter	Description
new_version	New timezone version to which the database is to be upgraded
error_on_overlap_time	Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error. The default is TRUE. For more information about boundary cases, see <i>Oracle Database SQL Language Reference</i> .
error_on_nonexisting_time	Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error. The default is TRUE.

## CREATE\_AFFECTED\_TABLE Procedure

This procedure creates a table that has the schema shown in the comments for the [FIND\\_AFFECTED\\_TABLES Procedure](#).

### Syntax

```
DBMS_DST.CREATE_AFFECTED_TABLE (  
    table_name      IN VARCHAR2);
```

### Parameters

**Table 60–5** *CREATE\_AFFECTED\_TABLE Procedure Parameters*

Parameter	Description
table_name	Name of the table created

### Usage Notes

This procedure takes a `table_name` without schema qualification, creating a table within the current user schema.

## CREATE\_ERROR\_TABLE Procedure

This procedure creates a log error table which has the following schema:

```
CREATE TABLE dst$error_table(  
  table_owner    VARCHAR2(30),  
  table_name     VARCHAR2(30),  
  column_name    VARCHAR2(4000),  
  rid            ROWID,  
  error_number   NUMBER)
```

### Syntax

```
DBMS_DST.CREATE_ERROR_TABLE (  
  table_name     IN VARCHAR2);
```

### Parameters

**Table 60–6** CREATE\_ERROR\_TABLE Procedure Parameters

Parameter	Description
table_name	Name of the table created

### Usage Notes

- This procedure takes a table\_name without schema qualification, creating a table within the current user schema.
- The error number is found when upgrading time zone file and timestamp with time zone data. For more information about error handling when upgrading time zone file and timestamp with time zone data, see Oracle Database Globalization Support Guide

## CREATE\_TRIGGER\_TABLE Procedure

This procedure creates a table that has the following schema.

```
CREATE TABLE dst_trigger_table (  
  trigger_owner VARCHAR2(30),  
  trigger_name  VARCHAR2(30));
```

This table is used to record active triggers that are disabled before performing upgrade on the table, having not been enabled due to fatal failure during the upgrading process.

### Syntax

```
DBMS_DST.CREATE_TRIGGER_TABLE (  
  table_name      IN  VARCHAR2);
```

### Parameters

**Table 60–7** CREATE\_TRIGGER\_TABLE Procedure Parameters

Parameter	Description
table_name	Name of table to be created

### Usage Notes

This procedure takes a table\_name without schema qualification, creating a table within the current user schema.

## END\_PREPARE Procedure

This procedure ends a prepare window.

### Syntax

```
DBMS_DST.BEGIN_PREPARE;
```

## END\_UPGRADE Procedure

This procedure ends an upgrade window. An upgraded window is ended if all the affected user tables have been upgraded. Otherwise, the OUT parameter `num_of_failures` indicates how many tables have not been converted.

### Syntax

```
DBMS_DST.END_UPGRADE (  
    num_of_failures    OUT  BINARY_INTEGER);
```

### Parameters

**Table 60–8** *END\_UPGRADE Procedure Parameters*

Parameter	Description
<code>num_of_failures</code>	Number of tables that fail to complete

## FIND\_AFFECTED\_TABLES Procedure

This procedure finds all the tables which have affected TSTZ data due to the new timezone version. This procedure can only be invoked during a prepare window. The tables which have affected TSTZ data are recorded into a table indicated by parameter `affected_tables`. If semantic errors must be logged, they are recorded into a table indicated by parameter `log_errors_table`.

### Syntax

```
DBMS_DST.FIND_AFFECTED_TABLES (
  affected_tables      IN  VARCHAR2 := 'sys.dst$affected_tables',
  log_errors           IN  BOOLEAN := FALSE,
  log_errors_table     IN  VARCHAR2 := 'sys.dst$error_table');
```

### Parameters

**Table 60–9 FIND\_AFFECTED\_TABLES Procedure Parameters**

Parameter	Description
<code>affected_tables</code>	<p>Name of table with the following schema:</p> <pre>CREATE TABLE dst\$affected_tables (   table_owner VARCHAR2(30),   table_name VARCHAR2(30),   column_name VARCHAR2(4000),   row_count NUMBER,   error_count NUMBER)</pre> <p>The table can be created with the <a href="#">CREATE_AFFECTED_TABLE Procedure</a>.</p>
<code>log_errors</code>	<p>Boolean flag indicating whether to log errors during upgrade. If <code>FALSE</code>, no error is logged into the <code>log_errors_table</code> after aborting conversion of the current table. If <code>TRUE</code>, the error is logged to the <code>log_errors_table</code>.</p> <p>The default is <code>FALSE</code>.</p>
<code>log_errors_table</code>	<p>Table name with the following schema:</p> <pre>CREATE TABLE dst\$error_table (   table_owner VARCHAR2(30),   table_name VARCHAR2(30),   column_name VARCHAR2(4000),   rid ROWID,   error_number NUMBER)</pre> <p>The table can be created with the <a href="#">CREATE_ERROR_TABLE Procedure</a>. The <code>rid</code> column records the rowids of the offending rows, and the <code>error_number</code> column records the corresponding error number.</p>



## UPGRADE\_DATABASE Procedure

This procedure upgrades all tables in the database, which have one or more columns defined on the TSTZ type or an ADT containing the TSTZ type. This procedure can only be invoked after an upgrade window has been started. Each table is upgraded in an atomic transaction. Note that, a base table and its materialized view log table are upgraded in an atomic transaction.

### Syntax

```
DBMS_DST.UPGRADE_DATABASE (
    num_of_failures      OUT BINARY_INTEGER,
    upgrade_data         IN  BOOLEAN := TRUE,
    parallel             IN  BOOLEAN := FALSE,
    continue_after_errors IN  BOOLEAN := TRUE,
    log_errors           IN  BOOLEAN := FALSE,
    log_errors_table     IN  VARCHAR2 := 'sys.dst$error_table' ,
    error_on_overlap_time IN  BOOLEAN := FALSE,
    error_on_nonexisting_time IN  BOOLEAN := FALSE,
    log_triggers_table  IN  VARCHAR2 := 'sys.dst$trigger_table');
```

### Parameters

**Table 60–10 UPGRADE\_DATABASE Procedure Parameters**

Parameter	Description
num_of_failures	Number of tables that fail to complete
upgrade_data	Boolean flag indicating whether to convert TSTZ data using the new Time Zone patch File (TRUE), or to leave it unconverted (FALSE).  The default is TRUE.
parallel	Boolean flag indicating whether to convert tables using PDML (Parallel DML) or Serial DML.  The default is FALSE.
continue_after_errors	Boolean flag indicating whether to continue after upgrade fails on the current table.  The default is TRUE.
log_errors	Boolean flag indicating whether to log errors during upgrade. If FALSE, no error is logged into the log_errors_table after aborting conversion of the current table. If TRUE, errors are logged to the log_errors_table.  The default is FALSE.
log_errors_table	Table name with the following schema:  <pre>CREATE TABLE dst\$error_table (   table_owner VARCHAR2(30),   table_name VARCHAR2(30),   column_name VARCHAR2(4000),   rid ROWID,   error_number NUMBER)</pre> The table can be created with the <a href="#">CREATE_ERROR_TABLE Procedure</a> . The rid column records the rowids of the offending rows, and the error_number column records the corresponding error number.

**Table 60–10 (Cont.) UPGRADE\_DATABASE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
error_on_overlap_time	Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error. The default is TRUE.
error_on_nonexisting_time	Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error. The default is TRUE.
log_triggers_table	Table to log triggers which are disabled before upgrade, having not been enabled due to a fatal failure when performing an upgrade

## UPGRADE\_SCHEMA Procedure

This procedure upgrades tables in a specified list of schemas that have one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type. This procedure can be invoked only after an upgrade window has been started. Each table is upgraded in an atomic transaction. Note that a base table and its materialized view log table are upgraded in an atomic transaction.

### Syntax

```
DBMS_DST.UPGRADE_SCHEMA (
    num_of_failures          OUT BINARY_INTEGER,
    schema_list              IN VARCHAR2,
    upgrade_data             IN BOOLEAN := TRUE,
    parallel                 IN BOOLEAN := FALSE,
    continue_after_errors   IN BOOLEAN := TRUE,
    log_errors               IN BOOLEAN := FALSE,
    log_errors_table        IN VARCHAR2 := 'sys.dst$error_table' ,
    error_on_overlap_time    IN BOOLEAN := FALSE,
    error_on_nonexisting_time IN BOOLEAN := FALSE,
    log_triggers_table      IN VARCHAR2 := 'sys.dst$trigger_table');
```

### Parameters

**Table 60–11 UPGRADE\_SCHEMA Procedure Parameters**

Parameter	Description
num_of_failures	Number of tables that fail to complete
schema_list	Schema name list (comma separated strings)
upgrade_data	Boolean flag indicating whether to convert TSTZ data using the new Time Zone patch File (TRUE) or to leave unconverted (FALSE). The default is TRUE.
parallel	Boolean flag indicating whether to convert tables using PDML (Parallel DML) or Serial DML. The default is FALSE.
continue_after_errors	Boolean flag indicating whether to continue after upgrade fails on the current table. The default is TRUE.
log_errors	Boolean flag indicating whether to log errors during upgrade. If FALSE, no error is logged into the log_errors_table after aborting conversion of the current table. If TRUE, the error is logged to the log_errors_table. The default is FALSE.

**Table 60–11 (Cont.) UPGRADE\_SCHEMA Procedure Parameters**

Parameter	Description
log_errors_table	<p>Table name with the following schema:</p> <pre>CREATE TABLE dst\$error_table (   table_owner VARCHAR2(30),   table_name VARCHAR2(30),   column_name VARCHAR2(4000),   rid ROWID,   error_number NUMBER)</pre> <p>The table can be created with the <a href="#">CREATE_ERROR_TABLE Procedure</a>. The <code>rid</code> column records the rowids of the offending rows, and the <code>error_number</code> column records the corresponding error number.</p>
error_on_overlap_time	<p>Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error.</p> <p>The default is <code>TRUE</code>.</p>
error_on_nonexisting_time	<p>Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error.</p> <p>The default is <code>TRUE</code>.</p>
log_triggers_table	<p>Table to log triggers that are disabled before upgrade, having not been enabled due to a fatal failure when performing an upgrade</p>

## UPGRADE\_TABLE Procedure

This procedure upgrades a specified list of tables that have one or more columns defined on the TSTZ type, or an ADT containing the TSTZ type.

### Syntax

```
DBMS_DST.UPGRADE_TABLE (
    num_of_failures      OUT BINARY_INTEGER,
    table_list           IN VARCHAR2,
    upgrade_data         IN BOOLEAN := TRUE,
    parallel             IN BOOLEAN := FALSE,
    continue_after_errors IN BOOLEAN := TRUE,
    log_errors           IN BOOLEAN := FALSE,
    log_errors_table     IN VARCHAR2 := 'sys.dst$error_table' ,
    error_on_overlap_time IN BOOLEAN := FALSE,
    error_on_nonexisting_time IN BOOLEAN := FALSE,
    log_triggers_table  IN VARCHAR2 := 'sys.dst$trigger_table',
    atomic_upgrade      IN BOOLEAN := FALSE);
```

### Parameters

**Table 60–12 UPGRADE\_TABLE Procedure Parameters**

Parameter	Description
num_of_failures	Number of tables that fail to complete
table_list	Table name list (comma separated strings)
upgrade_data	Boolean flag indicating whether to convert TSTZ data using the new Time Zone patch File (TRUE), or to leave unconverted (FALSE). The default is TRUE.
parallel	Boolean flag indicating whether to convert tables using PDML (Parallel DML), or Serial DML. The default is FALSE.
continue_after_errors	Boolean flag indicating whether to continue after upgrade fails on the current table. The default is TRUE.
log_errors	Boolean flag indicating whether to log errors during upgrade. If FALSE, no error is logged into the log_errors_table after aborting conversion of the current table. If TRUE, the error is logged to the log_errors_table. The default is FALSE.
log_errors_table	Table name with the following schema: <pre>CREATE TABLE dst\$error_table (   table_owner VARCHAR2(30),   table_name VARCHAR2(30),   column_name VARCHAR2(4000),   rid ROWID,   error_number NUMBER)</pre> The table can be created with the <a href="#">CREATE_ERROR_TABLE Procedure</a> . The rid parameter records the rowids of the offending rows and the corresponding error number.

**Table 60–12 (Cont.) UPGRADE\_TABLE Procedure Parameters**

Parameter	Description
<code>error_on_overlap_time</code>	Boolean flag indicating whether to report errors on the 'overlap' time semantic conversion error. The default is <code>TRUE</code> .
<code>error_on_nonexisting_time</code>	Boolean flag indicating whether to report errors on the 'non-existing' time semantic conversion error. The default is <code>TRUE</code> .
<code>log_triggers_table</code>	Table to log triggers that are disabled before upgrade, having not been enabled due to a fatal failure when performing an upgrade
<code>atomic_upgrade</code>	Boolean flag indicating whether to convert the listed tables atomically (in a single transaction). If <code>FALSE</code> , each table is converted in its own transaction. The default is <code>FALSE</code> .

### Usage Notes

This procedure can only be invoked after an upgrade window has been started. The table list has to satisfy the following partial ordering:

1. If a base table has a materialized view log table, the log table must be the next item in the list.
2. If the container table for a materialized view appears in the list, the materialized view's 'non-upgraded' base tables and log tables must appear in the table list and before the container table.

A base table and its materialized view log table need to be upgraded in an atomic transaction by specifying `atomic_upgrade` to `TRUE`.

---

---

## DBMS\_DISTRIBUTED\_TRUST\_ADMIN

DBMS\_DISTRIBUTED\_TRUST\_ADMIN procedures maintain the Trusted Servers List. Use these procedures to define whether a server is trusted. If a database is not trusted, Oracle refuses current user database links from the database.

This chapter contains the following topics:

- [Using DBMS\\_DISTRIBUTED\\_TRUST\\_ADMIN](#)
  - Overview
  - Security Model
  - Examples
- [Summary of DBMS\\_DISTRIBUTED\\_TRUST\\_ADMIN Subprograms](#)

---

## Using DBMS\_DISTRIBUTED\_TRUST\_ADMIN

- [Overview](#)
- [Security Model](#)
- [Examples](#)



## Overview

Oracle uses local Trusted Servers Lists, along with enterprise domain membership lists stored in the enterprise LDAP directory service, to determine if another database is trusted. The LDAP directory service entries are managed with the Enterprise Security Manager Tool in Oracle Enterprise Manager.

Oracle considers another database to be "trusted" if it meets the following criteria:

1. It is in the same enterprise domain in the directory service as the local database.
2. The enterprise domain is marked as trusted in the directory service.
3. It is not listed as untrusted in the local Trusted Servers List. Current user database links will only be accepted from another database if both databases involved trust each other.

You can list a database server locally in the Trusted Servers List regardless of what is listed in the directory service. However, if you list a database that is not in the same domain as the local database, or if that domain is untrusted, the entry will have no effect.

This functionality is part of the Enterprise User Security feature of the Oracle Advanced Security Option.

## Security Model

To execute `DBMS_DISTRIBUTED_TRUST_ADMIN`, the `EXECUTE_CATALOG_ROLE` role must be granted to the DBA. To select from the view `TRUSTED_SERVERS`, the `SELECT_CATALOG_ROLE` role must be granted to the DBA.

It is important to know whether all servers are trusted or not trusted. Trusting a particular server with the `ALLOW_SERVER` procedure does not have any effect if the database already trusts all databases, or if that database is already trusted. Similarly, denying a particular server with the `DENY_SERVER` procedure does not have any effect if the database already does not trust any database or if that database is already untrusted.

The procedures `DENY_ALL` and `ALLOW_ALL` delete all entries (in other words, server names) that are explicitly allowed or denied using the `ALLOW_SERVER` procedure or `DENY_SERVER` procedure respectively.

## Examples

If you have not yet used the package `DBMS_DISTRIBUTED_TRUST_ADMIN` to change the trust listing, by default you trust all databases in the same enterprise domain if that domain is listed as trusted in the directory service:

```
SELECT * FROM TRUSTED_SERVERS;
TRUST      NAME
-----
Trusted    All
```

Because all servers are currently trusted, you can execute the [DENY\\_SERVER Procedure](#) and specify that a particular server is not trusted:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER
        ('SALES.US.AMERICAS.ACME_AUTO.COM');
PL/SQL procedure successfully completed.
```

```
SELECT * FROM TRUSTED_SERVERS;
TRUST      NAME
-----
Untrusted  SALES.US.AMERICAS.ACME_AUTO.COM
```

By executing the [DENY\\_ALL Procedure](#), you can choose to not trust any database server:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

```
PL/SQL procedure successfully completed.
```

```
SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
-----
Untrusted  All
```

The [ALLOW\\_SERVER Procedure](#) can be used to specify that one particular database is to be trusted:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER
        ('SALES.US.AMERICAS.ACME_AUTO.COM');
PL/SQL procedure successfully completed.
```

```
SELECT * FROM TRUSTED_SERVERS;
TRUST      NAME
-----
Trusted    SALES.US.AMERICAS.ACME_AUTO.COM
```

---

## Summary of DBMS\_DISTRIBUTED\_TRUST\_ADMIN Subprograms

**Table 61–1** *DBMS\_DISTRIBUTED\_TRUST\_ADMIN Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">ALLOW_ALL Procedure</a> on page 61-7	Empties the list and inserts a row indicating that all servers should be trusted
<a href="#">ALLOW_SERVER Procedure</a> on page 61-8	Enables a specific server to be allowed access even though deny all is indicated in the list
<a href="#">DENY_ALL Procedure</a> on page 61-9	Empties the list and inserts a row indicating that all servers should be untrusted
<a href="#">DENY_SERVER Procedure</a> on page 61-10	Enables a specific server to be denied access even though allow all is indicated in the list

## ALLOW\_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers that are members of a trusted domain in an enterprise directory service and that are in the same domain are allowed access.

The view `TRUSTED_SERVERS` will show "TRUSTED ALL" indicating that the database trusts all servers that are currently trusted by the enterprise directory service.

### Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL;
```

### Usage Notes

`ALLOW_ALL` only applies to servers listed as trusted in the enterprise directory service and in the same enterprise domain.

## ALLOW\_SERVER Procedure

This procedure ensures that the specified server is considered trusted (even if you have previously specified "deny all").

### Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (  
    server IN VARCHAR2);
```

### Parameters

**Table 61–2** ALLOW\_SERVER Procedure Parameters

Parameter	Description
server	Unique, fully-qualified name of the server to be trusted.

### Usage Notes

If the Trusted Servers List contains the entry "deny all", then this procedure adds a specification indicating that a specific database (for example, DBx) is to be trusted.

If the Trusted Servers List contains the entry "allow all", and if there is no "deny DBx" entry in the list, then executing this procedure causes no change.

If the Trusted Servers List contains the entry "allow all", and if there is a "deny DBx" entry in the list, then that entry is deleted.

## DENY\_ALL Procedure

This procedure empties the Trusted Servers List and specifies that all servers are denied access.

The view `TRUSTED_SERVERS` will show "UNTRUSTED ALL" indicating that no servers are currently trusted.

### Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

## DENY\_SERVER Procedure

This procedure ensures that the specified server is considered untrusted (even if you have previously specified `allow all`).

### Syntax

```
DBMS_DISTRICTED_TRUST_ADMIN.DENY_SERVER (  
    server IN VARCHAR2);
```

### Parameters

**Table 61-3** *DENY\_SERVER Procedure Parameters*

Parameter	Description
<code>server</code>	Unique, fully-qualified name of the server to be untrusted.

### Usage Notes

If the Trusted Servers List contains the entry `allow all`, then this procedure adds an entry indicating that the specified database (for example, `DBx`) is not to be trusted.

If the Trusted Servers List contains the entry `"deny all"`, and if there is no `"allow DBx"` entry in the list, then this procedure causes no change.

If the Trusted Servers List contains the entry `"deny all"`, and if there is an `"allow DBx"` entry, then this procedure causes that entry to be deleted.



---

---

## DBMS\_EDITIONS\_UTILITIES

The DBMS\_EDITIONS\_UTILITIES package implements the Edition API which provides helper functions for edition-related operations.

The chapter contains the following topics:

- [Using DBMS\\_EDITIONS\\_UTILITIES](#)
  - Overview
  - Security Model
  - Exceptions
- [Summary of DBMS\\_EDITIONS\\_UTILITIES Subprograms](#)

---

## Using DBMS\_EDITIONS\_UTILITIES

- [Overview](#)
- [Security Model](#)
- [Exceptions](#)

## Overview

The `DBMS_EDITIONS_UTILITIES` package implements the Edition API which provides helper functions for edition-related operations.

## Security Model

This package is owned by `SYS` with execute access granted to `PUBLIC`. It runs with invokers rights, that is, with the security profile of the caller.

## Exceptions

**Table 62–1** *DBMS\_EDITIONS\_UTILITIES Error Messages*

<b>Error</b>	<b>Description</b>
ORA-38817	Insufficient privileges
ORA-942	Missing table

## Summary of DBMS\_EDITIONS\_UTILITIES Subprograms

**Table 62–2 DBMS\_EDITIONS\_UTILITIES Package Subprograms**

Subprogram	Description
<a href="#">SET_EDITIONING_VIEWS_READ_ONLY Procedure</a> on page 62-7	Given the schema name and table name, this procedure sets the corresponding editioning views in all editions to READ ONLY or READ/WRITE

## SET\_EDITIONING\_VIEWS\_READ\_ONLY Procedure

Given the schema name and table name, this procedure sets the corresponding editioning views in all editions to READ ONLY or READ/WRITE.

### Syntax

```
DBMS_EDITIONS_UTILITIES.SET_EDITIONING_VIEWS_READ_ONLY (
  table_name IN VARCHAR2,
  owner      IN VARCHAR2 DEFAULT NULL,
  read_only  IN BOOLEAN  DEFAULT TRUE);
```

### Parameters

**Table 62-3 SET\_EDITIONING\_VIEWS\_READ\_ONLY Procedure Parameters**

Parameter	Description
table_name	Base table of the editioning views
owner	Base table schema. The default (or NULL) is the current schema.
read_only	TRUE to set the views to read-only; FALSE (or NULL) sets the views to READ/WRITE. Default is TRUE.

### Usage Notes

The user must have the following privileges:

- Owner of the table, or have the ALTER ANY TABLE system privileges
- USE object privilege on all the editions for which the views are defined





The DBMS\_EPG package implements the embedded PL/SQL gateway that enables a Web browser to invoke a PL/SQL stored procedure through an HTTP listener.

This chapter contains the following topics:

- [Using DBMS\\_EPG](#)
  - Overview
  - Security Model
  - Exceptions
- [Data Structures](#)
  - [VARCHAR2\\_TABLE](#) Table Type
- [Subprogram Groups](#)
  - Configuration Subprograms
  - Authorization Subprograms
- [Summary of DBMS\\_EPG Subprograms](#)

## Using DBMS\_EPG

- [Overview](#)
- [Security Model](#)
- [Exceptions](#)

## Overview

The DBMS\_EPG package is a platform on which PL/SQL users develop and deploy PL/SQL Web applications. The embedded PL/SQL gateway is an embedded version of the gateway that runs in the XML database HTTP server in the Oracle database. It provides the core features of `mod_plsql` in the database but does not require the Oracle HTTP server powered by Apache.

In order to make a PL/SQL application accessible from a browser by way of HTTP, a Database Access Descriptor (DAD) must be created and mapped to a virtual path. A DAD is a set of configuration values used for database access and the virtual path mapping makes the application accessible under a virtual path of the XML DB HTTP Server. A DAD is represented as a servlet in XML DB HTTP Server.

## Security Model

The XDBADMIN role is required to invoke the configuration interface. It may be invoked by the database user "XDB".

The authorization interface can be invoked by any user.

## Exceptions

The following table lists the exceptions raised by the DBMS\_EPG package.

**Table 63–1** *DBMS\_EPG Exceptions*

<b>Exception</b>	<b>Error Code</b>	<b>Description</b>
DAD_NOT_FOUND	20000	Database Access Descriptor (DAD) %s not found. Ensure that the name of the DAD is correct and that it exists.

## Data Structures

The DBMS\_EPG package defines a TABLE type.

### **VARCHAR2\_TABLE Table Type**

This type is used by the procedures GET\_ALL\_GLOBAL\_ATTRIBUTES, GET\_ALL\_DAD\_ATTRIBUTES, GET\_ALL\_DAD\_MAPPINGS, and GET\_DAD\_LIST to return lists of attribute names, attribute values, virtual paths, and database access descriptors (DAD).

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

## Subprogram Groups

---

The DBMS\_EPG consists of two interfaces:

- [Configuration Subprograms](#)
- [Authorization Subprograms](#)

## Configuration Subprograms

The Configuration subprogram group contain the subprogram interfaces to examine and modify the global and database access descriptor (DAD) specific settings of the embedded PL/SQL gateway.

**Table 63–2 Configuration Subprogram Group**

Subprogram	Description
<a href="#">CREATE_DAD Procedure</a> on page 63-12	Creates a new DAD
<a href="#">DELETE_DAD_ATTRIBUTE Procedure</a> on page 63-14	Deletes a DAD attribute
<a href="#">DELETE_GLOBAL_ATTRIBUTE Procedure</a> on page 63-15	Deletes a global attribute
<a href="#">DROP_DAD Procedure</a> on page 63-16	Drops a DAD
<a href="#">GET_ALL_DAD_ATTRIBUTES Procedure</a> on page 63-17	Retrieves all the attributes of a DAD.
<a href="#">GET_ALL_DAD_MAPPINGS Procedure</a> on page 63-18	Retrieves all virtual paths to which the specified DAD is mapped.
<a href="#">GET_ALL_GLOBAL_ATTRIBUTES Procedure</a> on page 63-19	Retrieves all global attributes and values
<a href="#">GET_DAD_ATTRIBUTE Function</a> on page 63-20	Retrieves the value of a DAD attribute
<a href="#">GET_DAD_LIST Procedure</a> on page 63-21	Retrieves a list of all DADs for an Embedded Gateway instance.
<a href="#">GET_GLOBAL_ATTRIBUTE Function</a> on page 63-22	Retrieves the value of a global attribute
<a href="#">MAP_DAD Procedure</a> on page 63-23	Maps a DAD to the specified virtual path.
<a href="#">SET_DAD_ATTRIBUTE Procedure</a> on page 63-24	Sets the value for a DAD
<a href="#">SET_GLOBAL_ATTRIBUTE Procedure</a> on page 63-26	Sets the value of a global attribute
<a href="#">UNMAP_DAD Procedure</a> on page 63-27	Unmaps a DAD from the specified virtual path



## Authorization Subprograms

The Authorization subprogram group contains the subprogram interfaces to authorize and deauthorize the use of a database user's privileges by the embedded PL/SQL gateway through a specific database access descriptor (DAD)

**Table 63-3 Authorization Subprogram Group**

<b>Subprogram</b>	<b>Description</b>
<a href="#">AUTHORIZE_DAD Procedure</a> on page 63-11	Authorizes a DAD to invoke procedures and access document tables with a database user's privileges
<a href="#">DEAUTHORIZE_DAD Procedure</a> on page 63-13	Deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges

---

## Summary of DBMS\_EPG Subprograms

**Table 63–4 DBMS\_EPG Package Subprograms**

Subprogram	Description
<a href="#">AUTHORIZE_DAD Procedure</a> on page 63-11	authorizes a DAD to invoke procedures and access document tables with a database user's privileges
<a href="#">CREATE_DAD Procedure</a> on page 63-12	Creates a new DAD
<a href="#">DEAUTHORIZE_DAD Procedure</a> on page 63-13	Deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges
<a href="#">DELETE_DAD_ATTRIBUTE Procedure</a> on page 63-14	Deletes a DAD attribute
<a href="#">DELETE_GLOBAL_ATTRIBUTE Procedure</a> on page 63-15	Deletes a global attribute
<a href="#">DROP_DAD Procedure</a> on page 63-16	Drops a DAD
<a href="#">GET_ALL_DAD_ATTRIBUTES Procedure</a> on page 63-17	Retrieves all the attributes of a DAD.
<a href="#">GET_ALL_DAD_MAPPINGS Procedure</a> on page 63-18	Retrieves all virtual paths to which the specified DAD is mapped.
<a href="#">GET_ALL_GLOBAL_ATTRIBUTES Procedure</a> on page 63-19	Retrieves all global attributes and values
<a href="#">GET_DAD_ATTRIBUTE Function</a> on page 63-20	Retrieves the value of a DAD attribute
<a href="#">GET_DAD_LIST Procedure</a> on page 63-21	Retrieves a list of all DADs for an Embedded Gateway instance.
<a href="#">GET_GLOBAL_ATTRIBUTE Function</a> on page 63-22	Retrieves the value of a global attribute
<a href="#">MAP_DAD Procedure</a> on page 63-23	Maps a DAD to the specified virtual path.
<a href="#">SET_DAD_ATTRIBUTE Procedure</a> on page 63-24	Sets the value for a DAD
<a href="#">SET_GLOBAL_ATTRIBUTE Procedure</a> on page 63-26	Sets the value of a global attribute
<a href="#">UNMAP_DAD Procedure</a> on page 63-27	Unmaps a DAD from the specified virtual path

## AUTHORIZE\_DAD Procedure

This procedure authorizes a DAD to invoke procedures and access document tables with a database user's privileges. The invoker can always authorize the use of her/his own privileges.

**See Also:** [Authorization Subprograms](#) on page 63-9 for other subprograms in this group

### Syntax

```
DBMS_EPG.AUTHORIZE_DAD (
  dad_name IN VARCHAR2,
  path     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 63-5** *AUTHORIZE\_DAD Procedure Parameters*

Parameter	Description
dad_name	The name of the DAD to create
user	The user whose privileges to deauthorize. If use, the invoker is assumed.

### Usage Notes

- To authorize the use of another user's privileges, the invoker must have the ALTER USER system privilege.
- The DAD must exist but its "database-username" DAD attribute does not have to be set to user to authorize.
- Multiple users can authorize the same DAD and it is up to the DAD's "database-username" setting to decide which user's privileges to use.

### Exceptions

Raises an error if the DAD or user does not exist, or the invoker does not have the needed system privilege.

### Examples

```
DBMS_EPG.AUTHORIZE_DAD('HR');
```

## CREATE\_DAD Procedure

This procedure creates a new DAD.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.CREATE_DAD (  
    dad_name IN VARCHAR2,  
    path     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 63–6** CREATE\_DAD Procedure Parameters

Parameter	Description
dad_name	The name of the DAD to create
path	The virtual path to which to map the DAD

## DEAUTHORIZE\_DAD Procedure

This procedure deauthorizes a DAD with regard to invoking procedures and accessing document tables with a database user's privileges. The invoker can always deauthorize the use of his own privileges.

**See Also:** [Authorization Subprograms](#) on page 63-9 for other subprograms in this group

### Syntax

```
DBMS_EPG.DEAUTHORIZE_DAD (
  dad_name IN VARCHAR2,
  path     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 63–7 DEAUTHORIZE\_DAD Procedure Parameters**

Parameter	Description
dad_name	The name of the DAD for which to deauthorize use
user	The user whose privileges to deauthorize. If use, the invoker is assumed.

### Usage Notes

To deauthorize the use of another user's privileges, the invoker must have the ALTER USER system privilege.

### Exceptions

Raises an error if the DAD or user does not exist, or the invoker does not have the needed system privilege.

### Examples

```
DBMS_EPG.DEAUTHORIZE_DAD('HR');
```

## DELETE\_DAD\_ATTRIBUTE Procedure

This procedure deletes a DAD attribute.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.DELETE_DAD_ATTRIBUTE (  
    dad_name      IN  VARCHAR2,  
    attr_name     IN  VARCHAR2);
```

### Parameters

**Table 63–8** *DELETE\_DAD\_ATTRIBUTE Procedure Parameters*

Parameter	Description
dad_name	The name of the DAD for which to delete a DAD attribute
attr_name	The name of the DAD attribute to delete

### Exceptions

Raises an error if DAD does not exist

## DELETE\_GLOBAL\_ATTRIBUTE Procedure

This procedure deletes a global attribute.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.DELETE_GLOBAL_ATTRIBUTE (  
    attr_name      IN VARCHAR2);
```

### Parameters

**Table 63–9** *DELETE\_GLOBAL\_ATTRIBUTE Procedure Parameters*

Parameter	Description
attr_name	The global attribute to delete

## DROP\_DAD Procedure

This procedure drops a DAD. All the virtual-path mappings of the DAD will be dropped also

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.DROP_DAD (  
    dadname IN VARCHAR2);
```

### Parameters

**Table 63–10** *DROP\_DAD Procedure Parameters*

Parameter	Description
dad_name	The DAD to drop

### Exceptions

Raises an error if the DAD does not exist.



## GET\_ALL\_DAD\_ATTRIBUTES Procedure

This procedure retrieves all the attributes of a DAD. The outputs are 2 correlated index-by tables of the name/value pairs.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.GET_ALL_DAD_ATTRIBUTES (
  dad_name      IN          VARCHAR2,
  attr_names    OUT NOCOPY VARCHAR2_TABLE,
  attr_values   OUT NOCOPY VARCHAR2_TABLE);
```

### Parameters

**Table 63–11** GET\_ALL\_DAD\_ATTRIBUTES Procedure Parameters

Parameter	Description
dad_names	The name of the DAD
attr_names	The attribute names
attr_values	The attribute values

### Exceptions

Raises an error if DAD does not exist.

### Usage Notes

If the DAD has no attributes set, then `attr_names` and `attr_values` will be set to empty arrays.

## GET\_ALL\_DAD\_MAPPINGS Procedure

This procedure retrieves all virtual paths to which the specified DAD is mapped.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.GET_ALL_DAD_MAPPINGS (  
    dad_name      IN          VARCHAR2,  
    paths         OUT NOCOPY VARCHAR2_TABLE);
```

### Parameters

**Table 63–12** GET\_ALL\_DAD\_MAPPINGS Procedure Parameters

Parameter	Description
dad_names	The name of the DAD
paths	The virtual paths to which h the DAD is mapped

### Exceptions

Raises an error if DAD does not exist.

### Usage Notes

If the DAD is not mapped to any virtual path, `paths` will be set to empty arrays.

## GET\_ALL\_GLOBAL\_ATTRIBUTES Procedure

This procedure retrieves all global attributes and values. The outputs are 2 correlated index-by tables of the name/value pairs.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.GET_ALL_GLOBAL_ATTRIBUTES (
  attr_names      OUT  NOCOPY  VARCHAR2_TABLE,
  attr_values     OUT  NOCOPY  VARCHAR2_TABLE);
```

### Parameters

**Table 63–13** GET\_ALL\_GLOBAL\_ATTRIBUTES Procedure Parameters

Parameter	Description
attr_names	The global attribute names
attr_values	The values of the global attributes

### Usage Notes

If the gateway instance has no global attributes set, then attr\_names and attr\_values will be set to empty arrays.

## GET\_DAD\_ATTRIBUTE Function

This procedure retrieves the value of a DAD attribute.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.GET_DAD_ATTRIBUTE (  
    dad_name      IN  VARCHAR2,  
    attr_name     IN  VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 63–14** GET\_DAD\_ATTRIBUTE Function Parameters

Parameter	Description
dad_name	The name of the DAD for which to delete an attribute
attr_name	The name of the attribute to delete

### Return values

Returns the DAD attribute value. Returns NULL if attribute is unknown or has not been set.

### Exceptions

Raises an error if DAD does not exist.

## GET\_DAD\_LIST Procedure

This procedure retrieves a list of all DADs for an Embedded Gateway instance.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.GET_DAD_LIST (  
    dad_names      OUT NOCOPY  VARCHAR2_TABLE);
```

### Parameters

**Table 63–15** *GET\_DAD\_LIST Procedure Parameters*

Parameter	Description
dad_names	The list of all DADs

### Usage Notes

If no DADs exist then dad\_names will be set to an empty array.

## GET\_GLOBAL\_ATTRIBUTE Function

This function retrieves the value of a global attribute.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.GET_GLOBAL_ATTRIBUTE (  
    attr_name IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 63–16** GET\_GLOBAL\_ATTRIBUTE Procedure Parameters

Parameter	Description
attr_name	The global attribute to retrieve

### Return Values

Returns the global attribute value. Returns NULL if attribute has not been set or is not a valid attribute.

## MAP\_DAD Procedure

This procedure maps a DAD to the specified virtual path. If the virtual path exists already, the old virtual-path mapping will be overridden.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.MAP_DAD (  
    dad_name IN VARCHAR2,  
    path     IN VARCHAR2);
```

### Parameters

**Table 63–17** MAP\_DAD Procedure Parameters

Parameter	Description
dad_name	The name of the DAD to map
path	The virtual path to map

### Exceptions

Raises an error if the DAD does not exist.

## SET\_DAD\_ATTRIBUTE Procedure

This procedure sets the value for a DAD.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.SET_DAD_ATTRIBUTE (
  dad_name     IN  VARCHAR2,
  attr_name    IN  VARCHAR2,
  attr_value   IN  VARCHAR2);
```

### Parameters

**Table 63–18 SET\_DAD\_ATTRIBUTE Procedure Parameters**

Parameter	Description
dad_name	The name of the DAD for which to set the attribute
attr_name	The name of the attribute to set
attr_value	The attribute value to set

**Table 63–19 Mapping Between mod\_plsql and Embedded PL/SQL Gateway DAD Attributes**

mod_plsql DAD Attribute	Embedded PL/SQL Gateway DAD Attribute	Allows Multiple Occurrences	Legal Values
PlsqlAfterProcedure	after-procedure	No	String
PlsqlAlwaysDescribeProcedure	always-describe-procedure	No	Enumeration of On, Off
PlsqlAuthenticationMode	authentication-mode	No	Enumeration of Basic, SingleSignOn, GlobalOwa, CustomOwa, PerPackageOwa
PlsqlBeforeProcedure	before-procedure	No	String
PlsqlBindBucketLengths	bind-bucket-lengths	Yes	Unsigned integer
PlsqlBindBucketWidths	bind-bucket-widths	Yes	Unsigned integer
PlsqlCGIEnvironmentList	cgi-environment-list	Yes	String
PlsqlCompatibilityMode	compatibility-mode	No	Unsigned integer
PlsqlDatabaseUsername	database-username	No	String
PlsqlDefaultPage	default-page	No	String
PlsqlDocumentPath	document-path	No	String
PlsqlDocumentProcedure	document-procedure	No	String
PlsqlDocumentTablename	document-table-name	No	String



**Table 63–19 (Cont.) Mapping Between mod\_plsql and Embedded PL/SQL Gateway DAD Attributes**

<b>mod_plsql DAD Attribute</b>	<b>Embedded PL/SQL Gateway DAD Attribute</b>	<b>Allows Multiple Occurrences</b>	<b>Legal Values</b>
PlsqlErrorStyle	error-style	No	Enumeration of ApacheStyle, ModplsqlStyle, DebugStyle
PlsqlExclusionList	exclusion-list	Yes	String
PlsqlFetchBufferSize	fetch-buffer-size	No	Unsigned integer
PlsqlInfoLogging	info-logging	No	Enumeration of InfoDebug
PlsqlOWADebugEnable	owa-debug-enable	No	Enumeration of On, Off
PlsqlMaxRequestsPerSession	max-requests-per-session	No	Unsigned integer
PlsqlNLSLanguage	nls-language	No	String
PlsqlPathAlias	path-alias	No	String
PlsqlPathAliasProcedure	path-alias-procedure	No	String
PlsqlRequestValidationFunction	request-validation-function	No	String
PlsqlSessionCookieName	session-cookie-name	No	String
PlsqlSessionStateManagement	session-state-management	No	Enumeration of StatelessWithResetPackageState, StatelessWithFastRestPackageState, StatelessWithPreservePackageState
PlsqlTransferMode	transfer-mode	No	Enumeration of Char, Raw
PlsqlUploadAsLongRaw	upload-as-long-raw	No	String

## Exceptions

Raises an error if DAD does not exist or the attribute is unknown.

## Usage Notes

- If `attr_name` attribute has been set before, then the old value will be overwritten with the new `attr_value` argument.
- The embedded gateway assumes default values when the attributes are not set. The default values of the DAD attributes should be sufficient for most users of the embedded gateway. `mod_plsql` users should note the following
  - The `PlsqlDatabasePassword` attribute is not needed.
  - The `PlsqlDatabaseConnectString` attribute is not needed because the embedded gateway does not support logon to external databases.

## Examples

```
DBMS_EPG.SET_DAD_ATTRIBUTE('HR', 'default-page', 'HRApp.home');
```

## SET\_GLOBAL\_ATTRIBUTE Procedure

This procedure sets the value of a global attribute.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.SET_GLOBAL_ATTRIBUTE (
    attr_name    IN VARCHAR2,
    attr_value   IN VARCHAR2);
```

### Parameters

**Table 63–20** SET\_GLOBAL\_ATTRIBUTE Procedure Parameters

Parameter	Description
attr_name	The global attribute to set
attr_value	The attribute value to set

**Table 63–21** Mapping Between mod\_plsql and Embedded PL/SQL Gateway Global Attributes

mod_plsql Global Attribute	Embedded PL/SQL Gateway Global Attribute	Allows Multiple Occurrences	Legal Values
PlsqlLogLevel	log-level	No	Unsigned integer
PlsqlMaxParameters	max-parameters	No	Unsigned integer

### Usage Notes

- The attribute name is case sensitive. The value may or may not be case-sensitive depending on the attribute.
- If attr\_name attribute has been set before, then the old value will be overwritten with the new attr\_value argument.

### Exceptions

Raises an error if the attribute is unknown.

### Examples

```
dbms_epg.set_global_attribute('max-parameters', '100');
```

## UNMAP\_DAD Procedure

This procedure unmaps a DAD from the specified virtual path. If path is `NULL`, the procedure removes all virtual-path mappings for the DAD but keeps the DAD.

**See Also:** [Configuration Subprograms](#) on page 63-8 for other subprograms in this group

### Syntax

```
DBMS_EPG.UNMAP_DAD (  
    dad_name IN VARCHAR2,  
    path     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 63–22 UNMAP\_DAD Procedure Parameters**

Parameter	Description
dad_name	The name of the DAD to unmap
path	The virtual path to unmap

### Usage Notes

Raises an error if the DAD does not exist.



The DBMS\_ERRLOG package provides a procedure that enables you to create an error logging table so that DML operations can continue after encountering errors rather than abort and roll back. This enables you to save time and system resources.

**See Also:** *Oracle Database Data Warehousing Guide* for more information regarding how to use DBMS\_ERRLOG and *Oracle Database SQL Language Reference* for `error_logging_clause` syntax

This chapter contains the following topics:

- [Using DBMS\\_ERRLOG](#)
  - Security Model
- [Summary of DBMS\\_ERRLOG Subprograms](#)

---

## Using DBMS\_ERRLOG

This section contains topics which relate to using the DBMS\_ERRLOG package.

- [Security Model](#)

## Security Model

Security on this package can be controlled by granting `EXECUTE` on this package to selected users or roles. The `EXECUTE` privilege is granted publicly. However, to create an error logging table, you need `SELECT` access on the base table or view, the `CREATE TABLE` privilege, as well as tablespace quota for the target tablespace.

---

## Summary of DBMS\_ERRLOG Subprograms

**Table 64–1** *DBMS\_ERRLOG Package Subprograms*

Subprogram	Description
<a href="#">CREATE_ERROR_LOG Procedure</a> on page 64-5	Creates the error logging table used in DML error logging



## CREATE\_ERROR\_LOG Procedure

This procedure creates the error logging table needed to use the DML error logging capability.

LONG, CLOB, BLOB, BFILE, and ADT datatypes are not supported in the columns.

### Syntax

```
DBMS_ERRLOG.CREATE_ERROR_LOG (
    dml_table_name          IN VARCHAR2,
    err_log_table_name      IN VARCHAR2 := NULL,
    err_log_table_owner     IN VARCHAR2 := NULL,
    err_log_table_space     IN VARCHAR2 := NULL,
    skip_unsupported       IN BOOLEAN := FALSE);
```

### Parameters

**Table 64–2 CREATE\_ERROR\_LOG Procedure Parameters**

Parameter	Description
dml_table_name	The name of the DML table to base the error logging table on. The name can be fully qualified (for example, emp, scott.emp, "EMP", "SCOTT"."EMP"). If a name component is enclosed in double quotes, it will not be upper cased.
err_log_table_name	The name of the error logging table you will create. The default is the first 25 characters in the name of the DML table prefixed with 'ERR\$_'. Examples are the following: dml_table_name: 'EMP', err_log_table_name: 'ERR\$_EMP' dml_table_name: '"Emp2"', err_log_table_name: 'ERR\$_Emp2'
err_log_table_owner	The name of the owner of the error logging table. You can specify the owner in dml_table_name. Otherwise, the schema of the current connected user is used.
err_log_table_space	The tablespace the error logging table will be created in. If not specified, the default tablespace for the user owning the DML error logging table will be used.
skip_unsupported	When set to TRUE, column types that are not supported by error logging will be skipped over and not added to the error logging table. When set to FALSE, an unsupported column type will cause the procedure to terminate. The default is FALSE.

### Examples

First, create an error log table for the channels table in the SH schema, using the default name generation.

Then, see all columns of the table channels:

```
SQL> DESC channels
Name                               Null?      Type
-----
CHANNEL_ID                          NOT NULL   CHAR(1)
```

```

CHANNEL_DESC                NOT NULL    VARCHAR2 (20)
CHANNEL_CLASS                VARCHAR2 (20)

```

Finally, see all columns of the generated error log table. Note the mandatory control columns that are created by the package:

```

SQL> DESC ERR$_CHANNELS
Name                               Null?    Type
-----
ORA_ERR_NUMBER$                    NUMBER
ORA_ERR_MSGS$                      VARCHAR2 (2000)
ORA_ERR_ROWID$                     ROWID
ORA_ERR_OPTYP$                     VARCHAR2 (2)
ORA_ERR_TAG$                       VARCHAR2 (2000)
CHANNEL_ID                         VARCHAR2 (4000)
CHANNEL_DESC                       VARCHAR2 (4000)
CHANNEL_CLASS                      VARCHAR2 (4000)

```

See *Oracle Database Administrator's Guide* for more information regarding control columns.

---

---

**Note:** This functionality is deprecated with Oracle Database Release 11.2 and obsoleted with Release 12.1. For details regarding obsolescence, see My Oracle Support Note ID 1244535.1

---

---

The `DBMS_EXPFIL` package contains all the procedures used to manage attribute sets, expression sets, expression indexes, optimizer statistics, and privileges by Expression Filter.

**See Also:** *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information.

This chapter contains the following topics:

- [Using DBMS\\_EXPFIL](#)
- [Summary of Expression Filter Subprograms](#)

---

## Using DBMS\_EXPFIL

This section contains topics that relate to using the Rules Manager DBMS\_EXPFIL package.

- [Security Model](#)

## Security Model

The Oracle Database installation runs the `catexf.sql` script to load the `DBMS_EXPFIL` package and create the required Expression Filter schema objects in the `EXFSYS` schema.

`DBMS_EXPFIL` is an `EXFSYS`-owned package compiled with `AUTHID CURRENT_USER`. Any `DBMS_EXPFIL` subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

Before you issue `COPY_ATTRIBUTE_SET` procedure, the user must have the `EXECUTE` privilege for the object type associated with the original attribute set.

A user requires `SELECT` privileges on a table storing expressions to evaluate them. The `SQL EVALUATE` operator evaluates expressions with the privileges of the owner of the table that stores the expressions. The privileges of the user issuing the query are not considered. The owner of the table can insert, update, and delete expressions. Other users must have `INSERT` and `UPDATE` privileges for the table and `INSERT EXPRESSION` and `UPDATE EXPRESSION` privilege for a specific Expression column in the table to be able to make modifications to it.

Using the `GRANT_PRIVILEGE` procedure, the owner of the table can grant `INSERT EXPRESSION` or `UPDATE EXPRESSION` privileges on one or more Expression columns to other users. Both privileges can be granted to a user by specifying `ALL` for the privilege type.

A user with `CREATE INDEX` privileges on a table cannot create an Expression Filter index unless the user is the owner of the table.

A user must have `EXECUTE` privilege on the `CTX_DDL` package for successful synchronization of the text indexes using the `DBMS_EXPFIL.SYNC_TEXT_INDEXES` procedure.

The `USER_EXPFIL_PRIVILEGES` view lists the privileges of the current user on expression sets belonging to other schemas and the privileges of other users on the expression sets owned by the current user.

## Summary of Expression Filter Subprograms

Table 65–1 describes the subprograms in the DBMS\_EXPFIL package.

All the values and names passed to the procedures defined in the DBMS\_EXPFIL package are not case sensitive, unless otherwise mentioned. To preserve the case, you use double quotation marks around the values.

**Table 65–1 DBMS\_EXPFIL Package Subprograms**

Subprogram	Description
<a href="#">ADD_ELEMENTARY_ATTRIBUTE Procedures</a>	Adds the specified attribute to the attribute set
<a href="#">ADD_FUNCTIONS Procedure</a>	Adds a function, type, or package to the approved list of functions with an attribute set
<a href="#">ASSIGN_ATTRIBUTE_SET Procedure</a>	Assigns an attribute set to a column storing expressions
<a href="#">BUILD_EXCEPTIONS_TABLE Procedure</a>	Creates an exception table to hold references to invalid expressions
<a href="#">CLEAR_EXPRSET_STATS Procedure</a>	Clears the predicate statistics for an expression set
<a href="#">COPY_ATTRIBUTE_SET Procedure</a>	Makes a copy of the attribute set
<a href="#">CREATE_ATTRIBUTE_SET Procedure</a>	Creates an attribute set
<a href="#">DEFAULT_INDEX_PARAMETERS Procedure</a>	Assigns default index parameters to an attribute set
<a href="#">DEFAULT_XINDEX_PARAMETERS Procedure</a>	Assigns default XPath index parameters to an attribute set
<a href="#">DEFRAG_INDEX Procedure</a>	Rebuilds the bitmap indexes online to reduce fragmentation
<a href="#">DROP_ATTRIBUTE_SET Procedure</a>	Drops an unused attribute set
<a href="#">GET_EXPRSET_STATS Procedure</a>	Collects predicate statistics for an expression set
<a href="#">GRANT_PRIVILEGE Procedure</a>	Grants an expression DML privilege to a user
<a href="#">INDEX_PARAMETERS Procedure</a>	Assigns index parameters to an expression set
<a href="#">MODIFY_OPERATOR_LIST Procedure</a>	Modifies the list of common operators used in predicates with a certain attribute
<a href="#">REVOKE_PRIVILEGE Procedure</a>	Revokes an expression DML privilege from a user
<a href="#">SYNC_TEXT_INDEXES Procedure</a>	Synchronizes the indexes defined to process the predicates involving the CONTAINS operator in stored expressions
<a href="#">UNASSIGN_ATTRIBUTE_SET Procedure</a>	Breaks the association between a column storing expressions and the attribute set
<a href="#">VALIDATE_EXPRESSIONS Procedure</a>	Validates expression metadata and the expressions stored in a column
<a href="#">XINDEX_PARAMETERS Procedure</a>	Assigns XPath index parameters to an expression set

## ADD\_ELEMENTARY\_ATTRIBUTE Procedures

This procedure adds the specified attribute to the attribute set. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Adds the specified elementary attribute to the attribute set:

```
DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
  attr_set   IN   VARCHAR2,
  attr_name  IN   VARCHAR2,
  attr_type  IN   VARCHAR2,
  attr_defv1 IN   VARCHAR2 DEFAULT NULL);
```

Identifies the elementary attributes that are table aliases and adds them to the attribute set:

```
DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
  attr_set   IN   VARCHAR2,
  attr_name  IN   VARCHAR2,
  tab_alias  IN   exf$table_alias);
```

Allows addition of text attributes to the attribute set:

```
DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
  attr_set   IN   VARCHAR2,
  attr_name  IN   VARCHAR2,
  attr_type  IN   VARCHAR2,
  text_pref  IN   EXF$TEXT);
```

### Parameters

**Table 65–2 ADD\_ELEMENTARY\_ATTRIBUTE Procedure Parameters**

Parameter	Description
attr_set	Name of the attribute set to which this attribute is added
attr_name	Name of the elementary attribute to be added. No two attributes in a set can have the same name.
attr_type	Datatype of the attribute. This argument accepts any standard SQL datatype or the name of an object type that is accessible to the current user.
attr_defv1	Default value for the elementary attribute
tab_alias	Type that identifies the database table to which the attribute is aliased
text_pref	Text preferences such as LEXER and WORDLIST specification.

### Usage Notes

- If the attribute set receiving the elementary attribute was originally created from an existing object type, then additional attributes cannot be added.
- One or more, or all elementary attributes in an attribute set can be table aliases. If an elementary attribute is a table alias, then the value assigned to the elementary attribute is a ROWID from the corresponding table. An attribute set with one or more table alias attributes cannot be created from an existing object type. For more

information about table aliases, see *Oracle Database Rules Manager and Expression Filter Developer's Guide*.

- Elementary attributes cannot be added to an attribute set that is already assigned to a column storing expressions.
- The default value specification for an attribute is similar to a default value specification for a table column. The resulting default values should agree with the datatype of the attribute. For example, valid default values for an attribute of DATE datatype are SYSDATE and `to_date('01-01-2004', 'DD-MM-YYYY')`.
- See the section on defining attribute sets in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about adding elementary attributes.
- Related views: USER\_EXPFIL\_ATTRIBUTE\_SETS and USER\_EXPFIL\_ATTRIBUTES.
- This procedure with a text preference bound to the `text_pref` argument creates a text attribute in the attribute set. The data type for such an attribute should be a VARCHAR2 or a CLOB. The preferences specified with this procedure are validated only when an Expression Filter index is created using this attribute set.
- These preferences are used in the creation of the CTXRULE index that processes the predicates involving CONTAINS operator. The valid preferences are those that are valid in the PARAMETERS clause of CTXRULE index creation. See *Oracle Text Application Developer's Guide* for the syntax.

## Examples

The following commands add two elementary attributes to an attribute set:

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                                attr_set => 'HRAttrSet',
                                attr_name => 'HRREP',
                                attr_type => 'VARCHAR2(30)',
                                attr_defv1 => 'Betty Smith');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                                attr_set => 'HRAttrSet',
                                attr_name => 'DEPT',
                                tab_alias => exf$table_alias('DEPT'));
END;
/
```

The following commands define a CreationTime elementary attribute that takes the database time as the default value.

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
                                attr_set => 'PurchaseOrder',
                                attr_name => 'CreationTime',
                                attr_type => 'DATE',
                                attr_defv1 => 'SYSDATE');
END;
/
```

Alternately, the following commands initialize the CreationTime attribute to a specific value when it is not explicitly specified in the data item passed to the EVALUATE operator.

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
```



```

                                attr_set => 'PurchaseOrder',
                                attr_name => 'CreationTime',
                                attr_type => 'DATE',
                                attr_defvl => 'to_date(''01-01-2004'', ''DD-MM-YYYY'')');
END;
/

```

The following commands create an attribute set with a Model attribute of VARCHAR2 data type and an InsReport attribute configured for text predicates.

```

BEGIN
  DBMS_EXPFIL.CREATE_EVENT_STRUT(event_struct => 'Car4Sale');
  // create scalar attributes
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
    attr_set => 'Car4Sale',
    attr_name => 'Model',
    attr_type => 'VARCHAR2(30)');

  //create text attribute
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
    attr_set => 'Car4Sale',
    attr_name => 'InsReport',
    attr_type => 'CLOB',
    attr_type => exf$text(
      'LEXER insrpt_lexer
      WORDLIST insrpt_wordlist'));
END;
/

```

## ADD\_FUNCTIONS Procedure

This procedure adds a user-defined function, package, or type representing a set of functions to the attribute set.

### Syntax

```
DBMS_EXPFIL.ADD_FUNCTIONS (
    attr_set IN VARCHAR2,
    funcs_name IN VARCHAR2);
```

### Parameters

**Table 65–3 ADD\_FUNCTIONS Procedure Parameters**

Parameter	Description
attr_set	Name of the attribute set to which the functions are added
funcs_name	Name of a function, package, or type (representing a function set) or its synonyms

### Usage Notes

- By default, an attribute set implicitly allows references to all Oracle Database supplied SQL functions for use by the expression set. If the expression set refers to a user-defined function, the function must be explicitly added to the attribute set.
- The ADD\_FUNCTIONS procedure adds a user-defined function or a package (or type) representing a set of functions to the attribute set. Any new or modified expressions are validated using this list. The function added to the attribute set, and thus used in the stored expressions, should not perform any DML or DDL (database state changing) operations. Oracle Database catches violations to this rule only at run-time while evaluating the expressions. Violations are not checked during the ADD\_FUNCTIONS procedure call.
- The function or the package name can be specified with a schema extension. If a function name is specified without a schema extension, only such references in the expression set are considered valid. The expressions in a set can be restricted to use a synonym to a function or a package by adding the corresponding synonym to the attribute set. This preserves the portability of the expression set to other schemas.
- See the section on defining attribute sets in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about adding functions to an attribute set.
- Related views: USER\_EXPFIL\_ATTRIBUTE\_SETS and USER\_EXPFIL\_EXPRESSION\_SETS

### Examples

The following commands add two functions to the attribute set:

```
BEGIN
    DBMS_EXPFIL.ADD_FUNCTIONS (
        attr_set => 'Car4Sale',
        funcs_name => 'HorsePower');
    DBMS_EXPFIL.ADD_FUNCTIONS (
        attr_set => 'Car4Sale',
```

```
    funcs_name => 'Scott.CrashTestRating');  
END;  
/
```

## ASSIGN\_ATTRIBUTE\_SET Procedure

This procedure assigns an attribute set to a VARCHAR2 column in a user table to create an Expression column.

### Syntax

```
DBMS_EXPFIL.ASSIGN_ATTRIBUTE_SET (
  attr_set   IN   VARCHAR2,
  expr_tab   IN   VARCHAR2,
  expr_col   IN   VARCHAR2,
  force      IN   VARCHAR2 DEFAULT 'FALSE');
```

### Parameters

**Table 65–4** *ASSIGN\_ATTRIBUTE\_SET Procedure Parameters*

Parameter	Description
attr_set	Name of the attribute set
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions
force	Argument used to trust the existing expressions in a table (and skip validation)

### Usage Notes

- The ASSIGN\_ATTRIBUTE\_SET procedure assigns an attribute set to a VARCHAR2 column in a user table to create an Expression column. The attribute set contains the elementary attribute names and their data types and any functions used in the expressions. The Expression column, the column named by the expr\_col parameter to store the expression, uses the attribute set to validate changes and additions to the expression set.
- An attribute set can be assigned only to a table column in the same schema as the attribute set. An attribute set can be assigned to one or more table columns. Assigning an attribute set to a column storing expressions implicitly creates methods for the associated object type. For this operation to succeed, the object type cannot have any dependent objects before the attribute set is assigned.
- By default, the column should not have any expressions at the time of association. However, if the values in the column are known to be valid expressions, you can use a value of 'TRUE' for the force argument to assign the attribute set to a column containing expressions.
- See the information about defining Expression columns in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about adding elementary attributes.
- Related views: USER\_EXPFIL\_ATTRIBUTE\_SETS and USER\_EXPFIL\_EXPRESSION\_SETS

### Examples

The following command assigns the attribute set to a column storing expressions. The expression set should be empty at the time of association.

```
BEGIN
```

```
DBMS_EXPFIL.ASSIGN_ATTRIBUTE_SET (attr_set => 'Car4Sale',  
                                  expr_tab => 'consumer',  
                                  expr_col => 'interest');  
  
END;  
/
```

## BUILD\_EXCEPTIONS\_TABLE Procedure

This procedure creates the exception table, used in validation, in the current schema.

### Syntax

```
DBMS_EXPFIL.BUILD_EXCEPTIONS_TABLE (  
    exception_tab IN VARCHAR2);
```

### Parameters

**Table 65–5** *BUILD\_EXCEPTIONS\_TABLE Procedure Parameter*

Parameter	Description
exception_tab	Name of the exception table

### Usage Notes

- During expression validation, you can optionally provide the name of the exception table in which the references to the invalid expressions are stored. The `BUILD_EXCEPTIONS_TABLE` procedure creates the exception table in the current schema.
- See the section on evaluation semantics in *Oracle Database Rules Manager and Expression Filter Developer's Guide* and [VALIDATE\\_EXPRESSIONS Procedure](#) for more information.
- Related view: `USER_TABLES`

### Examples

The following command creates the exception table, `InterestExceptions`, in the current schema:

```
BEGIN  
    DBMS_EXPFIL.BUILD_EXCEPTIONS_TABLE (exception_tab => 'InterestExceptions');  
END;  
/
```

## CLEAR\_EXPRSET\_STATS Procedure

This procedure clears the predicate statistics for the expression set stored in a table column.

### Syntax

```
DBMS_EXPFIL.CLEAR_EXPRSET_STATS (
  expr_tab  IN  VARCHAR2,
  expr_col  IN  VARCHAR2);
```

### Parameters

**Table 65–6** CLEAR\_EXPRSET\_STATS Procedure Parameters

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions

### Usage Notes

- See also [GET\\_EXPRSET\\_STATS Procedure](#) for information about gathering the statistics.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS and USER\_EXPFIL\_EXPRSET\_STATS

### Examples

The following command clears the predicate statistics for the expression set stored in interest column of the consumer table:

```
BEGIN
  DBMS_EXPFIL.CLEAR_EXPRSET_STATS (expr_tab => 'consumer',
                                   expr_col => 'interest');
END;
/
```

## COPY\_ATTRIBUTE\_SET Procedure

This procedure copies an attribute set along with its user-defined function list and default index parameters to another set.

### Syntax

```
DBMS_EXPFIL.COPY_ATTRIBUTE_SET (
  from_set  IN  VARCHAR2,
  to_set    IN  VARCHAR2);
```

### Parameters

**Table 65–7** *COPY\_ATTRIBUTE\_SET Procedure Parameters*

Parameter	Description
from_set	Name of an existing attribute set to be copied
to_set	Name of the new attribute set

### Usage Notes

- A schema-extended name can be used for the `from_set` argument to copy an attribute set across schemas. Before you issue this command, you must have the EXECUTE privilege for the object type associated with the original attribute set. Ensure that any references to schema objects (user-defined functions, tables, and embedded objects) are valid in the new schema.
- The default index parameters and the user-defined function list of the new set can be changed independent of the original set.
- Related views: ALL\_EXPFIL\_ATTRIBUTE\_SETS and ALL\_EXPFIL\_ATTRIBUTES.

### Examples

The following command makes a copy of the Car4Sale attribute set:

```
BEGIN
  DBMS_EXPFIL.COPY_ATTRIBUTE_SET (from_set => 'Car4Sale',
                                to_set    => 'Vehicle');
END;
/
```



## CREATE\_ATTRIBUTE\_SET Procedure

This procedure creates an empty attribute set or an attribute set with a complete set of elementary attributes derived from an object type with a matching name.

### Syntax

```
DBMS_EXPFIL.CREATE_ATTRIBUTE_SET (
  attr_set   IN   VARCHAR2,
  from_type  IN   VARCHAR2 DEFAULT 'NO');
```

### Parameters

**Table 65–8 CREATE\_ATTRIBUTE\_SET Procedure Parameters**

Parameter	Description
attr_set	Name of the attribute set to be created
from_type	YES, if the attributes for the attribute set should be derived from an existing object type

### Usage Notes

- The object type used for an attribute set cannot contain any user methods, and it should not be an evolved type (with the use of `ALTER TYPE` command). This object type should not have any dependent objects at the time of the attribute set creation. If the attribute set is not derived from an existing object type, this procedure creates an object type with a matching name.
- An attribute set with one or more table alias attributes cannot be derived from an object type. For this purpose, create an empty attribute set and add one elementary attribute at a time using the `DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE` procedure. (See *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information.)
- See the section on defining attribute sets in *Oracle Database Rules Manager and Expression Filter Developer's Guide* and [ADD\\_ELEMENTARY\\_ATTRIBUTE Procedures](#) for more information.
- Related views: `USER_EXPFIL_ATTRIBUTE_SETS` and `USER_EXPFIL_ATTRIBUTES`.

### Examples

The following commands create an attribute set with all the required elementary attributes derived from the `Car4Sale` type:

```
CREATE OR REPLACE TYPE Car4Sale AS OBJECT
    (Model   VARCHAR2(20),
     Year    NUMBER,
     Price   NUMBER,
     Mileage NUMBER);
/

BEGIN
  DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set => 'Car4Sale',
                                  from_type => 'YES');
END;
/
```

Assuming that the `Car4Sale` type does not exist, the attribute set can be created from scratch as shown in the following example:

```
BEGIN
  DBMS_EXPFIL.CREATE_ATTRIBUTE_SET(attr_set => 'Car4Sale');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
    attr_set => 'Car4Sale',
    attr_name => 'Model',
    attr_type => 'VARCHAR2(20)');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
    attr_set => 'Car4Sale',
    attr_name => 'Year',
    attr_type => 'NUMBER');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
    attr_set => 'Car4Sale',
    attr_name => 'Price',
    attr_type => 'NUMBER');
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE(
    attr_set => 'Car4Sale',
    attr_name => 'Mileage',
    attr_type => 'NUMBER');
END;
/
```

## DEFAULT\_INDEX\_PARAMETERS Procedure

This procedure assigns default index parameters to an attribute set. It also adds or drops a partial list of stored and indexed attributes to or from the default list associated with the attribute list.

### Syntax

```
DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS (
  attr_set   IN   VARCHAR2,
  attr_list  IN   EXF$ATTRIBUTE_LIST,
  operation  IN   VARCHAR2 DEFAULT 'ADD');
```

### Parameters

**Table 65–9** *DEFAULT\_INDEX\_PARAMETERS Procedure Parameters*

Parameter	Description
attr_set	Name of the attribute set
attr_list	An instance of EXF\$ATTRIBUTE_LIST with a partial list of (default) stored and indexed attributes for an Expression Filter index
operation	Operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

### Usage Notes

- Existing Expression Filter indexes are not modified when the default parameters for the corresponding attribute set are changed. The new index defaults are used when a new Expression Filter index is created and when an existing index is rebuilt. (See the section on alter index rebuild in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about rebuilding indexes.)
- See the section on creating an index from default parameters in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about assigning default index parameters to an attribute set.
- Related views: USER\_EXPFIL\_ATTRIBUTE\_SETS and USER\_EXPFIL\_DEF\_INDEX\_PARAMS

### Examples

The following command adds the specified stored and indexed attributes to the attribute set's default index parameters list:

```
BEGIN
  DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS (
    attr_set => 'Car4Sale',
    attr_list => exf$attribute_list (
      exf$attribute (attr_name => 'Model',
                    attr_oper => exf$indexoper('='),
                    attr_indexed => 'TRUE'),
      exf$attribute (attr_name => 'Price',
                    attr_oper => exf$indexoper('all'),
                    attr_indexed => 'TRUE'),
      exf$attribute (attr_name => 'HorsePower(Model, Year)',
                    attr_oper => exf$indexoper('=', '<', '>', '>=', '<='),
                    attr_indexed => 'FALSE'),
      exf$attribute (attr_name => 'CrashTestRating(Model, Year)',
```

```

                attr_oper => exf$indexoper('=', '<', '>', '>=', '<='),
                attr_indexed => 'FALSE')),
        operation => 'ADD');
END;
/

```

The following command drops the `CrashTestRating(Model, Year)` attribute (stored or indexed) from the previous list.

```

BEGIN
  DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS(
    attr_set => 'Car4Sale',
    attr_list => exf$attribute_list (
      exf$attribute (attr_name => 'CrashTestRating(Model, Year)'),
    operation => 'DROP');
END;
/

```

## DEFAULT\_XPINDEX\_PARAMETERS Procedure

This procedure adds (or drops) a partial list of XPath parameters to the default index parameters associated with the attribute set.

### Syntax

```
DBMS_EXPFIL.DEFAULT_XPINDEX_PARAMETERS (
  attr_set   IN   VARCHAR2,
  xmlt_attr  IN   VARCHAR2,
  xptag_list IN   EXF$XP_PATH_TAGS,
  operation  IN   VARCHAR2 DEFAULT 'ADD');
```

### Parameters

**Table 65–10** DEFAULT\_XPINDEX\_PARAMETERS Procedure Parameters

Parameter	Description
attr_set	Name of the attribute set
xmlt_attr	Name of the attribute with the XMLType datatype
xptag_list	An instance of EXF\$XP_PATH_TAGS type with a partial list of XML elements and attributes to be configured for the Expression Filter index
operation	Operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

### Usage Notes

- The attribute set used for an expression set may have one or more XML type attributes (defined with XMLType datatype) and the corresponding expressions may contain XPath predicates on these attributes. The Expression Filter index created for the expression set can be tuned to process these XPath predicates efficiently by using some XPath-specific index parameters (in addition to some non-XPath index parameters).
- The DEFAULT\_XPINDEX\_PARAMETERS procedure adds (or drops) a partial list of XPath parameters to the default index parameters associated with the attribute set. The XPath parameters are assigned to a specific XMLType attribute in the attribute set and this information can be viewed using the USER\_EXPFIL\_DEF\_INDEX\_PARAMS view. The DEFAULT\_INDEX\_PARAMETERS procedure and the DEFAULT\_XPINDEX\_PARAMETERS procedure can be used independent of each other. They maintain a common list of default index parameters for the attribute set.
- See the section on index tuning for XPath predicates in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about XPath parameters to the default index parameters of an attribute set. See also [DEFAULT\\_INDEX\\_PARAMETERS Procedure](#) for more information about default index parameters.
- Related views: USER\_EXPFIL\_ATTRIBUTES and USER\_EXPFIL\_DEF\_INDEX\_PARAMS.

---

**Note:** The values assigned to the tag\_name argument of exf\$xmlt\_tag type are case sensitive.

---

## Examples

The following command adds the specified XML tags to the default index parameters list along with their preferences such as positional or value filter and indexed or stored predicate group:

```
BEGIN
  DBMS_EXPFIL.DEFAULT_XPINDEX_PARAMETERS(
    attr_set => 'Car4Sale',
    xmlt_attr => 'Details',
    xptag_list =>                                --- XPath tag list
    exf$xpath_tags(
      exf$xpath_tag(tag_name => 'stereo@make', --- XML attribute
                    tag_indexed => 'TRUE',
                    tag_type => 'VARCHAR(15)'), --- value filter
      exf$xpath_tag(tag_name => 'stereo', --- XML element
                    tag_indexed => 'FALSE',
                    tag_type => null), --- positional filter
      exf$xpath_tag(tag_name => 'memory', --- XML element
                    tag_indexed => 'TRUE',
                    tag_type => 'VARCHAR(10)'), --- value filter
      exf$xpath_tag(tag_name => 'GPS',
                    tag_indexed => 'TRUE',
                    tag_type => null)
    )
  );
END;
/
```

The following command drops the stereo@make tag from the default index parameters:

```
BEGIN
  DBMS_EXPFIL.DEFAULT_XPINDEX_PARAMETERS(
    attr_set => 'Car4Sale',
    xmlt_attr => 'Details',
    xptag_list =>                                --- XPath tag list
    exf$xpath_tags(
      exf$xpath_tag(tag_name => 'stereo@make')
    ),
    operation => 'DROP'
  );
END;
/
```

## DEFRAG\_INDEX Procedure

This procedure rebuilds the bitmap indexes online and thus reduces the fragmentation.

### Syntax

```
DBMS_EXPFIL.DEFRAG_INDEX (
  idx_name IN VARCHAR2);
```

### Parameters

**Table 65–11** DEFRAG\_INDEX Procedure Parameter

Parameter	Description
idx_name	Name of the Expression Filter index

### Usage Notes

- The bitmap indexes defined for the indexed attributes of an Expression Filter index become fragmented as additions and updates are made to the expression set. This can affect performance. The DEFRAG\_INDEX procedure rebuilds the bitmap indexes online and thus reduces the fragmentation.
- Indexes can be defragmented when the expression set is being modified. However, you should schedule defragmentation when the workload is relatively light, such as 2 a.m.
- See the section on index storage and maintenance in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about rebuilding indexes.
- Related views: USER\_EXPFIL\_INDEXES and USER\_INDEXES.

### Examples

The following command is issued to defragment the bitmap indexes associated with the Expression Filter index:

```
BEGIN
  DBMS_EXPFIL.DEFRAG_INDEX (idx_name => 'InterestIndex');
END;
/
```

## DROP\_ATTRIBUTE\_SET Procedure

This procedure drops an attribute set not being used for any expression set.

### Syntax

```
DBMS_EXPFIL.DROP_ATTRIBUTE_SET (  
    attr_set IN VARCHAR2);
```

### Parameters

**Table 65–12** *DROP\_ATTRIBUTE\_SET Procedure Parameter*

Parameter	Description
attr_set	Name of the attribute set to be dropped

### Usage Notes

- The `DROP_ATTRIBUTE_SET` procedure drops an attribute set not being used for any expression set. If the attribute set was initially created from an existing object type, the object type remains after dropping the attribute set. Otherwise, the object type is dropped with the attribute set.
- Related views: `USER_EXPFIL_ATTRIBUTE_SETS` and `USER_EXPFIL_EXPRESSION_SETS`.

### Examples

Assuming that the attribute set is not used by an Expression column, the following command drops the attribute set:

```
BEGIN  
    DBMS_EXPFIL.DROP_ATTRIBUTE_SET(attr_set => 'Car4Sale');  
END;  
/
```



## GET\_EXPRSET\_STATS Procedure

This procedure computes the predicate statistics for an expression set and stores them in the expression filter dictionary.

### Syntax

```
DBMS_EXPFIL.GET_EXPRSET_STATS (
  expr_tab  IN  VARCHAR2,
  expr_col  IN  VARCHAR2);
```

### Parameters

**Table 65–13** GET\_EXPRSET\_STATS Procedure Parameters

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions

### Usage Notes

- When a representative set of expressions are stored in a table column, you can use predicate statistics for those expressions to configure the corresponding Expression Filter index (using the TOP parameters clause). The GET\_EXPRSET\_STATS procedure computes the predicate statistics for an expression set and stores them in the expression filter dictionary.
- See the section on creating an index from statistics in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about using predicate statistics.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS and USER\_EXPFIL\_EXPRSET\_STATS.

### Examples

The following command computes the predicate statistics for the expressions stored in the interest column of the consumer table:

```
BEGIN
  DBMS_EXPFIL.GET_EXPRSET_STATS (expr_tab => 'consumer',
                                expr_col => 'interest');
END;
/
```

## GRANT\_PRIVILEGE Procedure

This procedure grants privileges on one or more Expression columns to other users.

### Syntax

```
DBMS_EXPFIL.GRANT_PRIVILEGE (
    expr_tab   IN  VARCHAR2,
    expr_col   IN  VARCHAR2,
    priv_type  IN  VARCHAR2,
    to_user    IN  VARCHAR2);
```

### Parameters

**Table 65–14 GRANT\_PRIVILEGE Procedure Parameters**

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions
priv_type	Type of the privilege to be granted. Valid values: INSERT EXPRESSION, UPDATE EXPRESSION, ALL.
to_user	User to whom the privilege is to be granted

### Usage Notes

- The SQL EVALUATE operator evaluates expressions with the privileges of the owner of the table that stores the expressions. The privileges of the user issuing the query are not considered. The owner of the table can insert, update, and delete expressions. Other users must have INSERT and UPDATE privileges for the table and INSERT EXPRESSION and UPDATE EXPRESSION privilege for a specific Expression column in the table.
- Using the GRANT\_PRIVILEGE procedure, the owner of the table can grant INSERT EXPRESSION or UPDATE EXPRESSION privileges on one or more Expression columns to other users. Both the privileges can be granted to a user by specifying ALL for the privilege type.
- See [REVOKE\\_PRIVILEGE Procedure](#) and the section on granting and revoking privileges in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about granting and revoking privileges.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS and USER\_EXPFIL\_PRIVILEGES.

### Examples

The owner of consumer table can grant INSERT EXPRESSION privileges to user SCOTT with the following command. User SCOTT should also have INSERT privileges on the table so that he can add new expressions to the set.

```
BEGIN
    DBMS_EXPFIL.GRANT_PRIVILEGE (expr_tab => 'consumer',
                                expr_col  => 'interest',
                                priv_type => 'INSERT EXPRESSION',
                                to_user   => 'SCOTT');
END;
```

## INDEX\_PARAMETERS Procedure

This procedure fine-tunes the index parameters for each expression set before index creation.

### Syntax

```
DBMS_EXPFIL.INDEX_PARAMETERS (
  expr_tab  IN  VARCHAR2,
  expr_col  IN  VARCHAR2,
  attr_list IN  EXF$ATTRIBUTE_LIST,
  operation IN  VARCHAR2 DEFAULT 'ADD');
```

### Parameters

**Table 65–15** INDEX\_PARAMETERS Procedure Parameters

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions.
attr_list	An instance of EXF\$ATTRIBUTE_LIST with a partial list of stored and indexed attributes
operation	Operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

### Usage Notes

- An attribute set can be used by multiple expression sets stored in different columns of user tables. By default, the index parameters associated with the attribute set are used to define an Expression Filter index on an expression set. If you need to fine-tune the index for each expression set, you can specify a small list of the index parameters in the PARAMETERS clause of the CREATE INDEX statement. However, when an Expression Filter index uses a large number of index parameters or if the index is configured for XPath predicates, fine-tuning the parameters with the CREATE INDEX statement is not possible.
- The INDEX\_PARAMETERS procedure fine-tunes the index parameters for each expression set before index creation. This procedure can be used to copy the defaults from the corresponding attribute set and selectively add (or drop) additional index parameters for the expression set. (You use the XPINDEX\_PARAMETERS procedure to add and drop XPath index parameters.) The Expression Filter index defined for an expression set with a non-empty list of index parameters always uses these parameters. The INDEX\_PARAMETERS procedure cannot be used when the Expression Filter index is already defined for the column storing expressions.
- The operations allowed with this procedure include:
  - Deriving the current list of default index parameters (including any XPath-specific parameters) from the corresponding attribute set and assigning them to the specified expression set (a value of DEFAULT for the operation argument).
  - Adding (or dropping) one or more attributes to (or from) the current list of parameters assigned to the expression set (values of ADD or DROP for the operation argument).

- Clearing the index parameters assigned to the expression set. This enables the user to start using default parameters or tune the parameters from scratch (a value of CLEAR for the operation argument).

---

**Note:** This procedure is useful only when an attribute set is shared across multiple expression sets. In all other cases, the defaults assigned to the attribute set can be tuned for the expression set using it.

---

- See the section on creating an index from exact parameters in *Oracle Database Rules Manager and Expression Filter Developer's Guide* and [XPINDEX\\_PARAMETERS Procedure](#) for more information.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS, USER\_EXPFIL\_DEF\_INDEX\_PARAMS and USER\_EXPFIL\_INDEX\_PARAMS.

## Examples

The following command synchronizes the expression set's index parameters with the defaults associated with the corresponding attribute set:

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab => 'consumer',
                              expr_col => 'interest',
                              attr_list => null,
                              operation => 'DEFAULT');

```

```
END;
/
```

The following command adds a stored attribute to the expression set's index parameters.

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab => 'consumer',
                              expr_col => 'interest',
                              attr_list =>
                                exf$attribute_list (
                                  exf$attribute (
                                    attr_name => 'CrashTestRating(Model, Year)',
                                    attr_oper => exf$indexoper('all'),
                                    attr_indexed => 'FALSE')),
                              operation => 'ADD');

```

```
END;
/
```

The following command clears the index parameters associated with the expression set:

```
BEGIN
  DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab => 'consumer',
                              expr_col => 'interest',
                              attr_list => null,
                              operation => 'CLEAR');

```

```
END;
/
```

A subsequent index creation will use the default index parameters assigned to the corresponding attribute set.

## MODIFY\_OPERATOR\_LIST Procedure

This procedure modifies the list of common operators associated with a certain attribute in the attribute set.

### Syntax

```
DBMS_EXPFIL.MODIFY_OPERATOR_LIST (
  attr_set   IN   VARCHAR2,
  attr_name  IN   VARCHAR2,
  attr_oper  IN   EXF$INDEXOPER);
```

### Parameters

**Table 65–16** *MODIFY\_OPERATOR\_LIST Procedure Parameters*

Parameter	Description
attr_set	Name of the attribute set
attr_name	Name of the stored or indexed attribute being modified
attr_oper	New list of operators that are frequently used in the predicates with the attribute

### Usage Notes

- The `MODIFY_OPERATOR_LIST` procedure modifies the operator list for the stored and indexed attributes defined in the default index parameters of the attribute set. Existing Expression Filter indexes are not affected when an attribute's operator list is modified. The updated index defaults are used when a new Expression Filter index is created or when an existing index is rebuilt.
- Related views: `USER_EXPFIL_DEF_INDEX_PARAMS`

### Examples

The following command modifies the operator list associated with the `HorsePower (Model, Year)` attribute defined in the `Car4Sale` attribute set.

```
BEGIN
  DBMS_EXPFIL.MODIFY_OPERATOR_LIST (
    attr_set => 'Car4Sale',
    attr_name => 'HorsePower (Model, Year)',
    attr_oper => exf$indexoper('=', '<', '>', 'between'));
END;
/
```

## REVOKE\_PRIVILEGE Procedure

This procedure revokes an expression privilege previously granted by the owner.

### Syntax

```
DBMS_EXPFIL.REVOKE_PRIVILEGE (
    expr_tab  IN  VARCHAR2,
    expr_col  IN  VARCHAR2,
    priv_type IN  VARCHAR2,
    from_user IN  VARCHAR2);
```

### Parameters

**Table 65–17** REVOKE\_PRIVILEGE Procedure Parameters

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions
priv_type	Type of privilege to be revoked
from_user	User from whom the privilege is to be revoked

### Usage Notes

- See [GRANT\\_PRIVILEGE Procedure](#) and the section on granting and revoking privileges in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about granting and revoking privileges.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS and USER\_EXPFIL\_PRIVILEGES.

### Examples

The following command revokes the INSERT EXPRESSION privilege on the interest column of the consumer table from user SCOTT:

```
BEGIN
    DBMS_EXPFIL.REVOKE_PRIVILEGE (expr_tab => 'consumer',
                                  expr_col => 'interest',
                                  priv_type => 'INSERT EXPRESSION',
                                  from_user => 'SCOTT');
END;
/
```

## SYNC\_TEXT\_INDEXES Procedure

This procedure synchronizes the indexes defined to process the predicates involving the CONTAINS operator in stored expressions.

### Syntax

```
DBMS_EXPFIL.SYNC_TEXT_INDEXES (
    expr_tab IN VARCHAR2);
```

### Parameters

**Table 65–18 SYNC\_TEXT\_INDEXES Procedure Parameters**

Parameter	Description
expr_tab	Name of table with expression columns containing text predicates

### Usage Notes

- When an expression filter index is defined on a column storing expressions, any text predicates in the expressions are indexed using a CTXRULE index. Unlike the other types of indexes (bitmap for scalar and XML predicates or spatial for spatial predicates) used to process the predicates in the expression set, the CTXRULE index defined to process the text predicates is not transactional in nature. That is, when the text predicates are updated with DML operations on the table storing expressions, the new predicates are not automatically reflected in the corresponding CTXRULE index. This could result in inconsistent results until the CTXRULE index is synchronized. This procedure can be used to synchronize all the CTXRULE indexes associated with a table with one or more expression columns. This procedure identifies all the CTXRULE indexes and invokes the CTX\_DDL.SYNC\_INDEX procedure on each of these indexes.

You must have the EXECUTE privilege on the CTX\_DDL package for successful synchronization of the text indexes.

### Examples

The following command synchronizes the text indexes associated with the expression columns in the consumer table:

```
BEGIN
    DBMS_EXPFIL.SYNC_TEXT_INDEXES (expr_tab => 'consumer');
END;
/
```

## UNASSIGN\_ATTRIBUTE\_SET Procedure

This procedure unassigns an attribute set from a column storing expressions.

### Syntax

```
DBMS_EXPFIL.UNASSIGN_ATTRIBUTE_SET (
  expr_tab  IN  VARCHAR2,
  expr_col  IN  VARCHAR2);
```

### Parameters

**Table 65–19 UNASSIGN\_ATTRIBUTE\_SET Procedure Parameters**

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions

### Usage Notes

- A column of an expression data type can be converted back to a VARCHAR2 type by unassigning the attribute set. You can unassign an attribute set from a column storing expressions if an Expression Filter index is not defined on the column.
- See [ASSIGN\\_ATTRIBUTE\\_SET Procedure](#) for information about assigning attribute sets.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS and USER\_EXPFIL\_INDEXES.

### Examples

The following command unassigns the attribute set previously assigned to the `interest` column of the `consumer` table. (See the section on bulk loading of expression data in *Oracle Database Rules Manager and Expression Filter Developer's Guide*.)

```
BEGIN
  DBMS_EXPFIL.UNASSIGN_ATTRIBUTE_SET (expr_tab => 'consumer',
                                     expr_col => 'interest');
END;
/
```



## VALIDATE\_EXPRESSIONS Procedure

This procedure validates all the expressions in a set.

### Syntax

```
DBMS_EXPFIL.VALIDATE_EXPRESSIONS (
  expr_tab      IN  VARCHAR2,
  expr_col      IN  VARCHAR2,
  exception_tab IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 65–20** VALIDATE\_EXPRESSIONS Procedure Parameters

Parameter	Description
expr_tab	Name of table storing the expression set
expr_col	Name of column in the table that stores the expressions
exception_tab	Name of the exception table. This table is created using the BUILD_EXCEPTIONS_TABLE procedure.

### Usage Notes

- The expressions stored in a table may have references to schema objects like user-defined functions and tables. When these schema objects are dropped or modified, the expressions could become invalid and the subsequent evaluation (query with EVALUATE operator) could fail.
- By default, the expression validation procedure fails on the first expression that is invalid. Optionally, the caller can pass an exception table to store references to all the invalid expressions. In addition to validating expressions in the set, this procedure validates the parameters (stored and indexed attributes) of the associated index and the approved list of user-defined functions. Any errors in the index parameters or the user-defined function list are immediately reported to the caller.
- See the section on evaluation semantics in *Oracle Database Rules Manager and Expression Filter Developer's Guide* and [BUILD\\_EXCEPTIONS\\_TABLE Procedure](#) for more information.
- Related views: USER\_EXPFIL\_EXPRESSION\_SETS, USER\_EXPFIL\_ASET\_FUNCTIONS, and USER\_EXPFIL\_PREDTAB\_ATTRIBUTES.

### Examples

The following command validates the expressions stored in the interest column of the consumer table.

```
BEGIN
  DBMS_EXPFIL.VALIDATE_EXPRESSIONS (expr_tab => 'consumer',
                                     expr_col => 'interest');
END;
/
```

## XPINDEX\_PARAMETERS Procedure

This procedure is used in conjunction with the `INDEX_PARAMETERS` procedure to fine-tune the XPath-specific index parameters for each expression set.

### Syntax

```
DBMS_EXPFIL.XPINDEX_PARAMETERS (
    expr_tab    IN    VARCHAR2,
    expr_col    IN    VARCHAR2,
    xmlt_attr   IN    VARCHAR2,
    xptag_list  IN    EXF$XPATH_TAGS,
    operation   IN    VARCHAR2 DEFAULT 'ADD');
```

### Parameters

**Table 65–21** *XPINDEX\_PARAMETERS Procedure Parameters*

Parameter	Description
<code>exp_tab</code>	Name of table storing the expression set
<code>expr_col</code>	Name of column in the table that stores the expressions
<code>xmlt_attr</code>	Name of the attribute with the XMLType datatype
<code>xptag_list</code>	An instance of EXF\$XPATH_TAGS type with a partial list of XML elements and attributes
<code>operation</code>	Operation to be performed on the list of index parameters. Default value: ADD. Valid values: ADD and DROP.

### Usage Notes

- When an attribute set is shared by multiple expression sets, the `INDEX_PARAMETERS` procedure can be used to tune the simple (non-XPath) index parameters for each expression set. The `XPINDEX_PARAMETERS` procedure is used in conjunction with the `INDEX_PARAMETERS` procedure to fine-tune the XPath-specific index parameters for each expression set.
- See also [INDEX\\_PARAMETERS Procedure](#) and the section on index tuning for XPath predicates in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information.
- Related views: `USER_EXPFIL_ATTRIBUTES`, `USER_EXPFIL_DEF_INDEX_PARAMS`, and `USER_EXPFIL_INDEX_PARAMS`.

---

**Note:** The values assigned to the `tag_name` argument of `exf$xpath_tag` type are case-sensitive.

---

### Examples

The following command synchronizes the index parameters of the expression set (XPath and non-XPath) with the defaults associated with the corresponding attribute set:

```
BEGIN
    DBMS_EXPFIL.INDEX_PARAMETERS(expr_tab => 'consumer',
                                expr_col => 'interest',
                                attr_list => null,
```

```
operation => 'DEFAULT');  
END;  
/
```

The following command adds an XPath-specific index parameter to the expression set:

```
BEGIN  
  DBMS_EXPFIL.XPINDEX_PARAMETERS(expr_tab => 'consumer',  
    expr_col => 'interest',  
    xslt_attr => 'details',  
    xptag_list =>  
      exf$xpath_tags(  
        exf$xpath_tag(tag_name => 'GPS',  
          tag_indexed => 'TRUE',  
          tag_type => NULL)),  
    operation => 'ADD');  
END;  
/
```



The DBMS\_FGA package provides fine-grained security functions.

This chapter contains the following topics:

- [Using DBMS\\_FGA](#)
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_FGA Subprograms](#)

## Using DBMS\_FGA

- [Security Model](#)
- [Operational Notes](#)

## Security Model

Execute privilege on DBMS\_FGA is needed for administering audit policies. Because the audit function can potentially capture all user environment and application context values, policy administration should be executable by privileged users only. The policy event handler module will be executed with the module owner's privilege.

## Operational Notes

This package is available for only cost-based optimization. The rule-based optimizer may generate unnecessary audit records since audit monitoring can occur before row filtering. For both the rule-based optimizer and the cost-based optimizer, you can refer to `DBA_FGA_AUDIT_TRAIL` to analyze the SQL text and corresponding bind variables that are issued.



---

## Summary of DBMS\_FGA Subprograms

**Table 66–1 DBMS\_FGA Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ADD_POLICY Procedure</a> on page 66-6	Creates an audit policy using the supplied predicate as the audit condition
<a href="#">DISABLE_POLICY Procedure</a> on page 66-11	Disables an audit policy
<a href="#">DROP_POLICY Procedure</a> on page 66-12	Drops an audit policy
<a href="#">ENABLE_POLICY Procedure</a> on page 66-13	Enables an audit policy

## ADD\_POLICY Procedure

This procedure creates an audit policy using the supplied predicate as the audit condition. The maximum number of FGA policies on any table or view object is 256.

### Syntax

```
DBMS_FGA.ADD_POLICY (
    object_schema    VARCHAR2,
    object_name      VARCHAR2,
    policy_name      VARCHAR2,
    audit_condition  VARCHAR2,
    audit_column     VARCHAR2,
    handler_schema   VARCHAR2,
    handler_module   VARCHAR2,
    enable           BOOLEAN,
    statement_types  VARCHAR2,
    audit_trail      BINARY_INTEGER IN DEFAULT,
    audit_column_opts BINARY_INTEGER IN DEFAULT);
```

### Parameters

**Table 66–2 ADD\_POLICY Procedure Parameters**

Parameter	Description	Default Value
object_schema	The schema of the object to be audited. (If NULL, the current log-on user schema is assumed.)	NULL
object_name	The name of the object to be audited.	-
policy_name	The unique name of the policy.	-
audit_condition	A condition in a row that indicates a monitoring condition. NULL is allowed and acts as TRUE.	NULL
audit_column	The columns to be checked for access. These can include OLS hidden columns or object type columns. The default, NULL, causes audit if any column is accessed or affected.	NULL
handler_schema	The schema that contains the event handler. The default, NULL, causes the current schema to be used.	NULL
handler_module	The function name of the event handler; includes the package name if necessary. This function is invoked only after the first row that matches the audit condition in the query is processed. If the procedure fails with an exception, the user SQL statement will fail as well.	NULL
enable	Enables the policy if TRUE, which is the default.	TRUE
statement_types	The SQL statement types to which this policy is applicable: INSERT, UPDATE, DELETE, or SELECT only.	SELECT
audit_trail	Destination (DB or XML) of fine grained audit records. Also specifies whether to populate LSQLTEXT and LSQLBIND in fga_log\$.	DB+EXTENDED
audit_column_opts	Establishes whether a statement is audited when the query references <i>any</i> column specified in the audit_column parameter or only when <i>all</i> such columns are referenced.	ANY_COLUMNS

### Usage Notes

- If `object_schema` is not specified, the current log-on user schema is assumed.

- An FGA policy should not be applied to out-of-line columns such as LOB columns.
- Each audit policy is applied to the query individually. However, at most one audit record may be generated for each policy, no matter how many rows being returned satisfy that policy's `audit_condition`. In other words, whenever any number of rows being returned satisfy an audit condition defined on the table, a single audit record will be generated for each such policy.
- If a table with an FGA policy defined on it receives a Fast Path insert or a vectored update, the hint is automatically disabled before any such operations. Disabling the hint allows auditing to occur according to the policy's terms. (One example of a Fast Path insert is the statement `INSERT-WITH-APPEND-hint`.)
- The `audit_condition` must be a boolean expression that can be evaluated using the values in the row being inserted, updated, or deleted. The expression can also use functions, such as the `USER` or `SYS_CONTEXT` functions.

The expression must not combine conditions using operators such as `AND` and `OR`. `audit_condition` can be `NULL` (or omitted), which is interpreted as `TRUE`, but it cannot contain the following elements:

- Subqueries or sequences
- The following attributes of the `USERENV` namespace when accessed using the `SYS_CONTEXT` function:
  - \* `CURRENT_SQL`
  - \* `CURRENT_SQL_LENGTH`
  - \* `CURRENT_BIND`
- Any use of the pseudo columns `LEVEL`, `PRIOR`, or `ROWNUM`.

Specifying an audit condition of "1=1" to force auditing of all specified statements ("`statement_types`") affecting the specified column ("`audit_column`") is no longer needed to achieve this purpose. A `NULL` value for `audit_condition` causes audit to happen even if no rows are processed, so that all actions on a table with this policy are audited.

- The `audit_condition` is evaluated using the privileges of the user who creates the policy.
- The audit function (`handler_module`) is an alerting mechanism for the administrator. The required interface for such a function is as follows:

```
PROCEDURE fname ( object_schema VARCHAR2, object_name VARCHAR2, policy_name
VARCHAR2 ) AS ...
```

where `fname` is the name of the procedure, `object_schema` is the name of the schema of the table audited, `object_name` is the name of the table to be audited, and `policy_name` is the name of the policy being enforced. The audit function will be executed with the function owner's privilege.

**See Also:** "Tutorial: Adding an E-Mail Alert to a Fine-Grained Audit Policy" in *Oracle Database Security Guide* for an example of creating an e-mail alert handler for a fine-grained audit policy

- The `audit_trail` parameter specifies both where the fine-grained audit trail will be written and whether it is to include the query's SQL Text and SQL Bind variable information (typically in columns named `LSQLTEXT` and `LSQLBIND`):

- If `audit_trail` includes XML, then fine-grained audit records are written to XML-format operating system files stored in the directory specified by an `AUDIT_FILE_DEST` statement in SQL. (The default `AUDIT_FILE_DEST` is `$ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump` on Unix-based systems, and `$ORACLE_BASE\admin\ $DB_UNIQUE_NAME\adump` on Windows systems.)
- If `audit_trail` includes DB instead, then the audit records are written to the `SYS.FGA_LOG$` table in the database. However, for read-only databases, Oracle Database writes the fine-grained audit records to XML files, regardless of the `audit_trail` settings.
- If `audit_trail` includes EXTENDED, then the query's SQL Text and SQL Bind variable information are included in the audit trail.

Note that the SQL Text information could include sensitive data, such as credit card numbers, in the audit trail. See "Auditing Sensitive Information" in *Oracle Database Security Guide* for ways in which you can handle this.

- For example:
  - \* Setting `audit_trail` to `DBMS_FGA.DB` sends the audit trail to the `SYS.FGA_LOG$` table in the database and omits SQL Text and SQL Bind.
  - \* Setting `audit_trail` to `DBMS_FGA.DB + DBMS_FGA.EXTENDED` sends the audit trail to the `SYS.FGA_LOG$` table in the database and includes SQL Text and SQL Bind.
  - \* Setting `audit_trail` to `DBMS_FGA.XML` writes the audit trail in XML files sent to the operating system and omits SQL Text and SQL Bind.
  - \* Setting `audit_trail` to `DBMS_FGA.XML + DBMS_FGA.EXTENDED` writes the audit trail in XML files sent to the operating system and includes SQL Text and SQL Bind.

The `audit_trail` parameter appears in the `ALL_AUDIT_POLICIES` view.

- You can change the operating system destination using the following command:
 

```
ALTER SYSTEM SET AUDIT_FILE_DEST = '<New Directory>' DEFERRED
```
- On many platforms, XML audit files are named `<process_name>_<processId>.xml`, for example, `ora_2111.xml`, or `s002_11.xml`. On Windows, the XML audit files are named `<process_name>_<ThreadId>.xml` (or `<process_name>_ProcessId.xml` if the process is not running as a thread).
- The `audit_column_opts` parameter establishes whether a statement is audited
  - when the query references *any* column specified in the `audit_column` parameter (`audit_column_opts = DBMS_FGA.ANY_COLUMNS`), or
  - only when *all* such columns are referenced (`audit_column_opts = DBMS_FGA.ALL_COLUMNS`).

The default is `DBMS_FGA.ANY_COLUMNS`.

The `ALL_AUDIT_POLICIES` view also shows `audit_column_opts`.

- When `audit_column_opts` is set to `DBMS_FGA.ALL_COLUMNS`, a SQL statement is audited only when all the columns mentioned in `audit_column` have been explicitly referenced in the statement. And these columns must be referenced in the same SQL-statement or in the sub-select.

Also, all these columns must refer to a single table/view or alias.

Thus, if a SQL statement selects the columns from different table aliases, the statement will not be audited.

### V\$XML\_AUDIT\_TRAIL View

The new values for the `audit_trail` parameter (`XML` and `XML+EXTENDED`) cause fine-grained auditing records to be written to operating system files in XML format.

Audit records stored in operating system files can be more secure than database-stored audit records because access can require file permissions that DBAs do not have. Operating system storage for audit records also offers higher availability, since such records remain available even if the database is temporarily inaccessible.

A new dynamic view, `V$XML_AUDIT_TRAIL`, makes such audit records from XML files available to DBAs through SQL query, providing enhanced usability. Querying this view causes all XML files (all files with an `.xml` extension) in the `AUDIT_FILE_DEST` directory to be parsed and presented in relational table format.

The `DBA_COMMON_AUDIT_TRAIL` view includes the contents of the `V$XML_AUDIT_TRAIL` dynamic view for standard and fine-grained audit records.

Since the audit XML files are stored in files with extension `.xml` on all platforms, the dynamic view presents audit information similarly on all platforms, using the following schema:

**Table 66–3 Elements in the V\$XML\_AUDIT\_TRAIL Dynamic View**

Element	Type
AUDIT_TYPE	VARCHAR2 (18)
SESSION_ID	NUMBER
PROXY_SESSIONID	NUMBER
STATEMENTID	NUMBER
ENTRYID	NUMBER
EXTENDED_TIMESTAMP	TIMESTAMP (6) WITH TIME ZONE
GLOBAL_UID	VARCHAR2 (32)
DB_USER	VARCHAR2 (30)
CLIENT_ID	VARCHAR2 (64)
EXT_NAME	VARCHAR2 (4000)
OS_USER	VARCHAR2 (255)
USERHOST	VARCHAR2 (128)
OS_PROCESS	VARCHAR2 (16)
TERMINAL	VARCHAR2 (255)
INSTANCE_NUMBER	NUMBER
OBJECT_SCHEMA	VARCHAR2 (30)
OBJECT_NAME	VARCHAR2 (128)
POLICY_NAME	VARCHAR2 (30)
STATEMENT_TYPE	VARCHAR2 (28)
TRANSACTIONID	RAW (8)
SCN	NUMBER

**Table 66–3 (Cont.) Elements in the V\$XML\_AUDIT\_TRAIL Dynamic View**

Element	Type
COMMENT_TEXT	VARCHAR2 (4000)
SQL_BIND	VARCHAR2 (4000)
SQL_TEXT	VARCHAR2 (4000)

## Usage Notes

- Every XML audit record contains the elements `AUDIT_TYPE` and `EXTENDED_TIMESTAMP`, with the latter printed in UTC zone (with no timezone information). Values retrieved using `V$XML_AUDIT_TRAIL` view are converted to session timezone and printed.
- For `SQL_TEXT` and `SQL_BIND` element values (CLOB type columns), the dynamic view shows only the first 4000 characters. The underlying XML file may have more than 4000 characters for such `SQL_TEXT` and `SQL_BIND` values.
- For large numbers of XML audit files, querying `V$XML_AUDIT_TRAIL` is faster when they are loaded into a database table using `SQL*Loader` or a similar tool. XML audit files are larger than the equivalent written to OS files when `AUDIT_TRAIL=OS`.
- Error handling is the same as when `AUDIT_TRAIL=OS`. If any error occurs in writing an audit record to disk, including the directory identified by `AUDIT_FILE_DEST` being full, the auditing operation fails. An alert message is logged.
- The policy event handler module will be executed with the module owner's privilege.

## Examples

```
DBMS_FGA.ADD_POLICY (
  object_schema => 'scott',
  object_name   => 'emp',
  policy_name   => 'mypolicy1',
  audit_condition => 'sal < 100',
  audit_column  => 'comm,sal',
  handler_schema => NULL,
  handler_module => NULL,
  enable        => TRUE,
  statement_types => 'INSERT, UPDATE',
  audit_trail   => DBMS_FGA.XML + DBMS_FGA.EXTENDED,
  audit_column_opts => DBMS_FGA.ANY_COLUMNS);
```

## DISABLE\_POLICY Procedure

This procedure disables an audit policy.

### Syntax

```
DBMS_FGA.DISABLE_POLICY (
  object_schema  VARCHAR2,
  object_name    VARCHAR2,
  policy_name    VARCHAR2 );
```

### Parameters

**Table 66–4** *DISABLE\_POLICY Procedure Parameters*

Parameter	Description
object_schema	The schema of the object to be audited. (If NULL, the current log-on user schema is assumed.)
object_name	The name of the object to be audited.
policy_name	The unique name of the policy.

The default value for `object_schema` is NULL. (If NULL, the current log-on user schema is assumed.)

### Examples

```
DBMS_FGA.DISABLE_POLICY (
  object_schema => 'scott',
  object_name   => 'emp',
  policy_name   => 'mypolicy1');
```

## DROP\_POLICY Procedure

This procedure drops an audit policy.

### Syntax

```
DBMS_FGA.DROP_POLICY (
  object_schema  VARCHAR2,
  object_name    VARCHAR2,
  policy_name    VARCHAR2 );
```

### Parameters

**Table 66–5** DROP\_POLICY Procedure Parameters

Parameter	Description
object_schema	The schema of the object to be audited. (If NULL, the current log-on user schema is assumed.)
object_name	The name of the object to be audited.
policy_name	The unique name of the policy.

### Usage Notes

The DBMS\_FGA procedures cause current DML transactions, if any, to commit before the operation unless they are inside a DDL event trigger. With DDL transactions, the DBMS\_FGA procedures are part of the DDL transaction. The default value for object\_schema is NULL. (If NULL, the current log-on user schema is assumed.)

---



---

**Note:** Oracle Database automatically drops the audit policy if you remove the object specified in the object\_name parameter of the DBMS\_FGA.ADD\_POLICY procedure, or if you drop the user who created the audit policy.

---



---

### Examples

```
DBMS_FGA.DROP_POLICY (
  object_schema => 'scott',
  object_name   => 'emp',
  policy_name   => 'mypolicy1');
```



## ENABLE\_POLICY Procedure

This procedure enables an audit policy.

### Syntax

```
DBMS_FGA.ENABLE_POLICY (
  object_schema  VARCHAR2,
  object_name    VARCHAR2,
  policy_name    VARCHAR2,
  enable         BOOLEAN);
```

### Parameters

**Table 66–6** *ENABLE\_POLICY Procedure Parameters*

Parameter	Description
object_schema	The schema of the object to be audited. (If <code>NULL</code> , the current log-on user schema is assumed.)
object_name	The name of the object to be audited.
policy_name	The unique name of the policy.
enable	Defaults to <code>TRUE</code> to enable the policy.

### Examples

```
DBMS_FGA.ENABLE_POLICY (
  object_schema => 'scott',
  object_name   => 'emp',
  policy_name   => 'mypolicy1',
  enable        => TRUE);
```



---

---

## DBMS\_FILE\_GROUP

The `DBMS_FILE_GROUP` package, one of a set of Oracle Streams packages, provides administrative interfaces for managing file groups, file group versions, and files. A file group repository is a collection of all of the file groups in a database and can contain multiple versions of a particular file group. You can use this package to create and manage file group repositories.

This chapter contains the following topics:

- [Using DBMS\\_FILE\\_GROUP](#)
  - Overview
  - Security Model
  - Constants
- [Summary of DBMS\\_FILE\\_GROUP Subprograms](#)

---

## Using DBMS\_FILE\_GROUP

This section contains topics which relate to using the DBMS\_FILE\_GROUP package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)

## Overview

The following terms pertain to the DBMS\_FILE\_GROUP package:

### File

A **file** is a reference to a file stored on hard disk. A file is composed of a file name, a directory object, and a file type. The directory object references the directory in which the file is stored on hard disk. For example, a file might have the following components:

- The file name is `expdat.dmp`.
- The directory object that contains the file is `db_files`.
- The file type is `DBMS_FILE_GROUP.EXPORT_DUMP_FILE`.

### Version

A **version** is a collection of related files. For example, a version might consist of a set of datafiles and a Data Pump export dump file generated by a Data Pump transportable tablespace export. Only one Data Pump export dump file is allowed in a version.

### File Group

A **file group** is a collection of versions. A file group can logically group a set of versions. For example, a file group named `financial_quarters` can keep track of quarterly financial data by logically grouping versions of files related to a tablespace set. The tablespaces containing the data can be exported at the end of each quarter and versioned under names such as `Q1FY04`, `Q2FY04`, and so on.

**See Also:** *Oracle Streams Concepts and Administration*

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

## Constants

The DBMS\_FILE\_GROUP package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS\_FILE\_GROUP.EXPORT\_DUMP\_FILE.

Table 67-1 lists the parameters and enumerated constants.

**Table 67-1 DBMS\_FILE\_GROUP Parameters with Enumerated Constants**

Parameter	Option	Type	Description
file_type	<ul style="list-style-type: none"> <li>■ DATAFILE</li> </ul>	VARCHAR2 (30)	DATAFILE is a datafile for a database. This constant can be specified as 'DATAFILE'.
new_file_type	<ul style="list-style-type: none"> <li>■ EXPORT_DUMP_FILE</li> <li>■ DATAPUMP_LOG_FILE</li> </ul>		<p>EXPORT_DUMP_FILE is a Data Pump export dump file. This constant can be specified as 'DUMPSET'.</p> <p>DATAPUMP_LOG_FILE is a Data Pump export log file. This constant can be specified as 'DATAPUMPLOG'.</p>
max_versions	<ul style="list-style-type: none"> <li>■ INFINITE</li> </ul>	NUMBER	INFINITE specifies no limit. The max_versions or retention_days can increase without reaching a limit.
retention_days			
privilege	<p>System privilege specified in the GRANT_SYSTEM_PRIVILEGE procedure:</p> <ul style="list-style-type: none"> <li>■ READ_ANY_FILE_GROUP</li> <li>■ MANAGE_ANY_FILE_GROUP</li> <li>■ MANAGE_FILE_GROUP</li> </ul> <p>Object privilege specified in the GRANT_OBJECT_PRIVILEGE procedure:</p> <ul style="list-style-type: none"> <li>■ READ_ON_FILE_GROUP</li> <li>■ MANAGE_ON_FILE_GROUP</li> </ul>	BINARY_INTEGER	<p>READ_ANY_FILE_GROUP grants the privilege to view information about any file group in any schema in the data dictionary.</p> <p>MANAGE_ANY_FILE_GROUP grants the privilege to create, manage, and drop any file group in any schema.</p> <p>MANAGE_FILE_GROUP grants the privilege to create, manage, and drop file groups in the user's schema.</p> <p>READ_ON_FILE_GROUP grants the privilege to view information about a specific file group in the data dictionary.</p> <p>MANAGE_ON_FILE_GROUP grants the privilege to manage a specific file group in a schema other than the user's schema.</p>

---

## Summary of DBMS\_FILE\_GROUP Subprograms

**Table 67–2 DBMS\_FILE\_GROUP Package Subprograms**

Subprogram	Description
<a href="#">ADD_FILE Procedure</a> on page 67-7	Adds a file to a version of a file group
<a href="#">ALTER_FILE Procedure</a> on page 67-9	Alters a file in a version of a file group
<a href="#">ALTER_FILE_GROUP Procedure</a> on page 67-11	Alters a file group
<a href="#">ALTER_VERSION Procedure</a> on page 67-13	Alters a version of a file group
<a href="#">CREATE_FILE_GROUP Procedure</a> on page 67-15	Creates a file group
<a href="#">CREATE_VERSION Procedure</a> on page 67-17	Creates a version of a file group
<a href="#">DROP_FILE_GROUP Procedure</a> on page 67-18	Drops a file group
<a href="#">DROP_VERSION Procedure</a> on page 67-19	Drops a version of a file group
<a href="#">GRANT_OBJECT_PRIVILEGE Procedure</a> on page 67-20	Grants object privileges on a file group to a user
<a href="#">GRANT_SYSTEM_PRIVILEGE Procedure</a> on page 67-21	Grants system privileges for file group operations to a user
<a href="#">PURGE_FILE_GROUP Procedure</a> on page 67-22	Purges a file group using the file group's retention policy
<a href="#">REMOVE_FILE Procedure</a> on page 67-23	Removes a file from a version of a file group
<a href="#">REVOKE_OBJECT_PRIVILEGE Procedure</a> on page 67-24	Revokes object privileges on a file group from a user
<a href="#">REVOKE_SYSTEM_PRIVILEGE Procedure</a> on page 67-25	Revokes system privileges for file group operations from a user

---

**Note:** All subprograms commit unless specified otherwise.

---



## ADD\_FILE Procedure

This procedure adds a file to a version of a file group.

### Syntax

```
DBMS_FILE_GROUP.ADD_FILE (
  file_group_name IN VARCHAR2,
  file_name       IN VARCHAR2,
  file_type       IN VARCHAR2 DEFAULT NULL,
  file_directory  IN VARCHAR2 DEFAULT NULL,
  version_name    IN VARCHAR2 DEFAULT NULL,
  comments        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 67-3 ADD\_FILE Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <i>hq_dba</i> and the file group name is <i>sales_tbs</i> , then specify <i>hq_dba.sales_tbs</i> . If the schema is not specified, then the current user is the default.
file_name	The name of the file being added to the version. Each file name in a version must be unique.
file_type	The file type. The following are reserved file types: <ul style="list-style-type: none"> <li>■ If the file is a datafile, then enter the following: 'DATAFILE'</li> <li>■ If the file is a Data Pump export dump file, then enter the following: 'DUMPSET' Data Pump metadata is populated when a Data Pump export dump file is imported.</li> <li>■ If the file is a Data Pump export log file, then enter the following: 'DATAPUMPLOG'</li> </ul> <p>If the file type is not one of the reserved file types, then either enter a text description of the file type, or specify <i>NULL</i> to omit a file type description.</p> <p>See "<a href="#">Constants</a>" on page 67-5 for more information about the reserved file types.</p>
file_directory	The name of the directory object that corresponds to the directory containing the file.  If <i>NULL</i> , then the procedure uses the default directory object for the version.  If <i>NULL</i> and no default directory object exists for the version, then the procedure uses the default directory object for the file group.  If <i>NULL</i> and no default directory object exists for the version or file group, then the procedure raises an error.

**Table 67–3 (Cont.) ADD\_FILE Procedure Parameters**

Parameter	Description
<code>version_name</code>	<p>The name of the version to which the file is added.</p> <p>If a positive integer is specified as a <code>VARCHAR2</code> value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file is added to version 1 of the file group.</p> <p>If <code>NULL</code>, then the procedure uses the version with the latest creation time for the file group.</p>
<code>comments</code>	Comments about the file being added

## Usage Notes

To run this procedure with either `DBMS_FILE_GROUP.EXPORT_DUMP_FILE` or 'DUMPSET' specified for the `file_type` parameter, a user must meet the following requirements:

- Have the appropriate privileges to import the Data Pump export dump file
- Have `READ` privilege on the directory object that contains the Data Pump export dump file

**See Also:** *Oracle Database Utilities* for more information about Data Pump privileges

## ALTER\_FILE Procedure

This procedure alters a file in a version of a file group.

### Syntax

```
DBMS_FILE_GROUP.ALTER_FILE(
  file_group_name    IN  VARCHAR2,
  file_name          IN  VARCHAR2,
  version_name       IN  VARCHAR2  DEFAULT NULL,
  new_file_name      IN  VARCHAR2  DEFAULT NULL,
  new_file_directory IN  VARCHAR2  DEFAULT NULL,
  new_file_type      IN  VARCHAR2  DEFAULT NULL,
  remove_file_type  IN  VARCHAR2  DEFAULT 'N',
  new_comments       IN  VARCHAR2  DEFAULT NULL,
  remove_comments   IN  VARCHAR2  DEFAULT 'N');
```

### Parameters

**Table 67-4 ALTER\_FILE Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <i>hq_dba</i> and the file group name is <i>sales_tbs</i> , then specify <i>hq_dba.sales_tbs</i> . If the schema is not specified, then the current user is the default.
file_name	The name of the file being altered in the version
version_name	The name of the version that contains the file being altered.  If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file in version 1 of the file group is altered.  If NULL, then the procedure uses the version with the latest creation time for the file group.
new_file_name	The new name of the file if the file name is being changed. Each file name in a version must be unique.  If NULL, then the procedure does not change the file name.  <b>Note:</b> When a non-NULL new file name is specified, this procedure changes the metadata for the file name in the data dictionary, but it does not change the file name on the hard disk.
new_file_directory	The new name of the directory object that corresponds to the directory containing the file, if the directory object is being changed.  If NULL, then the procedure does not change the directory object name.  <b>Note:</b> When a non-NULL new file directory is specified, this procedure changes the metadata for the file directory in the data dictionary, but it does not change the file directory on the hard disk.

**Table 67–4 (Cont.) ALTER\_FILE Procedure Parameters**

Parameter	Description
<code>new_file_type</code>	<p>The file type. The following are reserved file types:</p> <ul style="list-style-type: none"> <li>■ If the file is a datafile, then enter the following: 'DATAFILE'</li> <li>■ If the file is a Data Pump export dump file, then enter the following: 'DUMPSET'</li> <li>■ If the file is a Data Pump export log file, then enter the following: 'DATAPUMPLOG'</li> </ul> <p>If the file type is not one of the reserved file types, then enter a text description of the file type.</p> <p>If NULL, then the procedure does not change the file type.</p> <p><b>See Also:</b> "Constants" on page 67-5 for more information about the reserved file types.</p>
<code>remove_file_type</code>	<p>If Y, then the procedure removes the file type. If Y and the <code>new_file_type</code> parameter is non-NULL, then the procedure raises an error.</p> <p>If N, then the procedure does not remove the file type.</p>
<code>new_comments</code>	<p>New comments about the file being altered. If non-NULL, then the procedure replaces the existing comments with the specified comments.</p> <p>If NULL, then the procedure does not change the existing comments.</p>
<code>remove_comments</code>	<p>If Y, then the procedure removes the comments for the file. If Y and the <code>new_comments</code> parameter is non-NULL, then the procedure raises an error.</p> <p>If N, then the procedure does not change the existing comments.</p>

## Usage Notes

If the file type is changed to `DBMS_FILE_GROUP.EXPORT_DUMP_FILE` or 'DUMPSET', then Data Pump metadata for the file is populated. If the file type is changed from `DBMS_FILE_GROUP.EXPORT_DUMP_FILE` or 'DUMPSET', then Data Pump metadata for the file is purged.

To run this procedure with `DBMS_FILE_GROUP.EXPORT_DUMP_FILE` or 'DUMPSET' specified for the `new_file_type` parameter, a user must meet the following requirements:

- Have the appropriate privileges to import the Data Pump export dump file
- Have READ privilege on the directory object that contains the Data Pump export dump file

**See Also:** *Oracle Database Utilities* for more information about Data Pump privileges

## ALTER\_FILE\_GROUP Procedure

This procedure alters a file group.

### Syntax

```
DBMS_FILE_GROUP.ALTER_FILE_GROUP (
  file_group_name      IN  VARCHAR2,
  keep_files           IN  VARCHAR2  DEFAULT NULL,
  min_versions         IN  NUMBER    DEFAULT NULL,
  max_versions         IN  NUMBER    DEFAULT NULL,
  retention_days       IN  NUMBER    DEFAULT NULL,
  new_default_directory IN  VARCHAR2  DEFAULT NULL,
  remove_default_directory IN VARCHAR2  DEFAULT 'N',
  new_comments         IN  VARCHAR2  DEFAULT NULL,
  remove_comments     IN  VARCHAR2  DEFAULT 'N');
```

### Parameters

**Table 67-5 ALTER\_FILE\_GROUP Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group being altered, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <i>hq_dba</i> and the file group name is <i>sales_tbs</i> , then specify <i>hq_dba.sales_tbs</i> . If the schema is not specified, then the current user is the default.
keep_files	<p>If Y, then the files in the file group are retained on hard disk if the file group or a version of the file group is dropped or purged.</p> <p>If N, then the files in the file group are deleted from hard disk if the file group or a version of the file group is dropped or purged.</p> <p>If NULL, then this parameter is not changed.</p> <p><b>Note:</b> If the file group is dropped because of a DROP USER CASCADE statement, then the setting of this parameter determines whether the files are dropped from the hard disk.</p>
min_versions	<p>The minimum number of versions to retain. The specified value must be greater than or equal to 1.</p> <p>If NULL, then the procedure does not change the <i>min_versions</i> setting for the file group.</p>
max_versions	<p>The maximum number of versions to retain. The specified value must be greater than or equal to the value specified for <i>min_versions</i>. When the number of versions exceeds the specified <i>max_versions</i>, the oldest version is purged.</p> <p>Specify <i>DBMS_FILE_GROUP.INFINITE</i> for no limit to the number of versions.</p> <p>If NULL, then the procedure does not change the <i>max_versions</i> setting for the file group.</p>

**Table 67–5 (Cont.) ALTER\_FILE\_GROUP Procedure Parameters**

Parameter	Description
<code>retention_days</code>	<p>The maximum number of days to retain a version. The specified value must be greater than or equal to 0 (zero). When the age of a version exceeds the specified <code>retention_days</code> and there are more versions than the number specified in <code>min_versions</code>, the version is purged. The age of a version is calculated by subtracting the creation time from the current time.</p> <p>A decimal value can specify a fraction of a day. For example, 1.25 specifies one day and six hours.</p> <p>Specify <code>DBMS_FILE_GROUP.INFINITE</code> for no limit to the number of days a version can exist.</p> <p>If <code>NULL</code>, then the procedure does not change the <code>retention_days</code> setting for the file group.</p>
<code>new_default_directory</code>	<p>The default directory object used when files are added to a file group if no directory is specified when the files are added, and no default directory object is specified for the version.</p> <p>If <code>NULL</code>, then the procedure does not change the default directory.</p>
<code>remove_default_directory</code>	<p>If <code>Y</code>, then the procedure removes the default directory for the file group. If <code>Y</code> and the <code>new_default_directory</code> parameter is set to a non-<code>NULL</code> value, then the procedure raises an error.</p> <p>If <code>N</code>, then the procedure does not remove the default directory for the file group.</p>
<code>new_comments</code>	<p>Comments about the file group. If non-<code>NULL</code>, then the new comments replace the existing comments for the file group.</p> <p>If <code>NULL</code>, then the procedure does not change the existing comments.</p>
<code>remove_comments</code>	<p>If <code>Y</code>, then the comments for the file group are removed. If <code>Y</code> and the <code>new_comments</code> parameter is set to a non-<code>NULL</code> value, then the procedure raises an error.</p> <p>If <code>N</code>, then the procedure does not change the comments for the file group.</p>

## Usage Notes

If `min_versions` is set to 1, then the only version of the file group can be purged when a new version is added. If the addition of the new version is not complete when the existing version is purged, then there can be a period of time when no version of the file group is available. Therefore, set `min_versions` to at least 2 if a version of the file group must be available at all times.

## ALTER\_VERSION Procedure

This procedure alters a version of a file group.

### Syntax

```
DBMS_FILE_GROUP.ALTER_VERSION(
  file_group_name      IN  VARCHAR2,
  version_name         IN  VARCHAR2  DEFAULT NULL,
  new_version_name     IN  VARCHAR2  DEFAULT NULL,
  remove_version_name  IN  VARCHAR2  DEFAULT 'N',
  new_default_directory IN  VARCHAR2  DEFAULT NULL,
  remove_default_directory IN  VARCHAR2  DEFAULT 'N',
  new_comments         IN  VARCHAR2  DEFAULT NULL,
  remove_comments     IN  VARCHAR2  DEFAULT 'N');
```

### Parameters

**Table 67-6 ALTER\_VERSION Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <i>hq_dba</i> and the file group name is <i>sales_tbs</i> , then specify <i>hq_dba.sales_tbs</i> . If the schema is not specified, then the current user is the default.
version_name	The name of the version being altered.  If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then version 1 of the file group is altered.  If '*' is specified, then the procedure alters all versions, and the new_version_name parameter must be NULL.  If NULL, then the procedure uses the version with the latest creation time for the file group.
new_version_name	The new name of the version. Do not specify a schema.  The specified version name cannot be a positive integer or an asterisk ('*').  If NULL, then the procedure does not change the version name.
remove_version_name	If Y, then the procedure removes the version name. If the version name is removed, then the version number must be used to manage the version. If Y and the new_version_name parameter is set to a non-NULL value, then the procedure raises an error.  If N, then the procedure does not remove the version name.
new_default_directory	The default directory object used when files are added to a version if no directory is specified when the files are added.  If NULL, then the procedure does not change the default directory.
remove_default_directory	If Y, then the procedure removes the default directory. If Y and the new_default_directory parameter is set to a non-NULL value, then the procedure raises an error.  If N, then the procedure does not remove the default directory.

**Table 67-6 (Cont.) ALTER\_VERSION Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>new_comments</code>	Comments about the version. If non-NULL, then the new comments replace the existing comments for the version. If NULL, then the procedure does not change the comments.
<code>remove_comments</code>	If Y, then the procedure removes the comments for the version. If Y and the <code>new_comments</code> parameter is set to a non-NULL value, then the procedure raises an error. If N, then the procedure does not remove the comments for the version.

---



## CREATE\_FILE\_GROUP Procedure

This procedure creates a file group.

### Syntax

```
DBMS_FILE_GROUP.CREATE_FILE_GROUP (
  file_group_name    IN  VARCHAR2,
  keep_files         IN  VARCHAR2  DEFAULT 'Y',
  min_versions       IN  NUMBER     DEFAULT 2,
  max_versions       IN  NUMBER     DEFAULT DBMS_FILE_GROUP.INFINITE,
  retention_days     IN  NUMBER     DEFAULT DBMS_FILE_GROUP.INFINITE,
  default_directory IN  VARCHAR2  DEFAULT NULL,
  comments           IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 67–7 CREATE\_FILE\_GROUP Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <i>hq_dba</i> and the file group name is <i>sales_tbs</i> , then specify <i>hq_dba.sales_tbs</i> . If the schema is not specified, then the current user is the default.
keep_files	If <i>Y</i> , then the files in the file group are retained on hard disk if the file group or a version of the file group is dropped or purged.  If <i>N</i> , then the files in the file group are deleted from hard disk if the file group or a version of the file group is dropped or purged.  <b>Note:</b> If the file group is dropped because of a <i>DROP USER CASCADE</i> statement, then the setting of this parameter determines whether the files are dropped from the hard disk.
min_versions	The minimum number of versions to retain. The specified value must be greater than or equal to 1.
max_versions	The maximum number of versions to retain. The specified value must be greater than or equal to the value specified for <i>min_versions</i> . When the number of versions exceeds the specified <i>max_versions</i> , the oldest version is purged.  Specify <i>DBMS_FILE_GROUP.INFINITE</i> for no limit to the number of versions.
retention_days	The maximum number of days to retain a version. The specified value must be greater than or equal to 0 (zero). When the age of a version exceeds the specified <i>retention_days</i> and there are more versions than the number specified in <i>min_versions</i> , the version is purged. The age of a version is calculated by subtracting the creation time from the current time.  A decimal value can specify a fraction of a day. For example, <i>1.25</i> specifies one day and six hours.  Specify <i>DBMS_FILE_GROUP.INFINITE</i> for no limit to the number of days a version can exist.
default_directory	The default directory object used when files are added to a file group if no directory is specified when the files are added, and no default directory object is specified for the version.
comments	Comments about the file group being created.

**Usage Notes**

If `min_versions` is set to 1, then the only version of the file group can be purged when a new version is added. If the addition of the new version is not complete when the existing version is purged, then there can be a period of time when no version of the file group is available. Therefore, set `min_versions` to at least 2 if a version of the file group must be available at all times.

## CREATE\_VERSION Procedure

This procedure creates a version of a file group.

This procedure automatically runs the `PURGE_FILE_GROUP` procedure. Therefore, versions can be purged based on the file group's retention policy.

This procedure is overloaded. One version of the procedure contains the `OUT` parameter `version_out`, and the other does not.

**See Also:** [PURGE\\_FILE\\_GROUP Procedure](#) on page 67-22

### Syntax

```
DBMS_FILE_GROUP.CREATE_VERSION(
  file_group_name  IN  VARCHAR2,
  version_name     IN  VARCHAR2 DEFAULT NULL,
  default_directory IN  VARCHAR2 DEFAULT NULL,
  comments         IN  VARCHAR2 DEFAULT NULL);
```

```
DBMS_FILE_GROUP.CREATE_VERSION(
  file_group_name  IN  VARCHAR2,
  version_name     IN  VARCHAR2 DEFAULT NULL,
  default_directory IN  VARCHAR2 DEFAULT NULL,
  comments         IN  VARCHAR2 DEFAULT NULL,
  version_out      OUT VARCHAR2);
```

### Parameters

**Table 67–8 CREATE\_VERSION Procedure Parameters**

Parameter	Description
<code>file_group_name</code>	The name of the file group to which the new version is added, specified as <code>[schema_name.]file_group_name</code> . For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales_tbs</code> , then specify <code>hq_dba.sales_tbs</code> . If the schema is not specified, then the current user is the default.
<code>version_name</code>	The name of the version being created. Do not specify a schema. The specified version name cannot be a positive integer because, when a version is created, a version number is generated automatically. The specified version name cannot be an asterisk ( <code>'*' </code> ).
<code>default_directory</code>	The default directory object used when files are added to a version if no directory is specified when the files are added.
<code>comments</code>	Comments about the version being created
<code>version_out</code>	If the <code>version_name</code> parameter is set to a non-NULL value, then this parameter contains the specified version name. If the <code>version_name</code> parameter is set to NULL, then this parameter contains the generated version number.

## DROP\_FILE\_GROUP Procedure

This procedure drops a file group.

### Syntax

```
DBMS_FILE_GROUP.DROP_FILE_GROUP(  
    file_group_name IN VARCHAR2,  
    keep_files      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 67–9 DROP\_FILE\_GROUP Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group being dropped, specified as [ <i>schema_name</i> .] <i>file_group_name</i> . For example, if the schema is <i>hq_dba</i> and the file group name is <i>sales_tbs</i> , then specify <i>hq_dba.sales_tbs</i> . If the schema is not specified, then the current user is the default.
keep_files	If <i>Y</i> , then the procedure retains the files in the file group on hard disk. If <i>N</i> , then the procedure deletes the files in the file group from hard disk. If <i>NULL</i> , then the procedure uses the default keep files property of the file group.

### Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have `WRITE` privilege on the directory object that contains the files.

## DROP\_VERSION Procedure

This procedure drops a version of a file group.

### Syntax

```
DBMS_FILE_GROUP.DROP_VERSION(
  file_group_name IN VARCHAR2,
  version_name    IN VARCHAR2 DEFAULT NULL,
  keep_files      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 67–10 DROP\_VERSION Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales_tbs</code> , then specify <code>hq_dba.sales_tbs</code> . If the schema is not specified, then the current user is the default.
version_name	The name of the version being dropped.  If a positive integer is specified as a VARCHAR2 value, then the integer is interpreted as a version number. For example, if '1' is specified, then version 1 of the file group is dropped.  If NULL, then the procedure uses the version with the oldest creation time for the file group.  If '*', then the procedure drops all versions.
keep_files	If Y, then the procedure retains the files in the version on hard disk.  If N, then the procedure deletes the files in the version from hard disk.  If NULL, then the procedure uses the default keep files property of the file group.

### Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have WRITE privilege on the directory object that contains the files.

## GRANT\_OBJECT\_PRIVILEGE Procedure

This procedure grants object privileges on a file group to a user.

### Syntax

```
DBMS_FILE_GROUP.GRANT_OBJECT_PRIVILEGE(
  object_name  IN  VARCHAR2,
  privilege    IN  BINARY_INTEGER,
  grantee      IN  VARCHAR2,
  grant_option IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 67–11 GRANT\_OBJECT\_PRIVILEGE Procedure Parameters**

Parameter	Description
object_name	The name of the file group on which the privilege is granted, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales_tbs</code> , then specify <code>hq_dba.sales_tbs</code> . If the schema is not specified, then the current user is the default.
privilege	The constant that specifies the privilege. See "Constants" on page 67-5 for valid privileges.
grantee	The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.
grant_option	If <code>TRUE</code> , then the specified user granted the specified privilege can grant this privilege to others.  If <code>FALSE</code> , then the specified user granted the specified privilege cannot grant this privilege to others.

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

## GRANT\_SYSTEM\_PRIVILEGE Procedure

This procedure grants system privileges for file group operations to a user.

---

**Note:** When you grant a privilege on "ANY" object (for example, ALTER\_ANY\_RULE), and the initialization parameter O7\_DICTIONARY\_ACCESSIBILITY is set to FALSE, you give the user access to that type of object in all schemas, except the SYS schema. By default, the initialization parameter O7\_DICTIONARY\_ACCESSIBILITY is set to FALSE.

If you want to grant access to an object in the SYS schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the O7\_DICTIONARY\_ACCESSIBILITY initialization parameter to TRUE. Then privileges granted on "ANY" object allows access to any schema, including SYS. Set the O7\_DICTIONARY\_ACCESSIBILITY initialization parameter with caution.

---

### Syntax

```
DBMS_FILE_GROUP.GRANT_SYSTEM_PRIVILEGE(
  privilege      IN  BINARY_INTEGER,
  grantee        IN  VARCHAR2,
  grant_option   IN  BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 67–12 GRANT\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The constant that specifies the privilege. See " <a href="#">Constants</a> " on page 67-5 for valid privileges.
grantee	The name of the user or role for which the privilege is granted. The user who runs the procedure cannot be specified.
grant_option	If TRUE, then the specified user granted the specified privilege can grant this privilege to others. If FALSE, then the specified user granted the specified privilege cannot grant this privilege to others.

## PURGE\_FILE\_GROUP Procedure

This procedure purges a file group using the file group's retention policy.

A file group's retention policy is determined by its settings for the `max_versions`, `min_versions`, and `retention_days` parameters. The following versions of a file group are removed when a file group is purged:

- All versions greater than the `max_versions` setting for the file group when versions are ordered in descending order by creation time. Therefore, the older versions are purged before the newer versions.
- All versions older than the `retention_days` setting for the file group unless purging a version would cause the number of versions to drop below the `min_versions` setting for the file group.

A job named `SYS.FGR$AUTOPURGE_JOB` automatically purges all file groups in a database periodically according to the job's schedule. You can adjust this job's schedule using the `DBMS_SCHEDULER` package. Alternatively, you can create a job that runs the `PURGE_FILE_GROUP` procedure periodically.

### Syntax

```
DBMS_FILE_GROUP.PURGE_FILE_GROUP(
    file_group_name IN VARCHAR2);
```

### Parameter

**Table 67–13** *PURGE\_FILE\_GROUP Procedure Parameter*

Parameter	Description
<code>file_group_name</code>	The name of the file group, specified as <code>[schema_name.]file_group_name</code> . For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales_tbs</code> , then specify <code>hq_dba.sales_tbs</code> . If the schema is not specified, then the current user is the default.  If <code>NULL</code> and this procedure is run by <code>SYS</code> user, then the procedure purges all file groups.

### Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have `WRITE` privilege on the directory object that contains the files. Files are deleted when a version is purged and the `keep_files` parameter is set to `N` for the version's file group.



## REMOVE\_FILE Procedure

This procedure removes a file from a version of a file group.

### Syntax

```
DBMS_FILE_GROUP.REMOVE_FILE(
  file_group_name IN VARCHAR2,
  file_name       IN VARCHAR2,
  version_name    IN VARCHAR2 DEFAULT NULL,
  keep_file       IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 67–14 REMOVE\_FILE Procedure Parameters**

Parameter	Description
file_group_name	The name of the file group that contains the version, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales_tbs</code> , then specify <code>hq_dba.sales_tbs</code> . If the schema is not specified, then the current user is the default.
file_name	The name of the file being removed from the version
version_name	The name of the version from which the file is removed.  If a positive integer is specified as a <code>VARCHAR2</code> value, then the integer is interpreted as a version number. For example, if '1' is specified, then the file is removed from version 1 of the file group.  If <code>NULL</code> , then the procedure uses the version with the latest creation time for the file group.  If '*', then the procedure removes the file from all versions.
keep_file	If <code>Y</code> , then the procedure retains the file on hard disk.  If <code>N</code> , then the procedure deletes the file from hard disk.  If <code>NULL</code> , then the procedure uses the default keep files property of the file group.

### Usage Notes

If this procedure deletes files on hard disk, then the user who runs the procedure must have `WRITE` privilege on the directory object that contains the files.

## REVOKE\_OBJECT\_PRIVILEGE Procedure

This procedure revokes object privileges on a file group from a user.

### Syntax

```
DBMS_FILE_GROUP.REVOKE_OBJECT_PRIVILEGE(  
  object_name IN VARCHAR2,  
  privilege   IN BINARY_INTEGER,  
  revokee    IN VARCHAR2);
```

### Parameters

**Table 67–15 REVOKE\_OBJECT\_PRIVILEGE Procedure Parameters**

Parameter	Description
<code>object_name</code>	The name of the file group on which the privilege is revoked, specified as <i>[schema_name.]file_group_name</i> . For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales_tbs</code> , then specify <code>hq_dba.sales_tbs</code> . If the schema is not specified, then the current user is the default.
<code>privilege</code>	The constant that specifies the privilege. See " <a href="#">Constants</a> " on page 67-5 for valid privileges.
<code>revokee</code>	The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.

## REVOKE\_SYSTEM\_PRIVILEGE Procedure

This procedure revokes system privileges for file group operations from a user.

### Syntax

```
DBMS_FILE_GROUP.REVOKE_SYSTEM_PRIVILEGE(  
  privilege IN BINARY_INTEGER,  
  revokee   IN VARCHAR2);
```

### Parameters

**Table 67–16** *REVOKE\_SYSTEM\_PRIVILEGE Procedure Parameters*

Parameter	Description
privilege	The constant that specifies the privilege. See " <a href="#">Constants</a> " on page 67-5 for valid privileges.
revokee	The name of the user or role from which the privilege is revoked. The user who runs the procedure cannot be specified.



---

---

## DBMS\_FILE\_TRANSFER

The `DBMS_FILE_TRANSFER` package provides procedures to copy a binary file within a database or to transfer a binary file between databases.

**See Also:**

- *Oracle Database Administrator's Guide* for instructions about using file transfer
- *Oracle Streams Concepts and Administration* for applications of file transfer.

This chapter contains the following topic:

- [Using DBMS\\_FILE\\_TRANSFER](#)
  - Operating Notes
- [Summary of DBMS\\_FILE\\_TRANSFER Subprograms](#)

---

## Using DBMS\_FILE\_TRANSFER

- [Operating Notes](#)

## Operating Notes

---

---

**Caution:** DBMS\_FILE\_TRANSFER supports online backup. You should therefore be careful in copying or transferring a file that is being modified by the database because this can result in an inconsistent file, and require recovery. To guarantee consistency, bring files offline when the database is in use.

If you want to use DBMS\_FILE\_TRANSFER for performing backups, note that you are implementing self-managed backups, and should therefore put the files in hot backup mode.

---

---

## Summary of DBMS\_FILE\_TRANSFER Subprograms

**Table 68–1 DBMS\_FILE\_TRANSFER Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">COPY_FILE Procedure</a> on page 68-5	Reads a file from a source directory and creates a copy of it in a destination directory. The source and destination directories can both be in a local file system, or both be in an Automatic Storage Management (ASM) disk group, or between local file system and ASM with copying in either direction.
<a href="#">GET_FILE Procedure</a> on page 68-7	Contacts a remote database to read a remote file and then creates a copy of the file in the local file system or ASM
<a href="#">PUT_FILE Procedure</a> on page 68-9	Reads a local file or ASM and contacts a remote database to create a copy of the file in the remote file system



## COPY\_FILE Procedure

This procedure reads a file from a source directory and creates a copy of it in a destination directory. The source and destination directories can both be in a local file system, or both be in an Automatic Storage Management (ASM) disk group, or between local file system and ASM with copying in either direction.

You can copy any type of file to and from a local file system. However, you can copy only database files (such as datafiles, tempfiles, controlfiles, and so on) to and from an ASM disk group.

The destination file is not closed until the procedure completes successfully.

### Syntax

```
DBMS_FILE_TRANSFER.COPY_FILE (
  source_directory_object    IN  VARCHAR2,
  source_file_name          IN  VARCHAR2,
  destination_directory_object IN VARCHAR2,
  destination_file_name     IN  VARCHAR2);
```

### Parameters

**Table 68–2 COPY\_FILE Procedure Parameters**

Parameter	Description
source_directory_object	The directory object that designates the source directory. The directory object must already exist. (You create directory objects with the <code>CREATE DIRECTORY</code> command).
source_file_name	The name of the file to copy. This file must exist in the source directory.
destination_directory_object	The directory object that designates the destination directory. The directory object must already exist. If the destination is ASM, the directory object must designate either a disk group name (for example, <code>+diskgroup1</code> ) or a directory created for alias names. In the case of a directory, the full path to the directory must be specified (for example: <code>+diskgroup1/dbs/control</code> ).
destination_file_name	<p>The name to assign to the file in the destination directory. A file with the same name must not exist in the destination directory. If the destination is ASM:</p> <ul style="list-style-type: none"> <li>■ The file is given a fully qualified ASM filename and created in the appropriate directory (depending on the database name and file type)</li> <li>■ The file type tag assigned to the file is <code>COPY_FILE</code></li> <li>■ The value of the <code>destination_file_name</code> argument becomes the file's alias name in the designated destination directory</li> </ul> <p>The file name can be followed by an ASM template name in parentheses. The file is then given the attributes specified by the template.</p>

## Usage Notes

To run this procedure successfully, the current user must have the following privileges:

- READ privilege on the directory object specified in the `source_directory_object` parameter
- WRITE privilege on directory object specified in the `destination_directory_object` parameter

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. The copied file is treated as a binary file, and no character set conversion is performed. To monitor the progress of a long file copy, query the `V$SESSION_LONGOPS` dynamic performance view.

**See Also:** *Oracle Automatic Storage Management Administrator's Guide* for instructions about using file transfer

## Examples

```
SQL> create directory DGROUP as '+diskgroup1/dbs/backup';
```

Directory created.

```
SQL> BEGIN
  2   DBMS_FILE_TRANSFER.COPY_FILE('SOURCEDIR', 't_xdbtmp.f', 'DGROUP',
                                't_xdbtmp.f');
  3 END;
  4 /
```

PL/SQL procedure successfully completed.

```
SQL> EXIT
$ASMCMD
ASMCMD> ls
DISKGROUP1/
ASMCMD> cd diskgroup1/dbs/backup
ASMCMD> ls
t_xdbtmp.f => +DISKGROUP1/ORCL/TEMPFILE/COPY_FILE.267.546546525
```

## GET\_FILE Procedure

This procedure contacts a remote database to read a remote file and then creates a copy of the file in the local file system or ASM. The file that is copied is the source file, and the new file that results from the copy is the destination file. The destination file is not closed until the procedure completes successfully.

### Syntax

```
DBMS_FILE_TRANSFER.GET_FILE
  source_directory_object    IN VARCHAR2,
  source_file_name          IN VARCHAR2,
  source_database           IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  destination_file_name     IN VARCHAR2);
```

### Parameters

**Table 68–3** GET\_FILE Procedure Parameters

Parameter	Description
source_directory_object	The directory object from which the file is copied at the source site. This directory object must exist at the source site.
source_file_name	The name of the file that is copied in the remote file system. This file must exist in the remote file system in the directory associated with the source directory object.
source_database	The name of a database link to the remote database where the file is located.
destination_directory_object	The directory object into which the file is placed at the destination site. This directory object must exist in the local file system.
destination_file_name	The name of the file copied to the local file system. A file with the same name must not exist in the destination directory in the local file system.

### Usage Notes

To run this procedure successfully, the following users must have the following privileges:

- The connected user at the source database must have read privilege on the directory object specified in the `source_directory_object` parameter.
- The current user at the local database must have write privilege on the directory object specified in the `destination_directory_object` parameter.

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. The copied file is treated as a binary file, and no character set conversion is performed. To monitor the progress of a long file transfer, query the V\$SESSION\_LONGOPS dynamic performance view.

## Examples

```
CREATE OR REPLACE DIRECTORY df AS '+datafile' ;
GRANT WRITE ON DIRECTORY df TO "user";
CREATE DIRECTORY DSK_FILES AS '^t_work^';
GRANT WRITE ON DIRECTORY dsk_files TO "user";

-- assumes that dbs2 link has been created and we are connected to the instance.
-- dbs2 could be a loopback or point to another instance.

BEGIN
-- asm file to an os file
-- get an asm file from dbs1.asm/a1 to dbs2.^t_work^/oa5.dat
  DBMS_FILE_TRANSFER.GET_FILE ( 'df' , 'a1' , 'dbs1', 'dsk_files' , 'oa5.dat' );

-- os file to an os file
-- get an os file from dbs1.^t_work^/a2.dat to dbs2.^t_work^/a2back.dat
  DBMS_FILE_TRANSFER.GET_FILE ( 'dsk_files' , 'a2.dat' , 'dbs1', 'dsk_files' ,
'a2back.dat' );

END ;
/
```

## PUT\_FILE Procedure

This procedure reads a local file or ASM and contacts a remote database to create a copy of the file in the remote file system. The file that is copied is the source file, and the new file that results from the copy is the destination file. The destination file is not closed until the procedure completes successfully.

### Syntax

```
DBMS_FILE_TRANSFER.PUT_FILE(
  source_directory_object      IN  VARCHAR2,
  source_file_name            IN  VARCHAR2,
  destination_directory_object IN  VARCHAR2,
  destination_file_name       IN  VARCHAR2,
  destination_database        IN  VARCHAR2);
```

### Parameters

**Table 68–4** PUT\_FILE Procedure Parameters

Parameter	Description
source_directory_object	The directory object from which the file is copied at the local source site. This directory object must exist at the source site.
source_file_name	The name of the file that is copied from the local file system. This file must exist in the local file system in the directory associated with the source directory object.
destination_directory_object	The directory object into which the file is placed at the destination site. This directory object must exist in the remote file system.
destination_file_name	The name of the file placed in the remote file system. A file with the same name must not exist in the destination directory in the remote file system.
destination_database	The name of a database link to the remote database to which the file is copied.

### Usage Notes

To run this procedure successfully, the following users must have the following privileges:

- The current user at the local database must have read privilege on the directory object specified in the `source_directory_object` parameter.
- The connected user at the destination database must have write privilege to the directory object specified in the `destination_directory_object` parameter.

This procedure converts directory object parameters to uppercase unless they are surrounded by double quotation marks, but this procedure does not convert file names to uppercase.

Also, the copied file must meet the following requirements:

- The size of the copied file must be a multiple of 512 bytes.
- The size of the copied file must be less than or equal to two terabytes.

Transferring the file is not transactional. The copied file is treated as a binary file, and no character set conversion is performed. To monitor the progress of a long file transfer, query the V\$SESSION\_LONGOPS dynamic performance view.

## Examples

```
CREATE OR REPLACE DIRECTORY df AS '+datafile' ;
GRANT WRITE ON DIRECTORY df TO "user";
CREATE OR REPLACE DIRECTORY ft1 AS '+datafile/ft1' ;
GRANT READ,WRITE ON DIRECTORY ft1 TO "user";
CREATE OR REPLACE DIRECTORY ft1_1 AS '+datafile/ft1/ft1_1' ;

CONNECT user;
Enter password: password

-- - put a1.dat to a4.dat (using dbs2 dblink)
-- - level 2 sub dir to parent dir
-- - user has read privs on ft1_1 at dbs1 and write on df in dbs2
BEGIN
  DBMS_FILE_TRANSFER.PUT_FILE ( 'ft1_1' , 'a2.dat' , 'df' , 'a4.dat' ,
                                'dbs2' ) ;
END ;
```

---

---

## DBMS\_FLASHBACK

Using DBMS\_FLASHBACK, you can flash back to a version of the database at a specified wall-clock time or a specified system change number (SCN).

**See Also:** For detailed information about DBMS\_FLASHBACK:

- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database SQL Language Reference*.

This chapter contains the following topics:

- [Using DBMS\\_FLASHBACK](#)
  - Overview
  - Security Model
  - Types
  - Exceptions
  - Operational Notes
  - Examples
- [Summary of DBMS\\_FLASHBACK Subprograms](#)

## Using DBMS\_FLASHBACK

- [Overview](#)
- [Security Model](#)
- [Types](#)
- [Exceptions](#)
- [Operational Notes](#)
- [Examples](#)



## Overview

DBMS\_FLASHBACK provides an interface for the user to view the database at a particular time in the past, with the additional capacity provided by transaction backout features that allow for selective removal of the effects of individual transactions. This should not be confused with a flashback database which actually moves the database back in time.

When DBMS\_FLASHBACK is enabled, the user session uses the Flashback version of the database, and applications can execute against the Flashback version of the database.

You may want to use DBMS\_FLASHBACK for the following reasons:

- Self-service repair: If you accidentally delete rows from a table, you can recover the deleted rows.
- Packaged applications such as e-mail and voicemail: You can use Flashback to restore deleted e-mail by re-inserting the deleted message into the current message box.
- Decision support system (DSS) and online analytical processing (OLAP) applications: You can perform data analysis or data modeling to track seasonal demand.

## Security Model

To use this package, a database administrator must grant `EXECUTE` privileges for `DBMS_FLASHBACK`.

## Types

The following types are used by DBMS\_FLASHBACK subprograms:

**Table 69-1 DBMS\_FLASHBACK**

Type	Description
TXNAME_ARRAY	Creates a VARRAY for holding Transaction Names or Identifiers (XIDs)

## Exceptions

**Table 69–2** *DBMS\_FLASHBACK Error Messages*

<b>Error</b>	<b>Description</b>
ORA-08180	Time specified is too old
ORA-08181	Invalid system change number specified
ORA-08182	User cannot begin read-only or serializable transactions in Flashback mode
ORA-08183	User cannot enable Flashback within an uncommitted transaction
ORA-08184	User cannot enable Flashback within another Flashback session
ORA-08185	SYS cannot enable Flashback mode

## Operational Notes

DBMS\_FLASHBACK is automatically turned off when the session ends, either by disconnection or by starting another connection.

PL/SQL cursors opened in Flashback mode return rows as of the flashback time or SCN. Different concurrent sessions (connections) in the database can perform Flashback to different wall-clock times or SCNs. DML and DDL operations and distributed operations are not allowed while a session is running in Flashback mode. You can use PL/SQL cursors opened before disabling Flashback to perform DML.

Under Automatic Undo Management (AUM) mode, you can use retention control to control how far back in time to go for the version of the database you need.

If you need to perform a Flashback over a 24-hour period, the DBA should set the `undo_retention` parameter to 24 hours. This way, the system retains enough undo information to regenerate the older versions of the data.

You can set the `RETENTION GUARANTEE` clause for the undo tablespace to ensure that unexpired undo is not discarded. `UNDO_RETENTION` is not in itself a complete guarantee because, if the system is under space pressure, unexpired undo may be overwritten with freshly generated undo. In such cases, `RETENTION GUARANTEE` prevents this. For more information, see the *Oracle Database Administrator's Guide*

In a Flashback-enabled session, `SYSDATE` is not affected; it continues to provide the current time.

DBMS\_FLASHBACK can be used within logon triggers to enable Flashback without changing the application code.

## Examples

The following example illustrates how Flashback can be used when the deletion of a senior employee triggers the deletion of all the personnel reporting to him. Using the Flashback feature, you can recover and re-insert the missing employees.

```

DROP TABLE employee;
DROP TABLE keep_scn;

REM -- Keep_scn is a temporary table to store scns that we are interested in

CREATE TABLE keep_scn (scn number);
SET ECHO ON
CREATE TABLE employee (
    employee_no    number(5) PRIMARY KEY,
    employee_name  varchar2(20),
    employee_mgr   number(5)
        CONSTRAINT mgr_fkey REFERENCES EMPLOYEE ON DELETE CASCADE,
    salary         number,
    hiredate      date
);

REM -- Populate the company with employees
INSERT INTO employee VALUES (1, 'John Doe', null, 1000000, '5-jul-81');
INSERT INTO employee VALUES (10, 'Joe Johnson', 1, 500000, '12-aug-84');
INSERT INTO employee VALUES (20, 'Susie Tiger', 10, 250000, '13-dec-90');
INSERT INTO employee VALUES (100, 'Scott Tiger', 20, 200000, '3-feb-86');
INSERT INTO employee VALUES (200, 'Charles Smith', 100, 150000, '22-mar-88');
INSERT INTO employee VALUES (210, 'Jane Johnson', 100, 100000, '11-apr-87');
INSERT INTO employee VALUES (220, 'Nancy Doe', 100, 100000, '18-sep-93');
INSERT INTO employee VALUES (300, 'Gary Smith', 210, 75000, '4-nov-96');
INSERT INTO employee VALUES (310, 'Bob Smith', 210, 65000, '3-may-95');
COMMIT;

REM -- Show the entire org
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;

REM -- Sleep for a short time (approximately 10 to 20 seconds) to avoid
REM -- querying close to table creation

EXECUTE DBMS_LOCK.SLEEP(10);

REM -- Store this snapshot for later access through Flashback
DECLARE
I NUMBER;
BEGIN
I := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
INSERT INTO keep_scn VALUES (I);
COMMIT;
END;
/

REM -- Scott decides to retire but the transaction is done incorrectly
DELETE FROM EMPLOYEE WHERE employee_name = 'Scott Tiger';
COMMIT;

```

```

REM -- notice that all of scott's employees are gone
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM EMPLOYEE
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no = 1
ORDER BY LEVEL;

REM -- Flashback to see Scott's organization
DECLARE
    restore_scn number;
BEGIN
    SELECT scn INTO restore_scn FROM keep_scn;
    DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (restore_scn);
END;
/

REM -- Show Scott's org.
SELECT lpad(' ', 2*(level-1)) || employee_name Name
FROM employee
CONNECT BY PRIOR employee_no = employee_mgr
START WITH employee_no =
    (SELECT employee_no FROM employee WHERE employee_name = 'Scott Tiger')
ORDER BY LEVEL;

REM -- Restore scott's organization.
DECLARE
    scotts_emp NUMBER;
    scotts_mgr NUMBER;
    CURSOR c1 IS
        SELECT employee_no, employee_name, employee_mgr, salary, hiredate
        FROM employee
        CONNECT BY PRIOR employee_no = employee_mgr
        START WITH employee_no =
            (SELECT employee_no FROM employee WHERE employee_name = 'Scott Tiger');
    c1_rec c1 % ROWTYPE;
BEGIN
    SELECT employee_no, employee_mgr INTO scotts_emp, scotts_mgr FROM employee
    WHERE employee_name = 'Scott Tiger';
    /* Open c1 in flashback mode */
    OPEN c1;
    /* Disable Flashback */
    DBMS_FLASHBACK.DISABLE;
LOOP
    FETCH c1 INTO c1_rec;
    EXIT WHEN c1%NOTFOUND;
    /*
    Note that all the DML operations inside the loop are performed
    with Flashback disabled
    */
    IF (c1_rec.employee_mgr = scotts_emp) then
        INSERT INTO employee VALUES (c1_rec.employee_no,
            c1_rec.employee_name,
            scotts_mgr,
            c1_rec.salary,
            c1_rec.hiredate);
    ELSE
    IF (c1_rec.employee_no != scotts_emp) THEN
        INSERT INTO employee VALUES (c1_rec.employee_no,
            c1_rec.employee_name,

```

```
        cl_rec.employee_mgr,  
        cl_rec.salary,  
        cl_rec.hiredate);  
    END IF;  
END IF;  
END LOOP;  
END;  
/  
  
REM -- Show the restored organization.  
select lpad(' ', 2*(level-1)) || employee_name Name  
FROM employee  
CONNECT BY PRIOR employee_no = employee_mgr  
START WITH employee_no = 1  
ORDER BY LEVEL;
```



---

## Summary of DBMS\_FLASHBACK Subprograms

**Table 69–3 DBMS\_FLASHBACK Package Subprograms**

Subprogram	Description
<a href="#">DISABLE Procedure</a> on page 69-12	Disables the Flashback mode for the entire session
<a href="#">ENABLE_AT_SYSTEM_CHANGE_NUMBER Procedure</a> on page 69-13	Enables Flashback for the entire session. Takes an SCN as an Oracle number and sets the session snapshot to the specified number. Inside the Flashback mode, all queries return data consistent as of the specified wall-clock time or SCN
<a href="#">ENABLE_AT_TIME Procedure</a> on page 69-14	Enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in <code>query_time</code>
<a href="#">GET_SYSTEM_CHANGE_NUMBER Function</a> on page 69-15	Returns the current SCN as an Oracle number. You can use the SCN to store specific snapshots
<a href="#">TRANSACTION_BACKOUT Procedures</a> on page 69-16	Provides the mechanism to backout a transaction

## DISABLE Procedure

This procedure disables the Flashback mode for the entire session.

### Syntax

```
DBMS_FLASHBACK.DISABLE;
```

### Examples

The following example queries the salary of an employee, Joe, on August 30, 2000:

```
EXECUTE dbms_flashback.enable_at_time('30-AUG-2000');  
SELECT salary FROM emp where name = 'Joe'  
EXECUTE dbms_flashback.disable;
```

## ENABLE\_AT\_SYSTEM\_CHANGE\_NUMBER Procedure

This procedure takes an SCN as an input parameter and sets the session snapshot to the specified number. In the Flashback mode, all queries return data consistent as of the specified wall-clock time or SCN. It enables Flashback for the entire session.

### Syntax

```
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER (  
    query_scn IN NUMBER);
```

### Parameters

**Table 69–4** *ENABLE\_AT\_SYSTEM\_CHANGE\_NUMBER Procedure Parameters*

Parameter	Description
query_scn	The system change number (SCN), a version number for the database that is incremented on every transaction commit.

## ENABLE\_AT\_TIME Procedure

This procedure enables Flashback for the entire session. The snapshot time is set to the SCN that most closely matches the time specified in `query_time`. It enables Flashback for the entire session.

### Syntax

```
DBMS_FLASHBACK.ENABLE_AT_TIME (
    query_time    IN TIMESTAMP);
```

### Parameters

**Table 69–5** *ENABLE\_AT\_TIME Procedure Parameters*

Parameter	Description
<code>query_time</code>	<p>This is an input parameter of type <code>TIMESTAMP</code>. A time stamp can be specified in the following ways:</p> <ul style="list-style-type: none"> <li>■ Using the <code>TIMESTAMP</code> constructor           <pre>EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME (TIMESTAMP '2001-01-09 12:31:00');</pre> <p>Use the Globalization Support (NLS) format and supply a string. The format depends on the Globalization Support settings.</p> </li> <li>■ Using the <code>TO_TIMESTAMP</code> function:           <pre>EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME (TO_ TIMESTAMP ('12-02-2001 14:35:00', 'DD-MM-YYYY HH24:MI:SS'))</pre> <p>You provide the format you want to use. This example shows the <code>TO_TIMESTAMP</code> function for February 12, 2001, 2:35 PM.</p> </li> <li>■ If the time is omitted from query time, it defaults to the beginning of the day, that is, 12:00 A.M.</li> <li>■ Note that if the query time contains a time zone, the time zone information is truncated.</li> </ul>

## GET\_SYSTEM\_CHANGE\_NUMBER Function

This function returns the current SCN as an Oracle number datatype. You can obtain the current change number and store it for later use. This helps you retain specific snapshots.

### Syntax

```
DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER  
RETURN NUMBER;
```

## TRANSACTION\_BACKOUT Procedures

This procedure provides a mechanism to backout a set of transactions. The user can call these procedures with either transaction names or transaction identifiers (XIDS).

The procedure analyzes the transactional dependencies, perform DML's and generates an extensive report on the operation performed by the subprogram. This procedure does not commit the DML's performed as part of transaction backout. However it holds all the required locks on rows and tables in the right form, so that no other dependencies can enter the system. To make the changes permanent the user needs to explicitly commit the transaction.

A report is generated in the system tables DBA\_FLASHBACK\_TRANSACTION\_STATE and DBA\_FLASHBACK\_TRANSACTION\_REPORT.

### Syntax

```
DBMS_FLASHBACK.TRANSACTION_BACKOUT
  numtxns          NUMBER,
  xids              XID_ARRAY,
  options           NUMBER default NOCASCADE,
  timeHint         TIMESTAMP default MINTIME);
```

```
DBMS_FLASHBACK.TRANSACTION_BACKOUT
  numtxns          NUMBER,
  xids              XID_ARRAY,
  options           NUMBER default NOCASCADE,
  scnHint          TIMESTAMP default 0 );
```

```
DBMS_FLASHBACK.TRANSACTION_BACKOUT
  numtxns          NUMBER,
  txnnames         TXNAME_ARRAY,
  options           NUMBER default NOCASCADE,
  timehint         TIMESTAMP MINTIME );
```

```
DBMS_FLASHBACK.TRANSACTION_BACKOUT
  numtxns          NUMBER,
  txnNames         TXNAME_ARRAY,
  options           NUMBER default NOCASCADE,
  scnHint          NUMBER 0);
```

### Parameters

**Table 69–6 TRANSACTION\_BACKOUT Procedure Parameters**

Parameter	Description
numtxns	Number of transactions passed as input
xids	List of transaction IDs in the form of an array
txnnames	List of transaction names in the form of an array

**Table 69–6 (Cont.) TRANSACTION\_BACKOUT Procedure Parameters**

Parameter	Description
options	<p>Backout dependent transactions:</p> <ul style="list-style-type: none"> <li>■ NOCASCADE - The user expects no dependency. If a dependency is found, this raises an error, with the first dependent transaction provided in the report.</li> <li>■ NOCASCADE_FORCE - The user forcibly backs out the given transactions without considering the dependent transactions. The RDBMS executes the UNDO SQL for the given transactions in reverse order of their commit times. If no constraints break, and the result is satisfactory, the user can either COMMIT the changes or else ROLL BACK.</li> <li>■ NONCONFLICT_ONLY - This option lets the user backout the changes to the non-conflicting rows of the given transactions. Note that a transaction dependency happens due to a row conflict, whether either through WAW or primary/unique key constraints. If the user chooses to back out only the non-conflicting rows, this should not cause any problem with database consistency, although transaction atomicity is lost. As this is a recovery operation, the user can correct the data.</li> <li>■ CASCADE - This completely removes the given transactions including their dependents in a post order fashion (reverse order of commit times).</li> </ul>
timehint	Time hint on the start of the transaction
scnhint	SCN hint on the start of the transaction

## Usage Notes

---

**Note:** For information about restrictions in using TRANSACTION\_BACKOUT, see "Using Flashback Transaction" in the *Oracle Database Advanced Application Developer's Guide*.

---

- If transaction name is used, a time hint must be provided. The time hint should be a time before the start of all the given transactions to back out.
- If the SCN hint is provided, it must be before the start of the earliest transaction in the specified input set, or this raises an error and terminate. If it is not provided and the transaction has committed within undo retention, the database system is able to determine the start time.





---

---

## DBMS\_FLASHBACK\_ARCHIVE

The DBMS\_FLASHBACK\_ARCHIVE package contains two simple procedures for disassociation and reassociation of a Flashback Data Archive (FDA) enabled table from/with its underlying FDA respectively.

**See Also:** Oracle Database Advanced Application Developer's Guide

This chapter contains the following topics:

- [Using DBMS\\_FLASHBACK\\_ARCHIVE](#)
  - Overview
  - Security Model
  - Examples
- [Summary of DBMS\\_FLASHBACK\\_ARCHIVE Subprograms](#)

---

## Using DBMS\_FLASHBACK\_ARCHIVE

- [Overview](#)
- [Security Model](#)
- [Examples](#)

## Overview

The Flashback Data Archive feature, which is available through the Total Recall option, provides strict protection on the internal history tables that it creates and maintains for users.

The read-only semantics provided prohibits users, including a DBA, from doing updates, deletes, and inserts on the Flashback Data Archive internal history tables. The feature also prohibits users from issuing any DDL statements on these tables. This strict security enforcement by Flashback Data Archive meets the requirements of regulatory-compliance type of applications. The feature supports a lot of common DDL statements, including some DDL statements that alter table definition or incur data movement, on user tables that are enabled for Flashback Data Archive. However, there are some DDL statements that are not supported on tables enabled for the feature. Since most applications schemas evolve as they release new versions of their software, the ability to perform DDL operations on the base table is a key customer requirement.

In order to support user applications' schema evolution during application upgrade and other table maintenance tasks beyond DDL statements supported by Flashback Data Archive, the `DBMS_FLASHBACK_ARCHIVE` package provides a set of simple-to-use PL/SQL procedures:

- To disassociate a Flashback Data Archive enabled base table from the underlying FDA.
- To reassociate a temporarily disassociated base table with its underlying FDA.

After a user has disassociated the base table from its FDA, it's possible to issue any DDL statements on the base table or the history tables in the FDA. Having finished with the schema changes, the user can then reassociate the base table with its FDA so that Flashback Data Archive protection is in operation and automatic tracking and archiving is resumed.

## Security Model

Users with the `FLASHBACK ARCHIVE ADMINISTER` system privilege can execute the disassociation and reassociation PL/SQL procedures. Once a table is disassociated, normal users can perform DDL and DML statements as long as they have the necessary privileges on the tables.

## Examples

Normally, users cannot perform any modification to the history table:

```
SQL> DELETE FROM scott.SYS_FBA_HIST_61527;
```

```
ERROR at line 1:
```

```
ORA-55622: DML, ALTER and CREATE UNIQUE INDEX operations are not allowed on table
"SCOTT"."SYS_FBA_HIST_61527"
```

Users also cannot issue DDL statements on history tables:

```
SQL> ALTER TABLE scott.SYS_FBA_HIST_61527 DROP COLUMN comm;
```

```
ERROR at line 1:
```

```
ORA-55622: DML, ALTER and CREATE UNIQUE INDEX operations are not allowed on table
"SCOTT"."SYS_FBA_HIST_61527"
```

Use the [DISASSOCIATE\\_FBA Procedure](#) to disassociate the `scott.emp_test` table:

```
SQL> EXEC DBMS_FLASHBACK_ARCHIVE.DISASSOCIATE_FBA('scott','emp_test');
```

```
PL/SQL procedure successfully completed.
```

Now a user can perform table structural modifications (through DDL statements) to the user base table:

```
SQL> ALTER TABLE scott.emp_test RENAME COLUMN sal TO salary;
```

```
Table altered.
```

Users can also modify the contents in the history table that they couldn't modify previously.

```
SQL> DELETE FROM scott.SYS_FBA_HIST_61527 WHERE empno=3968;
```

```
2 rows deleted.
```

If a user tries to reassociate the history table with the base table, this will fail as the user has not performed corresponding structural modifications (through DDL statements) to the history table:

```
SQL> EXEC DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBA('scott','emp_test');
BEGIN DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBA('scott','emp_test');
END;
```

```
ERROR at line 1:
```

```
ORA-55636: Flashback Data Archive enabled table "SCOTT"."EMP_TEST" has different
definition from its history table
```

```
ORA-06512: at "SYS.DBMS_FLASHBACK_ARCHIVE", line 17
```

```
ORA-06512: at line 1
```

Disassociate the table and fix the table definition problem:

```
SQL> EXEC DBMS_FLASHBACK_ARCHIVE.DISASSOCIATE_FBA('scott','emp_test');
```

```
PL/SQL procedure successfully completed.
```

Perform the same rename column DDL on the history table to make its definition conform to its base table `scott.emp_test`:

```
SQL> ALTER TABLE scott.SYS_FBA_HIST_61527 RENAME COLUMN sal TO salary;
```

Table altered.

**Finally, reassociate the base table with its history table successfully:**

```
SQL> EXEC DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBA('scott','emp_test')
```

PL/SQL procedure successfully completed.

## Summary of DBMS\_FLASHBACK\_ARCHIVE Subprograms

**Table 70-1 DBMS\_FLASHBACK\_ARCHIVE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DISASSOCIATE_FBA Procedure</a> on page 70-8	Disassociates the given table from the flashback data archive
<a href="#">REASSOCIATE_FBA Procedure</a> on page 70-9	Reassociates the given table with the flashback data archive

## DISASSOCIATE\_FBA Procedure

This procedure disassociates the given table from the flashback data archive.

### Syntax

```
DBMS_FLASHBACK_ARCHIVE.DISASSOCIATE_FBA (  
    owner_name    VARCHAR2,  
    table_name    VARCHAR2);
```

### Parameters

**Table 70–2 DISASSOCIATE\_FBA Procedure Parameters**

Parameter	Description
owner_name	Schema of the Flashback Data Archive enabled base table
table_name	Name of the Flashback Data Archive enabled base table

### Exceptions

**Table 70–3 DISASSOCIATE\_FBA Procedure Exceptions**

Parameter	Description
ORA-55602	User table is not enabled for Flashback Data Archive
ORA-55634	Cannot acquire the lock on the table for disassociation



## REASSOCIATE\_FBA Procedure

This procedure reassociates the given table with the flashback data archive.

### Syntax

```
DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBA (
  owner_name      VARCHAR2,
  table_name      VARCHAR2);
```

### Parameters

**Table 70–4 REASSOCIATE\_FBA Procedure Parameters**

Parameter	Description
owner_name	Schema of the Flashback Data Archive enabled base table
table_name	Name of the Flashback Data Archive enabled base table

### Exceptions

**Table 70–5 REASSOCIATE\_FBA Procedure Exceptions**

Parameter	Description
ORA-55602	User table is not enabled for Flashback Data Archive
ORA-55636	table definition validation failed

### Usage Notes

- The procedure will signal an error if the base table and the history table do not have identical data definitions. For example when columns are added or table is split, the resulting base table and history table need to have the same schema.
- The FDA internal history table schema has some row versions metadata columns. The procedure will signal an error if any of the metadata columns is dropped by users.



---

---

## DBMS\_FREQUENT\_ITEMSET

The DBMS\_FREQUENT\_ITEMSET package enables frequent itemset counting. The two functions are identical except in the input cursor format difference.

This chapter contains the following topics:

- [Summary of DBMS\\_FREQUENT\\_ITEMSET Subprograms](#)

## Summary of DBMS\_FREQUENT\_ITEMSET Subprograms

**Table 71-1 DBMS\_FREQUENT\_ITEMSET Package Subprograms**

Subprogram	Description
<a href="#">FI_HORIZONTAL Function</a> on page 71-3	Counts all frequent itemsets given a cursor for input data which is in 'HORIZONTAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded
<a href="#">FI_TRANSACTIONAL Function</a> on page 71-5	Counts all frequent itemsets given a cursor for input data which is in 'TRANSACTIONAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded

## FI\_HORIZONTAL Function

The purpose of this table function is to count all frequent itemsets given a cursor for input data which is in 'HORIZONTAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded. The result will be a table of rows in form of itemset, support, length, total transactions counted.

In 'HORIZONTAL' row format, each row contains all of the item ids for a single transaction. Since all of the items come together, no transaction id is necessary.

The benefit of this table function is that if an application already has data in horizontal format, the database can skip the step of transforming rows that are in transactional format into horizontal format.

### Syntax

```
DBMS_FREQUENT_ITEMSET.FI_HORIZONTAL (
  tranx_cursor      IN      SYSREFCURSOR,
  support_threshold IN      NUMBER,
  itemset_length_min IN     NUMBER,
  itemset_length_max IN     NUMBER,
  including_items   IN      SYS_REFCURSOR DEFAULT NULL,
  excluding_items   IN      SYS_REFCURSOR DEFAULT NULL)
RETURN TABLE OF ROW (
  itemset [Nested Table of Item Type DERIVED FROM tranx_cursor],
  support      NUMBER,
  length       NUMBER,
  total_tranx  NUMBER);
```

### Parameters

**Table 71–2 FI\_HORIZONTAL Function Parameters**

Parameter	Description
tranx_cursor	The cursor parameter that the user will supply when calling the function. There is no limits on the number of returning columns. Each column of cursor represents an item. All columns of the cursor must be of the same data type. The item id must be number or character type (for example, VARCHAR2(n)).
support_threshold	A fraction number of total transaction count. An itemset is termed "frequent" if [the number of transactions it occurs in] divided by [the total number of transactions] exceed the fraction. The parameter must be a NUMBER.
itemset_length_min	The minimum length for interested frequent itemset. The parameter must be a NUMBER between 1 and 20, inclusive.
itemset_length_max	The maximum length for interested frequent itemset. This parameter must be a NUMBER between 1 and 20, inclusive, and must not be less than itemset_length_min.
including_items	A cursor from which a list of items can be fetched. At least one item from the list must appear in frequent itemsets that are returned. The default is NULL.
excluding_items	A cursor from which a list of items can be fetched. No item from the list can appear in frequent itemsets that are returned. The default is NULL.

## Return Values

**Table 71–3** *FI\_HORIZONTAL Return Values*

Parameter	Description
support	The number of transactions in which a frequent itemset occurs. This will be returned as a NUMBER.
itemset	A collection of items which is computed as frequent itemset. This will be returned as a nested table of item type which is the item column type of the input cursor.
length	Number of items in a frequent itemset. This will be returned as a NUMBER.
total_tranx	The total transaction count. This will be returned as a NUMBER.

## Example

Suppose you have a table `horiz_table_in`.

```
horiz_table_in(iid1 VARCHAR2(30), iid2 VARCHAR2(30), iid3 VARCHAR2(30), iid4
VARCHAR2(30), iid5 VARCHAR2(30));
```

and the data in `horiz_table_in` looks as follows:

```
('apple', 'banana', NULL, NULL, NULL)
('apple', 'milk', 'banana', NULL, NULL)
('orange', NULL, NULL, NULL, NULL)
```

Suppose you want to find out what combinations of items is frequent with a given support threshold of 30%, requiring itemset containing at least one of ('apple','banana','orange'), but excluding any of ('milk') in any itemset. You use the following query:

```
CREATE TYPE fi_varchar_nt AS TABLE OF VARCHAR2(30);
SELECT CAST(itemset as FI_VARCHAR_NT) itemset, support, length, total_tranx
FROM table(DBMS_FREQUENT_ITEMSET.FI_HORIZONTAL(
    CURSOR(SELECT iid1, iid2, iid3, iid4, iid5
            FROM horiz_table_in),
    0.3,
    2,
    5,
    CURSOR(SELECT * FROM table(FI_VARCHAR_NT
                              ('apple','banana','orange'))),
    CURSOR(SELECT * FROM table(FI_VARCHAR_NT('milk'))));
```

## FI\_TRANSACTIONAL Function

This procedure counts all frequent itemsets given a cursor for input data which is in 'TRANSACTIONAL' row format, support threshold, minimum itemset length, maximum itemset length, items to be included, items to be excluded. The result will be a table of rows in form of itemset, support, length, total number of transactions.

In 'TRANSACTIONAL' row format, each transaction is spread across multiple rows. All the rows of a given transaction have the same transaction id, and each row has a different item id. Combining all of the item ids which share a given transaction id results in a single transaction.

### Syntax

```
DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL (
  tranx_cursor      IN   SYSREFCURSOR,
  support_threshold IN   NUMBER,
  itemset_length_min IN  NUMBER,
  itemset_length_max IN  NUMBER,
  including_items   IN   SYS_REFCURSOR DEFAULT NULL,
  excluding_items   IN   SYS_REFCURSOR DEFAULT NULL)
RETURN TABLE OF ROW (
  itemset [Nested Table of Item Type DERIVED FROM tranx_cursor],
  support      NUMBER,
  length       NUMBER,
  total_tranx  NUMBER);
```

### Parameters

**Table 71-4 FI\_TRANSACTIONAL Function Parameters**

Parameter	Description
tranx_cursor	The cursor parameter that the user will supply when calling the function. It should return two columns in its returning row, the first column being the transaction id, the second column being the item id. The item id must be number or character type (for example, VARCHAR2(n)).
support_threshold	A fraction number of total transaction count. An itemset is termed "frequent" if [the number of transactions it occurs in] divided by [the total number of transactions] exceed the fraction. The parameter must be a NUMBER.
itemset_length_min	The minimum length for interested frequent itemset. The parameter must be a NUMBER between 1 and 20, inclusive.
itemset_length_max	The maximum length for interested frequent itemset. This parameter must be a NUMBER between 1 and 20, inclusive, and must not be less than itemset_length_min.
including_items	A cursor from which a list of items can be fetched. At least one item from the list must appear in frequent itemsets that will be returned. The default is NULL.
excluding_items	A cursor from which a list of items can be fetched. No item from the list can appear in frequent itemsets that will returned. The default is NULL.

## Return Values

**Table 71–5** *FI\_TRANSACTIONAL Return Values*

Parameter	Description
support	The number of transactions in which a frequent itemset occurs. This will be returned as a NUMBER.
itemset	A collection of items which is computed as frequent itemset. This will be returned as a nested table of item type which is the item column type of the input cursor.
length	Number of items in a frequent itemset. This will be returned as a NUMBER.
total_tranx	The total transaction count. This will be returned as a NUMBER, and will be the same for all returned rows, similar to a reporting aggregate.

## Usage Notes

Applications must predefine a nested table type of the input item type and cast the output itemset into this predefined nested table type before further processing, such as loading into a table.

## Examples

Suppose that the input table `tranx_table_in` looks as follows:

```
(1, 'apple')
(1, 'banana')
(2, 'apple')
(2, 'milk')
(2, 'banana')
(3, 'orange')
```

and the user is trying to find itemsets that satisfy a support-threshold of 60% and have the itemset-length greater than 1 (namely, (apple, banana)).

The output of this function would contain the following output row:

```
itemset=('apple','banana'), support=2, length=2, total_tranx=3
```

You need to create a nested table of item type before you submit a query to perform the frequent itemset counting. In this example, since item is of `VARCHAR2(30)`, you must create a nested table of `VARCHAR2(30)`:

```
CREATE TYPE fi_varchar_nt AS TABLE OF VARCHAR2(30);
SELECT CAST(itemset as FI_VARCHAR_NT) itemset, support, length, total_tranx
FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
        cursor(SELECT tid, iid FROM tranx_table_in),
        0.6,
        2,
        5,
        NULL,
        NULL));
```

Here is another example to illustrate how to include certain items and exclude certain items in the counting.

```
SELECT CAST(itemset as FI_VARCHAR_NT) itemset, support, length, total_tranx
FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
        CURSOR(SELECT tid, iid FROM tranx_table_in),
```



```

0.6,
2,
5,
CURSOR(SELECT * FROM table(FI_VARCHAR_NT
('apple', 'banana', 'orange'))),
CURSOR(SELECT * FROM table(FI_VARCHAR_NT('milk'))));

```

Using the including/excluding items parameter, you are able to further optimize the execution by ignoring itemsets that are not expected by application.

You can also use transactional output through collection unnesting:

```

SELECT
  bt.setid, nt.*
FROM
  (SELECT cast(Itemset as FI_VARCHAR_NT) itemset, rownum setid
   FROM table(
     DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
       CURSOR(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
       NULL, NULL)) bt,
   table(bt.itemset) nt);

```

If you want to use an insert statement to load frequent itemsets into a nested table, it is better to use the NESTED\_TABLE\_FAST\_INSERT hint for performance:

```

CREATE TABLE fq_nt (coll FI_VARCHAR_NT) NESTED TABLE coll STORE AS
  coll_nest;
INSERT /*+ NESTED_TABLE_FAST_INSERT */ INTO fq_nt
  SELECT cast(itemset as FI_VARCHAR_NT)
  FROM table(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
    cursor(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
    NULL, NULL));

```

Note that if you want to use the package inside a PL/SQL cursor, you must cast the return type of the table function:

```

CREATE TYPE fi_res AS OBJECT (
  itemset      FI_VARCHAR_NT,
  support      NUMBER,
  length       NUMBER,
  total_tranx  NUMBER
);
/
CREATE TYPE fi_coll AS TABLE OF fi_res;
/

DECLARE
  cursor freqC is
    SELECT Itemset
    FROM table(
      CAST(DBMS_FREQUENT_ITEMSET.FI_TRANSACTIONAL(
        cursor(SELECT tid, iid FROM tranx_table_in), 0.6, 2, 5,
        NULL, NULL) AS fi_coll));
  coll_nt  FI_VARCHAR_NT;
  num_rows int;
  num_itms int;
BEGIN
  num_rows := 0;
  num_itms := 0;
  OPEN freqC;
  LOOP

```

```
        FETCH freqC INTO coll_nt;
        EXIT WHEN freqC%NOTFOUND;
        num_rows := num_rows + 1;
        num_itms := num_itms + coll_nt.count;
    END LOOP;
    CLOSE freqC;
    DBMS_OUTPUT.PUT_LINE('Totally ' || num_rows || ' rows ' || num_itms || '
items were produced.');
```

```
END;
/
```

This package contains constants and procedure declarations for health check management. Health Monitor provides facilities to run a check store and retrieve the reports through DBMS\_HM package

**See Also:** *Oracle Database Administrator's Guide* for more information about "Health Monitor"

This chapter contains the following topics:

- [Using DBMS\\_HM](#)
  - Security Model
- [Summary of DBMS\\_HM Subprograms](#)

## Using DBMS\_HM

---

- [Security Model](#)

## Security Model

Users must have EXECUTE privilege to run the procedures of DBMS\_HM package.

---

## Summary of DBMS\_HM Subprograms

**Table 72-1 DBMS\_HM Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_RUN_REPORT Function</a> on page 72-5	Returns the report for the specified checker run
<a href="#">RUN_CHECK Procedure</a> on page 72-6	Runs the specified checker with the given arguments

## GET\_RUN\_REPORT Function

This function returns the report for the specified checker run.

### Syntax

```
DBMS_HM.GET_RUN_REPORT (  
    run_name      IN  VARCHAR2,  
    type          IN  VARCHAR2 := 'TEXT',  
    level         IN  VARCHAR2 := 'BASIC',)  
RETURN CLOB;
```

### Parameters

**Table 72–2** GET\_RUN\_REPORT Function Parameters

Parameter	Description
run_name	Name of the check's run
type	Report format type. Possible values are 'HTML', 'XML' and 'TEXT'. Default report type is 'TEXT'.
level	Details of report, possible value are 'BASIC' and 'DETAIL'. Caution: Currently only 'BASIC' level is supported.

## RUN\_CHECK Procedure

This procedure runs the specified checker with the given arguments. It lets user to specify a name for the run, inputs needed and maximum timeout for the run. The run's report will be maintained persistently in database.

### Syntax

```
DBMS_HM.RUN_CHECK (
    check_name      IN  VARCHAR2,
    run_name        IN  VARCHAR2 := NULL,
    timeout         IN  NUMBER := NULL,
    input_params    IN  VARCHAR2 := NULL);
```

### Parameters

**Table 72-3 RUN\_CHECK Procedure Parameters**

Parameter	Description
check_name	Name of the check to be invoked. Check names and their parameters can be accessed from the V\$HM_CHECK and V\$HM_CHECK_PARAM views. Users can run all checks which are not internal in nature: <code>SELECT name FROM V\$HM_CHECK WHERE INTERNAL_CHECK = 'N'</code> retrieves the list of checks that can be run manually by users.
run_name	Name with which external users can uniquely identify this check's run. If NULL value is passed, then HM creates a unique name and associates with this check's run.
timeout	Maximum amount of time (in units of seconds), this checker run is allowed to run. HM will interrupt the run, if it the specified time elapses for the run. If NULL value is passed, HM doesn't impose any timeout limits on the run.
input_params	<p>Input string: which consists of name, value pairs de-limited by a special character ';'.                      Example ('Data Block Integrity Check' invocation may take following type of input parameters.                      'BLC_DF_NUM=1;BLC_BL_NUM=23456'</p> <p>Input parameters BLC_DF_NUM and BLC_BL_NUM have values '1' and '23456' respectively.</p> <p>Every check will have well defined set of inputs associated with it. These Input parameters, their types, default values and descriptions can be obtained using V\$HM_CHECK_PARAM view.</p> <p>Example: The following query gets the list of parameters, their default values and descriptions for a 'Data Block Integrity Check'</p> <pre>SELECT a.* FROM v\$hm_check_param a, v\$hm_check b WHERE a.check_id = b.id AND b.name = 'Data Block Integrity Check';</pre>



The DBMS\_HPROF package provides an interface for profiling the execution of PL/SQL applications. It provides services for collecting the hierarchical profiler data, analyzing the raw profiler output and profiling information generation.

**See Also:** *Oracle Database Advanced Application Developer's Guide* for more information about the "PL/SQL Hierarchical Profiler"

This chapter contains the following topic:

- [Summary of DBMS\\_HPROF Subprograms](#)

## Summary of DBMS\_HPROF Subprograms

This table lists the package subprograms in alphabetical order.

**Table 73-1 DBMS\_HPROF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ANALYZE Function</a> on page 73-3	Analyzes the raw profiler output and produces hierarchical profiler information in database tables
<a href="#">START_PROFILING Procedure</a> on page 73-5	Starts hierarchical profiler data collection in the user's session
<a href="#">STOP_PROFILING Procedure</a> on page 73-6	Stops profiler data collection in the user's session.

## ANALYZE Function

This function analyzes the raw profiler output and produces hierarchical profiler information in database tables.

### Syntax

```
DBMS_HPROF.ANALYZE (
  location      VARCHAR2,
  filename      VARCHAR2,
  summary_mode  BOOLEAN      DEFAULT FALSE,
  trace         VARCHAR2    DEFAULT NULL,
  skip          PLS_INTEGER  DEFAULT 0,
  collect       PLS_INTEGER  DEFAULT NULL,
  run_comment   VARCHAR2    DEFAULT NULL)
RETURN NUMBER;
```

### Parameters

**Table 73–2 ANALYZE Function Parameters**

Parameter	Description
location	Name of a directory object. The raw profiler data file is read from the file system directory mapped to this directory object. Output files are also written to this directory.
filename	Name of the raw profiler data file to be analyzed. The file must exist in the directory specified by the <code>location</code> parameter.
summary_mode	By default (that is, when <code>summary_mode</code> is <code>FALSE</code> ), the detailed analysis is done. When <code>summary_mode</code> is <code>TRUE</code> , only top-level summary information is generated into the database table.
trace	Analyze only the subtrees rooted at the specified trace entry. By default (when <code>trace</code> is <code>NULL</code> ), the analysis/reporting is generated for the entire run. The trace entry must be specified in a special quoted qualified format (including the schema name, module name & function name) as in for example, <code>"SCOTT"."PKG"."FOO"</code> or <code>"".""."__plssql_vm"</code> . If multiple overloads exist for the specified name, all of them will be analyzed.
skip	Used only when <code>trace</code> is specified. Analyze only the subtrees rooted at the specified trace, but ignore the first <code>skip</code> invocations to trace. The default value for <code>skip</code> is 0.
collect	Used only when <code>trace</code> is specified. Analyze <code>collect</code> number of invocations of traces (starting from <code>skip+1</code> 'th invocation). By default only 1 invocation is collected.
run_comment	User-provided comment for this run

### Return Values

A unique run identifier for this run of the analyzer. This can then be used to look up the results corresponding to this run from the hierarchical profiler tables.

## Usage Notes

- Use the `dbmshptab.sql` script located in the `rdbms/admin` directory to create the hierarchical profiler database tables and other data structures required for persistently storing the results of analyzing the raw profiler data.
- Running `dbmshptab.sql` drops the any previously created hierarchical profiler tables.

## Examples

The following snippet installs the hierarchical profiler tables in HR schema.

```
connect HR/HR;  
@?/rdbms/admin/dbmshptab.sql
```

## START\_PROFILING Procedure

This procedure starts hierarchical profiler data collection in the user's session.

### Syntax

```
DBMS_HPROF.START_PROFILING (
  location    VARCHAR2 DEFAULT NULL,
  filename    VARCHAR2 DEFAULT NULL,
  max_depth   PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 73-3 START\_PROFILING Procedure Parameters**

Parameter	Description
location	Name of a directory object. The file system directory mapped to this directory object is where the raw profiler output is generated.
filename	Output filename for the raw profiler data. The file is created in the directory specified by the location parameter.
max_depth	By default (that is, when max_depth value is NULL) profile information is gathered for all functions irrespective of their call depth. When a non-NULL value is specified for max_depth, the profiler collects data only for functions up to a call depth level of max_depth.

### Usage Notes

Even though the profiler does not individually track functions at depth greater than max\_depth, the time spent in such functions is charged to the ancestor function at depth max\_depth.

## **STOP\_PROFILING Procedure**

This procedure stops profiler data collection in the user's session. This subprogram also has the side effect of flushing data collected so far in the session, and it signals the end of a run.

### **Syntax**

```
DBMS_HPROF.STOP_PROFILING;
```

---

---

## DBMS\_HS\_PARALLEL

The DBMS\_HS\_PARALLEL PL/SQL package enables parallel processing for heterogeneous targets access. This package is designed to improve performance when retrieving data from a large foreign table.

This chapter discusses the following topics:

- [Using DBMS\\_HS\\_PARALLEL](#)
- [Summary of DBMS\\_HS\\_PARALLEL Subprograms](#)

---

## Using DBMS\_HS\_PARALLEL

DBMS\_HS\_PARALLEL is compiled with the authorization ID of CURRENT\_USER, which uses invoker's rights. In other words, all procedures in this package are executed with the privileges of the calling user.



---

## Summary of DBMS\_HS\_PARALLEL Subprograms

**Table 74–1 DBMS\_HS\_PARALLEL Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CREATE_OR_REPLACE_VIEW</a> on page 74-4	Creates (or replaces) a read-only view to be referenced for retrieving the data from a remote table in parallel.
<a href="#">CREATE_TABLE_TEMPLATE</a> on page 74-6	Writes out a CREATE TABLE template based on information gathered from the remote table. You can use the information to add any optimal Oracle CREATE TABLE clauses.
<a href="#">DROP_VIEW</a> on page 74-7	Drops the view and internal objects created by the CREATE_OR_REPLACE_VIEW procedure. If the view has not already been created by the CREATE_OR_REPLACE_VIEW procedure, an error message is returned.
<a href="#">LOAD_TABLE</a> on page 74-8	Loads the data from a remote table to a local Oracle table in parallel. If the local Oracle table does not already exist, it is created automatically.

---

## CREATE\_OR\_REPLACE\_VIEW

This procedure creates (or replaces) a read-only view to be referenced for retrieving the data from a remote table in parallel.

### Syntax

```
CREATE_OR_REPLACE_VIEW (remote_table, database_link, oracle_view, parallel_degree)
```

### Parameters

**Table 74–2 CREATE\_OR\_REPLACE\_VIEW Parameter**

Parameter	Value	Description
remote_table	IN VARCHAR2 NOT NULL	The name of the remote database table. It is specified as [remote_schema_name.]remote_table_name.
database_link	IN VARCHAR2 NOT NULL	The remote database link name. The call can only be applied to a heterogeneous services database link.
oracle_view	IN VARCHAR2	The name of the Oracle view. It is specified as [schema_name.]oracle_view_name. The default schema name is the current user. If the oracle_view parameter is not specified, the remote table name will be used as the view name.
parallel_degree	IN NUMBER	The number of parallel processes for the operation is computed based on the range-partition number if applicable, or the number of CPUs. The range of values is 2 to 16.

### Usage Notes

- The specified Oracle view is created and future reference of this view utilizes internal database objects for parallel retrieval of remote non-Oracle table data. If the Oracle view already exists, the following Oracle error message is raised:
 

```
ORA-00955: name is already used by an existing object
```
- This view is created as a read-only view. If you attempt to insert and update the view, the following Oracle error message is raised:
 

```
ORA-01733: virtual column not allowed here
```
- If the remote table or the database link does not exist, one of the following Oracle error messages is raised:
 

```
ORA-00942: table or view does not exist
```

 or
 

```
ORA-02019: connection description for remote database not found
```
- You need the `CREATE VIEW`, `CREATE TABLE`, `CREATE TYPE`, `CREATE PACKAGE`, and `CREATE FUNCTION` privileges to execute the `CREATE_OR_REPLACE_VIEW` procedure.

- If you encounter either of the following Oracle error messages, increase the `PROCESSES` and `SESSIONS` parameter in the Oracle initialization parameter file:  
`ORA-12801: error signaled in parallel query server P003`  
or  
`ORA-00018: maximum number of session exceeded`
- Because the `CREATE_OR_REPLACE_VIEW` procedure creates some internal objects, use the `DROP_VIEW` procedure to drop the view and the internal objects. The SQL `DROP VIEW` statement only drops the view and not the internal objects.

## CREATE\_TABLE\_TEMPLATE

This procedure writes out a CREATE TABLE template based on information gathered from the remote table. You can use the information to add any optimal Oracle CREATE TABLE clauses.

### Syntax

```
CREATE_TABLE_TEMPLATE (remote_table, database_link, oracle_table, create_table_
template_string)
```

### Parameters

**Table 74–3** CREATE\_TABLE\_TEMPLATE Parameter

Parameter	Value	Description
remote_table	IN VARCHAR2 NOT NULL	The name of the remote database table. It is specified as [remote_schema_name.]remote_table_name.
database_link	IN VARCHAR2 NOT NULL	The remote database link name. The call can only be applied to a heterogeneous services database link.
oracle_table	IN VARCHAR2	The name of the local Oracle table the data will be loaded into. It is specified as [schema_name.]oracle_table_name. The default schema name is the current user. If the oracle_table parameter is not specified, the remote table name will be used as the local Oracle name.
create_table_template_string	OUT VARCHAR2	Contains the Oracle CREATE TABLE SQL template when the procedure is returned.

## DROP\_VIEW

This procedure drops the view and internal objects created by the CREATE\_OR\_REPLACE\_VIEW procedure. If the view has not already been created by the CREATE\_OR\_REPLACE\_VIEW procedure, an error message is returned.

### Syntax

```
DROP_VIEW (oracle_view)
```

### Parameters

**Table 74–4** DROP\_VIEW Parameter

Parameter	Value	Description
oracle_view	IN VARCHAR2 NOT NULL	The name of the Oracle view created by the CREATE_OR_REPLACE_VIEW procedure. If the view has not been created by the CREATE_OR_REPLACE_VIEW procedure, an error is returned.

## LOAD\_TABLE

This procedure loads the data from a remote table to a local Oracle table in parallel. If the local Oracle table does not already exist, it is created automatically.

### Syntax

```
LOAD_TABLE (remote_table, database_link, oracle_table, truncate, parallel_degree,
row_count)
```

### Parameters

**Table 74–5 LOAD\_TABLE Parameters**

Parameter	Value	Description
remote_table	IN VARCHAR2 NOT NULL	The name of the remote database table. It is specified as [remote_schema_name.]remote_table_name
database_link	IN VARCHAR2 NOT NULL	The remote database link name. The call can only be applied to a heterogeneous services database link.
oracle_table	IN VARCHAR2	The name of the local Oracle table the data will be loaded into. It is specified as [schema_name.]oracle_table_name. The default schema name is the current user. If the oracle_table parameter is not specified, the remote table name will be used as the local Oracle name.
truncate	IN BOOLEAN	Determines whether the Oracle table is truncated before the data is loaded. The value is either TRUE or FALSE. The default value is TRUE which means the Oracle table is truncated first. When set to FALSE, the Oracle table will not be truncated before the data is loaded.
parallel_degree	IN NUMBER	The number of parallel processes for the operation is computed based on the range-partition number if applicable, or the number of CPUs. The range of values is 2 to 16.
row_count	OUT NUMBER	Contains the number of rows just added with the load table operation.

### Usage Notes

- This procedure only loads the remote table data into Oracle local table. It does not create a key, index, constraints or any other dependencies such as triggers. It is recommended that you create these dependencies after the table data is loaded as performance will improve greatly. You will need to decide whether to create the dependencies before or after the data is loaded based on your knowledge of the remote table data and dependencies.

- If the local table does not exist, the `LOAD_TABLE` procedure creates a simple (non-partitioned) local table based on the exact column matching of the remote table after which the data is inserted into the local table.
- If the remote table or the database link does not exist, an error message is returned.
- If the local table is incompatible with the remote table, an error message is returned.
- You need the `CREATE TABLE`, `CREATE TYPE`, `CREATE PACKAGE`, and `CREATE FUNCTION` privileges to execute the `LOAD_TABLE` procedure.
- If you encounter either of the following Oracle error messages, increase the `PROCESSES` and `SESSIONS` parameter in Oracle initialization parameter file:  

```
ORA-12801: error signaled in parallel query server P003  
or  
ORA-00018: maximum number of session exceeded
```
- One of the following is required for parallel processing:
  - The remote table is range partitioned.
  - Histogram information for a numeric column is available.
  - There is a numeric index or primary key.
- To drop the local table, use the `DROP TABLE SQL` statement.





---

---

## DBMS\_HS\_PASSTHROUGH

The DBMS\_HS\_PASSTHROUGH PL/SQL package allows you to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows operations in statements for which there is no equivalent in Oracle.

This chapter discusses the following topics:

- [Using DBMS\\_HS\\_PASSTHROUGH](#)
  - Overview
  - Operational Notes
- [Summary of DBMS\\_HS\\_PASSTHROUGH Subprograms](#)

**See Also:** *Oracle Database Heterogeneous Connectivity User's Guide* for more information about this package

---

## Using DBMS\_HS\_PASSTHROUGH

This section contains topics which relate to using the DBMS\_HS\_PASSTHROUGH package.

## Overview

You can execute passthrough SQL statements directly at the non-Oracle system using the PL/SQL package DBMS\_HS\_PASSTHROUGH. Any statement executed with this package is executed in the same transaction as standard SQL statements.

**See Also:** *Oracle Database Heterogeneous Connectivity User's Guide* for information about this package

## Operational Notes

The `DBMS_HS_PASSTHROUGH` package is a virtual package. It conceptually resides at the non-Oracle system. In reality, however, calls to this package are intercepted by Heterogeneous Services and mapped to one or more Heterogeneous Services calls. The driver, in turn, maps these Heterogeneous Services calls to the API of the non-Oracle system. The client application should invoke the procedures in the package through a database link in exactly the same way as it would invoke a non-Oracle system stored procedure. The special processing done by Heterogeneous Services is transparent to the user.

---

## Summary of DBMS\_HS\_PASSTHROUGH Subprograms

**Table 75–1 DBMS\_HS\_PASSTHROUGH Package Subprograms**

Subprogram	Description
<a href="#">BIND_INOUT_VARIABLE Procedure</a> on page 75-6	Binds IN OUT bind variables
<a href="#">BIND_INOUT_VARIABLE_RAW Procedure</a> on page 75-7	Binds IN OUT bind variables of data type RAW
<a href="#">BIND_OUT_VARIABLE Procedure</a> on page 75-8	Binds an OUT variable with a PL/SQL program variable
<a href="#">BIND_OUT_VARIABLE_RAW Procedure</a> on page 75-9	Binds an OUT variable of data type RAW with a PL/SQL program variable
<a href="#">BIND_VARIABLE Procedure</a> on page 75-10	Binds an IN variable positionally with a PL/SQL program variable
<a href="#">BIND_VARIABLE_RAW Procedure</a> on page 75-11	Binds IN variables of type RAW
<a href="#">CLOSE_CURSOR Procedure</a> on page 75-12	Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system
<a href="#">EXECUTE_IMMEDIATE Procedure</a> on page 75-13	Runs a (non-SELECT) SQL statement immediately, without bind variables
<a href="#">EXECUTE_NON_QUERY Function</a> on page 75-14	Runs a (non-SELECT) SQL statement
<a href="#">FETCH_ROW Function</a> on page 75-15	Fetches rows from a query
<a href="#">GET_VALUE Procedure</a> on page 75-16	Retrieves column value from SELECT statement, or retrieves OUT bind parameters
<a href="#">GET_VALUE_RAW Procedure</a> on page 75-17	Similar to GET_VALUE, but for data type RAW
<a href="#">OPEN_CURSOR Function</a> on page 75-18	Opens a cursor for running a passthrough SQL statement at the non-Oracle system
<a href="#">PARSE Procedure</a> on page 75-19	Parses SQL statement at non-Oracle system

## BIND\_INOUT\_VARIABLE Procedure

This procedure binds IN OUT bind variables.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
  c      IN      BINARY_INTEGER NOT NULL,
  p      IN      BINARY_INTEGER NOT NULL,
  v      IN OUT  <dt>,
  n      IN      VARCHAR2);
```

<dt> is either DATE, NUMBER, or VARCHAR2.

**See Also:** For binding IN OUT variables of data type RAW see [BIND\\_INOUT\\_VARIABLE\\_RAW Procedure](#) on page 75-7.

### Parameters

**Table 75–2 BIND\_INOUT\_VARIABLE Procedure Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
n	(Optional) Name of the bind variable.  For example, in SELECT * FROM emp WHERE ename= :ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

### Exceptions

**Table 75–3 BIND\_INOUT\_VARIABLE Procedure Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS

## BIND\_INOUT\_VARIABLE\_RAW Procedure

This procedure binds IN OUT bind variables of data type RAW.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW (
  c      IN      BINARY_INTEGER NOT NULL,
  p      IN      BINARY_INTEGER NOT NULL,
  v      IN OUT  RAW,
  n      IN      VARCHAR2);
```

### Parameters

**Table 75-4 BIND\_INOUT\_VARIABLE\_RAW Procedure Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
n	(Optional) Name the bind variable.  For example, in SELECT * FROM emp WHERE ename= :ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

### Exceptions

**Table 75-5 BIND\_INOUT\_VARIABLE\_RAW Procedure Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS

## BIND\_OUT\_VARIABLE Procedure

This procedure binds an OUT variable with a PL/SQL program variable.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
  c      IN  BINARY_INTEGER NOT NULL,
  p      IN  BINARY_INTEGER NULL,
  v      OUT <dt>,
  n      IN  VARCHAR2);
```

<dt> is either DATE, NUMBER, or VARCHAR2.

**See Also:** For binding OUT variables of data type RAW, see [BIND\\_OUT\\_VARIABLE\\_RAW Procedure](#) on page 75-9.

### Parameters

**Table 75–6 BIND\_OUT\_VARIABLE Procedure Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename= :ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

### Exceptions

**Table 75–7 BIND\_OUT\_VARIABLE Procedure Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS



## BIND\_OUT\_VARIABLE\_RAW Procedure

This procedure binds an OUT variable of data type RAW with a PL/SQL program variable.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW (
  c      IN  BINARY_INTEGER NOT NULL,
  p      IN  BINARY_INTEGER NOT NULL,
  v      OUT RAW,
  n      IN  VARCHAR2);
```

### Parameters

**Table 75–8 BIND\_OUT\_VARIABLE\_RAW Procedure Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE_RAW.
n	(Optional) Name of the bind variable.  For example, in SELECT * FROM emp WHERE ename= :ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

### Exceptions

**Table 75–9 BIND\_OUT\_VARIABLE\_RAW Procedure Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS

## BIND\_VARIABLE Procedure

This procedure binds an IN variable positionally with a PL/SQL program variable.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (
  c      IN BINARY_INTEGER NOT NULL,
  p      IN BINARY_INTEGER NOT NULL,
  v      IN <dt>,
  n      IN VARCHAR2);
```

<dt> is either DATE, NUMBER, or VARCHAR2.

**See Also:** To bind RAW variables use [BIND\\_VARIABLE\\_RAW Procedure](#) on page 75-11.

### Parameters

**Table 75–10 BIND\_VARIABLE Procedure Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Value that must be passed to the bind variable name.
n	(Optional) Name of the bind variable.  For example, in <code>SELECT * FROM emp WHERE ename= :ename</code> , the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

### Exceptions

**Table 75–11 BIND\_VARIABLE Procedure Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined: WNDS, RNDS

## BIND\_VARIABLE\_RAW Procedure

This procedure binds IN variables of type RAW.

### Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (
  c    IN BINARY_INTEGER NOT NULL,
  p    IN BINARY_INTEGER NOT NULL,
  v    IN RAW,
  n    IN VARCHAR2);
```

### Parameters

**Table 75–12 BIND\_VARIABLE\_RAW Procedure Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	Value that must be passed to the bind variable.
n	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename= :ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

### Exceptions

**Table 75–13 BIND\_VARIABLE\_RAW Procedure Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS

## CLOSE\_CURSOR Procedure

This function closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system. If the cursor was not open, then the operation is a "no operation".

### Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (  
  c IN BINARY_INTEGER NOT NULL);
```

### Parameters

**Table 75–14** *CLOSE\_CURSOR Procedure Parameters*

Parameter	Description
c	Cursor to be released.

### Exceptions

**Table 75–15** *CLOSE\_CURSOR Procedure Exceptions*

Exception	Description
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS

## EXECUTE\_IMMEDIATE Procedure

This function runs a SQL statement immediately. Any valid SQL command except SELECT can be run immediately. The statement must not contain any bind variables. The statement is passed in as a VARCHAR2 in the argument. Internally the SQL statement is run using the PASSTHROUGH SQL protocol sequence of OPEN\_CURSOR, PARSE, EXECUTE\_NON\_QUERY, CLOSE\_CURSOR.

### Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
    s IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 75–16 EXECUTE\_IMMEDIATE Procedure Parameters**

Parameter	Description
s	VARCHAR2 variable with the statement to be executed immediately.

### Return Values

The number of rows affected by the execution of the SQL statement.

### Exceptions

**Table 75–17 EXECUTE\_IMMEDIATE Procedure Exceptions**

Exception	Description
ORA-28551	SQL statement is invalid.
ORA-28554	Max open cursors.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

## EXECUTE\_NON\_QUERY Function

This function runs a SQL statement. The SQL statement cannot be a `SELECT` statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be run.

### Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (  
    c IN BINARY_INTEGER NOT NULL)  
RETURN BINARY_INTEGER;
```

### Parameters

**Table 75–18 EXECUTE\_NON\_QUERY Function Parameters**

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines <code>OPEN_CURSOR</code> and <code>PARSE</code> respectively.

### Return Values

The number of rows affected by the SQL statement in the non-Oracle system

### Exceptions

**Table 75–19 EXECUTE\_NON\_QUERY Function Exceptions**

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	<code>BIND_VARIABLE</code> procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A <code>NULL</code> value was passed for a <code>NOT NULL</code> parameter.

## FETCH\_ROW Function

This function fetches rows from a result set. The result set is defined with a SQL `SELECT` statement. When there are no more rows to be fetched, the exception `NO_DATA_FOUND` is raised. Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed.

### Syntax

```
DBMS_HS_PASSTHROUGH.FETCH_ROW (
  c   IN BINARY_INTEGER NOT NULL,
  f   IN BOOLEAN)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 75–20** *FETCH\_ROW Function Parameters*

Parameter	Description
<code>c</code>	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines <code>OPEN_CURSOR</code> and <code>PARSE</code> respectively.
<code>first</code>	(Optional) Reexecutes <code>SELECT</code> statement. Possible values: <ul style="list-style-type: none"> <li>- <code>TRUE</code>: reexecute <code>SELECT</code> statement.</li> <li>- <code>FALSE</code>: fetch the next row, or if run for the first time, then execute and fetch rows (default).</li> </ul>

### Return Values

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

### Exceptions

**Table 75–21** *FETCH\_ROW Function Exceptions*

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A <code>NULL</code> value was passed for a <code>NOT NULL</code> parameter.

### Pragmas

Purity level defined : `WNDS`

## GET\_VALUE Procedure

This procedure has two purposes:

- It retrieves the select list items of `SELECT` statements, after a row has been fetched.
- It retrieves the `OUT` bind values, after the `SQL` statement has been run.

### Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE (
  c      IN  BINARY_INTEGER NOT NULL,
  p      IN  BINARY_INTEGER NOT NULL,
  v      OUT <dt>);
```

<dt> is either `DATE`, `NUMBER`, or `VARCHAR2`.

**See Also:** For retrieving values of data type `RAW`, see [GET\\_VALUE\\_RAW Procedure](#) on page 75-17.

### Parameters

**Table 75–22** GET\_VALUE Procedure Parameters

Parameter	Description
c	Cursor associated with the passthrough <code>SQL</code> statement. Cursor must be opened and parsed, using the routines <code>OPEN_CURSOR</code> and <code>PARSE</code> respectively.
p	Position of the bind variable or select list item in the <code>SQL</code> statement: Starts at 1.
v	Variable in which the <code>OUT</code> bind variable or select list item stores its value.

### Exceptions

**Table 75–23** GET\_VALUE Procedure Exceptions

Exception	Description
ORA-1403	Returns <code>NO_DATA_FOUND</code> exception when running the <code>GET_VALUE</code> after the last row was fetched (that is, <code>FETCH_ROW</code> returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the <code>SQL</code> statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A <code>NULL</code> value was passed for a <code>NOT NULL</code> parameter.

### Pragmas

Purity level defined : `WNDS`



## GET\_VALUE\_RAW Procedure

This procedure is similar to `GET_VALUE`, but for data type `RAW`.

### Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
  c    IN BINARY_INTEGER NOT NULL,
  p    IN BINARY_INTEGER NOT NULL,
  v    OUT RAW);
```

### Parameters

**Table 75–24** *GET\_VALUE\_RAW Procedure Parameters*

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines <code>OPEN_CURSOR</code> and <code>PARSE</code> respectively.
p	Position of the bind variable or select list item in the SQL statement: Starts at 1.
v	Variable in which the <code>OUT</code> bind variable or select list item stores its value.

### Exceptions

**Table 75–25** *GET\_VALUE\_RAW Procedure Exceptions*

Exception	Description
ORA-1403	Returns <code>NO_DATA_FOUND</code> exception when running the <code>GET_VALUE</code> after the last row was fetched (that is, <code>FETCH_ROW</code> returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A <code>NULL</code> value was passed for a <code>NOT NULL</code> parameter.

### Pragmas

Purity level defined : `WNDS`

## OPEN\_CURSOR Function

This function opens a cursor for running a passthrough SQL statement at the non-Oracle system. This function must be called for any type of SQL statement.

The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, call the procedure CLOSE\_CURSOR.

### Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR  
RETURN BINARY_INTEGER;
```

### Return Values

The cursor to be used on subsequent procedure and function calls.

### Exceptions

**Table 75–26 OPEN\_CURSOR Function Exceptions**

Exception	Description
ORA-28554	Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' OPEN_CURSORS initialization parameter.

### Pragmas

Purity level defined : WNDS, RNDS

## PARSE Procedure

This procedure parses SQL statement at non-Oracle system.

### Syntax

```
DBMS_HS_PASSTHROUGH.PARSE (
  c          IN  BINARY_INTEGER NOT NULL,
  stmt      IN  VARCHAR2 NOT NULL);
```

### Parameters

**Table 75–27** *PARSE Procedure Parameters*

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened using function OPEN_CURSOR.
stmt	Statement to be parsed.

### Exceptions

**Table 75–28** *PARSE Procedure Exceptions*

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28551	SQL statement is illegal.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

### Pragmas

Purity level defined : WNDS, RNDS



The `DBMS_IOT` package creates a table into which references to the chained rows for an index-organized table can be placed using the `ANALYZE` command. `DBMS_IOT` can also create an exception table into which references to the rows of an index-organized table that violate a constraint can be placed during the `enable_constraint` operation.

`DBMS_IOT` is not loaded during database installation. To install `DBMS_IOT`, run `dbmsiotc.sql`, available in the `ADMIN` directory.

This chapter contains the following topics:

- [Summary of DBMS\\_IOT Subprograms](#)

---

---

**Note:** With the introduction of logical-rowids for IOTs with Oracle Database Release 8.1, you no longer need to use the procedures contained in this package which is retained for backward compatibility only. It is however required for servers running with Oracle Database Release 8.0.

---

---

## Summary of DBMS\_IOT Subprograms

**Table 76–1** *DBMS\_IOT Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">BUILD_CHAIN_ROWS_TABLE Procedure</a> on page 76-3	Creates a table into which references to the chained rows for an index-organized table can be placed using the <code>ANALYZE</code> command
<a href="#">BUILD_EXCEPTIONS_TABLE Procedure</a> on page 76-4	Creates an exception table into which rows of an index-organized table that violate a constraint can be placed

## BUILD\_CHAIN\_ROWS\_TABLE Procedure

This procedure creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command.

### Syntax

```
DBMS_IOT.BUILD_CHAIN_ROWS_TABLE (
  owner          IN VARCHAR2,
  iot_name       IN VARCHAR2,
  chainrow_table_name IN VARCHAR2 default 'IOT_CHAINED_ROWS');
```

### Parameters

**Table 76-2** BUILD\_CHAIN\_ROWS\_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
chainrow_table_name	Intended name for the chained-rows table.

### Usage Notes

You should create a separate chained-rows table for each index-organized table to accommodate its primary key.

### Examples

```
CREATE TABLE l(a char(16),b char(16), c char(16), d char(240),
PRIMARY KEY(a,b,c)) ORGANIZATION INDEX pctthreshold 10 overflow;
EXECUTE DBMS_IOT.BUILD_CHAIN_ROWS_TABLE('SYS', 'L', 'LC');
```

A chained-row table is created with the following columns:

Column Name	Null?	Type
OWNER_NAME		VARCHAR2(30)
TABLE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
PARTITION_NAME		VARCHAR2(30)
SUBPARTITION_NAME		VARCHAR2(30)
HEAD_ROWID		ROWID
TIMESTAMP		DATE
A		CHAR(16)
B		CHAR(16)
C		CHAR(16)

## BUILD\_EXCEPTIONS\_TABLE Procedure

This procedure creates an exception table into which rows of an index-organized table that violate a constraint can be placed during the execution of the following SQL statements:

- ALTER TABLE ... ENABLE CONSTRAINT ... EXCEPTIONS INTO
- ALTER TABLE ... ADD CONSTRAINT ... EXCEPTIONS INTO

### Syntax

```
DBMS_IOT.BUILD_EXCEPTIONS_TABLE (
  owner          IN VARCHAR2,
  iot_name       IN VARCHAR2,
  exceptions_table_name IN VARCHAR2 default 'IOT_EXCEPTIONS');
```

### Parameters

**Table 76-3** BUILD\_EXCEPTIONS\_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
exceptions_table_name	Intended name for exception-table.

### Usage Notes

You should create a separate exception table for each index-organized table to accommodate its primary key.

### Examples

```
EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE('SYS', 'L', 'LE');
```

An exception table for the preceding index-organized table with the following columns:

Column Name	Null?	Type
ROW_ID		VARCHAR2(30)
OWNER		VARCHAR2(30)
TABLE_NAME		VARCHAR2(30)
CONSTRAINT		VARCHAR2(30)
A		CHAR(16)
B		CHAR(16)
C		CHAR(16)



The DBMS\_JAVA package provides a PL/SQL interface for accessing database functionality from Java.

- [Documentation of DBMS\\_JAVA](#)

---

## Documentation of DBMS\_JAVA

For a complete description of this package within the context of DBMS\_JAVA, see DBMS\_JAVA in the *Oracle Database Java Developer's Guide*.

The DBMS\_JOB package schedules and manages jobs in the job queue.

---

---

**Note:** The DBMS\_JOB package has been superseded by the DBMS\_SCHEDULER package. In particular, if you are administering jobs to manage system load, you should consider disabling DBMS\_JOB by revoking the package execution privilege for users.

For more information, see [Chapter 128, "DBMS\\_SCHEDULER"](#) and "Moving from DBMS\_JOB to DBMS\_SCHEDULER" in *Oracle Database Administrator's Guide*.

---

---

This chapter contains the following topics:

- [Using DBMS\\_JOB](#)
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_JOB Subprograms](#)

## Using DBMS\_JOB

---

- [Security Model](#)
- [Operational Notes](#)

## Security Model

No specific system privileges are required to use DBMS\_JOB. No system privileges are available to manage DBMS\_JOB. Jobs cannot be altered or deleted other than jobs owned by the user. This is true for all users including those users granted DBA privileges.

You can execute procedures that are owned by the user or for which the user is explicitly granted EXECUTE. However, procedures for which the user is granted the execute privilege through roles cannot be executed.

Note that, once a job is started and running, there is no easy way to stop the job.

## Operational Notes

- [Working with Oracle Real Application Clusters](#)
- [Stopping a Job](#)

### Working with Oracle Real Application Clusters

DBMS\_JOB supports multi-instance execution of jobs. By default jobs can be executed on any instance, but only one single instance will execute the job. In addition, you can force instance binding by binding the job to a particular instance. You implement instance binding by specifying an instance number to the instance affinity parameter. Note, however, that in Oracle Database 10g Release 1 (10.1) instance binding is not recommended. Service affinity is preferred. This concept is implemented in the [DBMS\\_SCHEDULER](#) package.

The following procedures can be used to create, alter or run jobs with instance affinity. Note that not specifying affinity means any instance can run the job.

#### DBMS\_JOB.SUBMIT

To submit a job to the job queue, use the following syntax:

```
DBMS_JOB.SUBMIT(  
  job      OUT      BINARY_INTEGER,  
  what     IN       VARCHAR2, NEXT_DATE IN DATE DEFAULTSYSDATE,  
  interval IN       VARCHAR2 DEFAULT 'NULL',  
  no_parse IN       BOOLEAN DEFAULT FALSE,  
  instance IN       BINARY_INTEGER DEFAULT ANY_INSTANCE,  
  force    IN       BOOLEAN DEFAULT FALSE);
```

Use the parameters `instance` and `force` to control job and instance affinity. The default value of `instance` is 0 (zero) to indicate that any instance can execute the job. To run the job on a certain instance, specify the `instance` value. Oracle displays error ORA-23319 if the `instance` value is a negative number or NULL.

The `force` parameter defaults to `false`. If `force` is TRUE, any positive integer is acceptable as the job instance. If `force` is FALSE, the specified instance must be running, or Oracle displays error number ORA-23428.

#### DBMS\_JOB.INSTANCE

To assign a particular instance to execute a job, use the following syntax:

```
DBMS_JOB.INSTANCE( JOB IN BINARY_INTEGER,  
  instance          IN BINARY_INTEGER,  
  force             IN BOOLEAN DEFAULT FALSE);
```

The `FORCE` parameter in this example defaults to `FALSE`. If the `instance` value is 0 (zero), job affinity is altered and any available instance can execute the job despite the value of `force`. If the `INSTANCE` value is positive and the `FORCE` parameter is `FALSE`, job affinity is altered only if the specified instance is running, or Oracle displays error ORA-23428.

If the `force` parameter is `TRUE`, any positive integer is acceptable as the job instance and the job affinity is altered. Oracle displays error ORA-23319 if the `instance` value is negative or NULL.

#### DBMS\_JOB.CHANGE

To alter user-definable parameters associated with a job, use the following syntax:

```
DBMS_JOB.CHANGE( JOB IN BINARY_INTEGER,
```

```
        what                IN VARCHAR2 DEFAULT NULL,  
next_date          IN DATE DEFAULT NULL,  
interval           IN VARCHAR2 DEFAULT NULL,  
instance           IN BINARY_INTEGER DEFAULT NULL,  
force              IN BOOLEAN DEFAULT FALSE );
```

Two parameters, `instance` and `force`, appear in this example. The default value of `instance` is `null` indicating that job affinity will not change.

The default value of `force` is `FALSE`. Oracle displays error `ORA-23428` if the specified instance is not running and error `ORA-23319` if the instance number is negative.

### **DBMS\_JOB.RUN**

The `force` parameter for `DBMS_JOB.RUN` defaults to `FALSE`. If `force` is `TRUE`, instance affinity is irrelevant for running jobs in the foreground process. If `force` is `FALSE`, the job can run in the foreground only in the specified instance. Oracle displays error `ORA-23428` if `force` is `FALSE` and the connected instance is the incorrect instance.

```
DBMS_JOB.RUN(  
    job      IN BINARY_INTEGER,  
    force    IN BOOLEAN DEFAULT FALSE);
```

## **Stopping a Job**

Note that, once a job is started and running, there is no easy way to stop the job.

## Summary of DBMS\_JOB Subprograms

**Table 78–1** *DBMS\_JOB Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">BROKEN Procedure</a> on page 78-7	Disables job execution
<a href="#">CHANGE Procedure</a> on page 78-8	Alters any of the user-definable parameters associated with a job
<a href="#">INSTANCE Procedure</a> on page 78-9	Assigns a job to be run by a instance
<a href="#">INTERVAL Procedure</a> on page 78-10	Alters the interval between executions for a specified job
<a href="#">NEXT_DATE Procedure</a> on page 78-11	Alters the next execution time for a specified job
<a href="#">REMOVE Procedure</a> on page 78-12	Removes specified job from the job queue
<a href="#">RUN Procedure</a> on page 78-13	Forces a specified job to run
<a href="#">SUBMIT Procedure</a> on page 78-14	Submits a new job to the job queue
<a href="#">USER_EXPORT Procedures</a> on page 78-16	Re-creates a given job for export, or re-creates a given job for export with instance affinity
<a href="#">WHAT Procedure</a> on page 78-17	Alters the job description for a specified job



## BROKEN Procedure

This procedure sets the broken flag. Broken jobs are never run.

### Syntax

```
DBMS_JOB.BROKEN (  
  job      IN  BINARY_INTEGER,  
  broken   IN  BOOLEAN,  
  next_date IN DATE DEFAULT SYSDATE);
```

### Parameters

**Table 78–2** *BROKEN Procedure Parameters*

Parameter	Description
job	Number of the job being run.
broken	Job broken: IN value is FALSE.
next_date	Date of the next refresh.

---

---

**Note:** If you set job as broken while it is running, Oracle resets the job's status to normal after the job completes. Therefore, only execute this procedure for jobs that are not running.

---

---

### Usage Notes

You must issue a COMMIT statement immediately after the statement.

## CHANGE Procedure

This procedure changes any of the fields a user can set in a job.

### Syntax

```
DBMS_JOB.CHANGE (
  job      IN  BINARY_INTEGER,
  what     IN  VARCHAR2,
  next_date IN DATE,
  interval IN  VARCHAR2,
  instance IN  BINARY_INTEGER DEFAULT NULL,
  force    IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 78–3** *CHANGE Procedure Parameters*

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.
next_date	Date of the next refresh.
interval	Date function; evaluated immediately before the job starts running.
instance	When a job is submitted, specifies which instance can run the job. This defaults to <code>NULL</code> , which indicates that instance affinity is not changed.
force	If this is <code>FALSE</code> , then the specified instance (to which the instance number change) must be running. Otherwise, the routine raises an exception.  If this is <code>TRUE</code> , then any positive integer is acceptable as the job instance.

### Usage Notes

- You must issue a `COMMIT` statement immediately after the statement.
- The parameters `instance` and `force` are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.
- If the parameters `what`, `next_date`, or `interval` are `NULL`, then leave that value as it is.

### Example

```
BEGIN
  DBMS_JOB.CHANGE(14144, null, null, 'sysdate+3');
  COMMIT;
END;
```

## INSTANCE Procedure

This procedure changes job instance affinity.

### Syntax

```
DBMS_JOB.INSTANCE (
  job          IN BINARY_INTEGER,
  instance     IN BINARY_INTEGER,
  force        IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 78–4** *INSTANCE Procedure Parameters*

Parameter	Description
job	Number of the job being run.
instance	When a job is submitted, a user can specify which instance can run the job.
force	If this is <code>TRUE</code> , then any positive integer is acceptable as the job instance. If this is <code>FALSE</code> (the default), then the specified instance must be running; otherwise the routine raises an exception.

### Usage Notes

You must issue a `COMMIT` statement immediately after the statement.

## INTERVAL Procedure

This procedure changes how often a job runs.

### Syntax

```
DBMS_JOB.INTERVAL (
  job      IN  BINARY_INTEGER,
  interval IN  VARCHAR2);
```

### Parameters

**Table 78–5** *INTERVAL Procedure Parameters*

Parameter	Description
job	Number of the job being run.
interval	Date function, evaluated immediately before the job starts running.

### Usage Notes

- If the job completes successfully, then this new date is placed in `next_date`. `interval` is evaluated by plugging it into the statement `select interval into next_date from dual`;
- The `interval` parameter must evaluate to a time in the future. Legal intervals include:

Interval	Description
'sysdate + 7'	Run once a week.
'next_day(sysdate, ''TUESDAY'')'	Run once every Tuesday.
'null'	Run only once.

- If `interval` evaluates to `NULL` and if a job completes successfully, then the job is automatically deleted from the queue.
- You must issue a `COMMIT` statement immediately after the statement.

## NEXT\_DATE Procedure

This procedure changes when an existing job next runs.

### Syntax

```
DBMS_JOB.NEXT_DATE (  
    job          IN BINARY_INTEGER,  
    next_date IN DATE);
```

### Parameters

**Table 78–6** NEXT\_DATE Procedure Parameters

Parameter	Description
job	Number of the job being run.
next_date	Date of the next refresh: it is when the job will be automatically run, assuming there are background processes attempting to run it.

### Usage Notes

You must issue a COMMIT statement immediately after the statement.

## REMOVE Procedure

This procedure removes an existing job from the job queue. This currently does not stop a running job.

### Syntax

```
DBMS_JOB.REMOVE (  
    job          IN BINARY_INTEGER );
```

### Parameters

**Table 78–7 REMOVE Procedure Parameters**

Parameter	Description
job	Number of the job being run.

### Usage Notes

You must issue a COMMIT statement immediately after the statement.

### Example

```
BEGIN  
    DBMS_JOB.REMOVE(14144);  
    COMMIT;  
END;
```

## RUN Procedure

This procedure runs job `JOB` now. It runs it even if it is broken.

Running the job recomputes `next_date`. See view `user_jobs`.

### Syntax

```
DBMS_JOB.RUN (
  job      IN  BINARY_INTEGER,
  force    IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 78–8** *RUN Procedure Parameters*

Parameter	Description
<code>job</code>	Number of the job being run.
<code>force</code>	If this is <code>TRUE</code> , then instance affinity is irrelevant for running jobs in the foreground process. If this is <code>FALSE</code> , then the job can be run in the foreground only in the specified instance.

### Example

```
EXECUTE DBMS_JOB.RUN(14144);
```

---

**Caution:** This re-initializes the current session's packages.

---

### Exceptions

An exception is raised if `force` is `FALSE`, and if the connected instance is the wrong one.

## SUBMIT Procedure

This procedure submits a new job. It chooses the job from the sequence `sys.jobseq`.

### Syntax

```
DBMS_JOB.SUBMIT (
  job      OUT BINARY_INTEGER,
  what     IN  VARCHAR2,
  next_date IN  DATE DEFAULT sysdate,
  interval IN  VARCHAR2 DEFAULT 'null',
  no_parse IN  BOOLEAN DEFAULT FALSE,
  instance IN  BINARY_INTEGER DEFAULT any_instance,
  force    IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 78–9 SUBMIT Procedure Parameters**

Parameter	Description
<code>job</code>	Number of the job being run.
<code>what</code>	PL/SQL procedure to run.
<code>next_date</code>	Next date when the job will be run.
<code>interval</code>	Date function that calculates the next time to run the job. The default is <code>NULL</code> . This must evaluate to either a future point in time or <code>NULL</code> .
<code>no_parse</code>	A flag. The default is <code>FALSE</code> . If this is set to <code>FALSE</code> , then Oracle parses the procedure associated with the job. If this is set to <code>TRUE</code> , then Oracle parses the procedure associated with the job the first time that the job is run.  For example, if you want to submit a job before you have created the tables associated with the job, then set this to <code>TRUE</code> .
<code>instance</code>	When a job is submitted, specifies which instance can run the job.
<code>force</code>	If this is <code>TRUE</code> , then any positive integer is acceptable as the job instance. If this is <code>FALSE</code> (the default), then the specified instance must be running; otherwise the routine raises an exception.

### Usage Notes

- You must issue a `COMMIT` statement immediately after the statement.
- The parameters `instance` and `force` are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

### Example

This submits a new job to the job queue. The job calls the procedure `DBMS_DDL.ANALYZE_OBJECT` to generate optimizer statistics for the table `DQUON.ACCOUNTS`. The statistics are based on a sample of half the rows of the `ACCOUNTS` table. The job is run every 24 hours:

```
VARIABLE jobno number;
BEGIN
```



```
DBMS_JOB.SUBMIT(:jobno,  
  'dbms_ddl.analyze_object(''TABLE'',  
  'DQUON', 'ACCOUNTS',  
  'ESTIMATE', NULL, 50);'  
  SYSDATE, 'SYSDATE + 1');  
COMMIT;  
END;  
/  
Statement processed.  
print jobno  
JOBNO  
-----  
14144
```

## USER\_EXPORT Procedures

There are two overloaded procedures. The first produces the text of a call to re-create the given job. The second alters instance affinity (*8i* and after) and preserves the compatibility.

### Syntax

```
DBMS_JOB.USER_EXPORT (  
  job      IN      BINARY_INTEGER,  
  mycall  IN OUT  VARCHAR2);  
  
DBMS_JOB.USER_EXPORT (  
  job      IN      BINARY_INTEGER,  
  mycall  IN OUT  VARCHAR2,  
  myinst  IN OUT  VARCHAR2);
```

### Parameters

**Table 78–10** *USER\_EXPORT Procedure Parameter*

Parameter	Description
job	Number of the job being run.
mycall	Text of a call to re-create the given job.
myinst	Text of a call to alter instance affinity.

## WHAT Procedure

This procedure changes what an existing job does, and replaces its environment.

### Syntax

```
DBMS_JOB.WHAT (  
  job      IN  BINARY_INTEGER,  
  what     IN  VARCHAR2);
```

### Parameters

**Table 78–11** WHAT Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.

### Usage Notes

- You must issue a COMMIT statement immediately after the statement.
- Some legal values of what (assuming the routines exist) are:
  - 'myproc(''10-JAN-82'', next\_date, broken);'
  - 'scott.emppackage.give\_raise(''JENKINS'', 30000.00);'
  - 'dbms\_job.remove(job);'



The DBMS\_LDAP package lets you access data from LDAP servers.

- [Documentation of DBMS\\_LDAP](#)

---

## Documentation of DBMS\_LDAP

For a complete description of this package within the context of Oracle Internet Directory, see DBMS\_LDAP in the *Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management*.

---

## DBMS\_LDAP\_UTL

The DBMS\_LDAP\_UTL package contains the Oracle Extension utility functions.

- [Documentation of DBMS\\_LDAP\\_UTL](#)

---

## Documentation of DBMS\_LDAP\_UTL

For a complete description of this package within the context of Oracle Internet Directory, see `DBMS_LDAP_UTL` in the *Oracle Internet Directory Application Developer's Guide*.



---

---

## DBMS\_LIBCACHE

The `DBMS_LIBCACHE` package consists of one subprogram that prepares the library cache on an Oracle instance by extracting SQL and PL/SQL from a remote instance and compiling this SQL locally without execution. The value of compiling the cache of an instance is to prepare the information the application requires to execute in advance of failover or switchover.

This chapter contains the following topics:

- [Using DBMS\\_LIBCACHE](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_LIBCACHE Subprograms](#)

---

## Using DBMS\_LIBCACHE

- [Overview](#)
- [Security Model](#)

## Overview

Compiling a shared cursor consists of open, parse, and bind operations, plus the type-checking and execution plan functions performed at the first execution. All of these steps are executed in advance by the package `DBMS_LIBCACHE` for `SELECT` statements. The open and parse functions are executed in advance for `PL/SQL` and `DML`. For `PL/SQL`, executing the parse phase has the effect of loading all library cache heaps other than the `MCODE`.

## Security Model

To execute `DBMS_LIBCACHE` you must directly access the same objects as do SQL statements. You can best accomplish this by utilizing the same user id as the original system on the remote system.

When there are multiple schema users, `DBMS_LIBCACHE` should be called for each.

Alternatively, `DBMS_LIBCACHE` may be called with the generic user `PARSER`. However, this user cannot parse the SQL that uses objects with access granted through roles. This is a standard PL/SQL security limitation.

## Summary of DBMS\_LIBCACHE Subprograms

**Table 81-1 DBMS\_LIBCACHE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">COMPILE_FROM_REMOTE Procedure</a> on page 81-6	Extracts SQL in batch from the source instance and compiles the SQL at the target instance

## COMPILE\_FROM\_REMOTE Procedure

This procedure extracts SQL in batch from the source instance and compiles the SQL at the target instance.

### Syntax

```
DBMS_LIBCACHE.COMPILE_FROM_REMOTE (
  p_db_link          IN      dbms_libcache$def.db_link%type,
  p_username         IN      VARCHAR2 default null,
  p_threshold_executions IN  NATURAL default 3,
  p_threshold_sharable_mem IN NATURAL default 1000,
  p_parallel_degree  IN      NATURAL default 1);
```

### Parameters

**Table 81–2** *COMPILE\_FROM\_REMOTE Procedure Parameters*

Parameter	Description
p_db_link	Database link to the source name (mandatory). The database link pointing to the instance that will be used for extracting the SQL statements. The user must have the role <code>SELECT_ON_CATALOG</code> at the source instance. For improved security, the connection may use a password file or LDAP authentication. The database link is mandatory only for releases with <code>dbms_libcache\$def.ACCESS_METHOD = DB_LINK_METHOD</code>
p_instance_name	(Reserved for future use). The name of the instance that will be used for extracting the SQL statements. The instance name must be unique for all instances excluding the local instance. The name is not case sensitive.
p_username	Source username (default is all users). The name of the username that will be used for extracting the SQL statements. The username is an optional parameter that is used to ensure the parsing user id is the same as that on the source instance. For an application where users connect as a single <code>user_id</code> , for example <code>APPS</code> , <code>APPS</code> is the parsing <code>user_id</code> that is recorded in the shared pool. To select only SQL statements parsed by <code>APPS</code> , enter the string 'APPS' in this field. To also select statements executed by batch, repeat the executing the procedure with the schema owner, for example <code>GL</code> . If the username is supplied, it must be valid. The name is not case sensitive.
p_threshold_executions	The lower bound for the number of executions, below which a SQL statement will not be selected for parsing. This parameter is optional. It allows the application to extract and compile statements with executions, for example, greater than 3. The default value is 1. This means SQL statements that have never executed, including invalid SQL statements, will not be extracted.

**Table 81-2 (Cont.) COMPILE\_FROM\_REMOTE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>p_threshold_sharable_mem</code>	The lower bound for the size of the shared memory consumed by the cursors on the source instance. Below this value a SQL statement will not be selected for parsing. This parameter is optional. It allows the application to extract and compile statements with shared memory for example, greater than 10000 bytes.
<code>p_parallel_degree</code>	The number of parallel jobs that execute to complete the parse operation. These tasks are spawned as parallel jobs against a sub-range of the SQL statements selected for parsing. This parameter is reserved for parallel compile jobs which are currently not implemented.





The DBMS\_LOB package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use DBMS\_LOB to access and manipulation specific parts of a LOB or complete LOBs.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_LOB](#)
  - Overview
  - Security Model
  - Constants
  - Datatypes
  - Operational Notes
  - Rules and Limits
  - Exceptions
- [Summary of DBMS\\_LOB Subprograms](#)

## Using DBMS\_LOB

---

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Datatypes](#)
- [Operational Notes](#)
- [Rules and Limits](#)
- [Exceptions](#)

## Overview

DBMS\_LOB can read and modify BLOBs, CLOBs, and NLOBs; it provides read-only operations for BFILEs. The bulk of the LOB operations are provided by this package.

## Security Model

This package must be created under SYS. Operations provided by this package are performed under the current calling user, not under the package owner SYS.

Any DBMS\_LOB subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user. Any DBMS\_LOB subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

When creating the procedure, users can set the AUTHID to indicate whether they want definer's rights or invoker's rights. For example:

```
CREATE PROCEDURE proc1 AUTHID DEFINER ...
```

or

```
CREATE PROCEDURE proc1 AUTHID CURRENT_USER ...
```

**See Also:** For more information on AUTHID and privileges, see *Oracle Database PL/SQL Language Reference*

You can provide secure access to BFILES using the DIRECTORY feature discussed in BFILENAME function in the *Oracle Database SecureFiles and Large Objects Developer's Guide* and the *Oracle Database SQL Language Reference*.

For information about the security model pertaining to temporary LOBs, see [Operational Notes](#).

## Constants

The DBMS\_LOB package uses the constants shown in following tables:

- [Table 82–1, "DBMS\\_LOB Constants - Basic"](#)
- [Table 82–2, "DBMS\\_LOB Constants - Option Types"](#)
- [Table 82–3, "DBMS\\_LOB Constants - Option Values"](#)
- [Table 82–4, "DBMS\\_LOB Constants - DBFS State Value Types"](#)
- [Table 82–5, "DBMS\\_LOB Constants - DBFS Cache Flags"](#)
- [Table 82–6, "DBMS\\_LOB Constants - Maximum 1-Byte ASCII Characters for Contenttype"](#)

**Table 82–1 DBMS\_LOB Constants - Basic**

Constant	Type	Value	Description
CALL	PLS_INTEGER	12	Create the TEMP LOB with call duration
FILE_READONLY	BINARY_INTEGER	0	Open the specified BFILE read-only
LOB_READONLY	BINARY_INTEGER	0	Open the specified LOB read-only
LOB_READWRITE	BINARY_INTEGER	1	Open the specified LOB read-write
LOBMAXSIZE	INTEGER	18446744073709551615	Maximum size of a LOB in bytes
SESSION	PLS_INTEGER	10	Create the TEMP LOB with session duration

**Table 82–2 DBMS\_LOB Constants - Option Types**

Constant	Definition	Value	Description
OPT_COMPRESS	BINARY_INTEGER	1	Set/Get the SECUREFILE compress option value
OPT_DEDUPLICATE	BINARY_INTEGER	4	Set/Get the SECUREFILE Deduplicate option value
OPT_ENCRYPT	BINARY_INTEGER	2	Get the SECUREFILE encrypt option value

**Table 82–3 DBMS\_LOB Constants - Option Values**

Constant	Definition	Value	Description
COMPRESS_OFF	BINARY_INTEGER	0	For <a href="#">SETOPTIONS Procedures</a> , set compress off; for <a href="#">GETOPTIONS Functions</a> , compress is off
COMPRESS_ON	BINARY_INTEGER	1	For <a href="#">SETOPTIONS Procedures</a> , set compress on; for <a href="#">GETOPTIONS Functions</a> , compress is on
DEDUPLICATE_OFF	BINARY_INTEGER	0	For <a href="#">SETOPTIONS Procedures</a> , set deduplicate is off; for <a href="#">GETOPTIONS Functions</a> , deduplicate is off

**Table 82–3 (Cont.) DBMS\_LOB Constants - Option Values**

Constant	Definition	Value	Description
DEDUPLICATE_ON	BINARY_INTEGER	4	For <a href="#">SETOPTIONS Procedures</a> , set deduplicate is on; for <a href="#">GETOPTIONS Functions</a> , deduplicate is on
ENCRYPT_OFF	BINARY_INTEGER	0	For <a href="#">GETOPTIONS Functions</a> , encrypt is off
ENCRYPT_ON	BINARY_INTEGER	2	For <a href="#">GETOPTIONS Functions</a> , encrypt is on

**Table 82–4 DBMS\_LOB Constants - DBFS State Value Types**

Constant	Definition	Value	Description
DBFS_LINK_NEVER	PLS_INTEGER	0	LOB has never been archived
DBFS_LINK_NO	PLS_INTEGER	2	LOB was archived, but as been read back in to the RDBMS
DBFS_LINK_YES	PLS_INTEGER	1	LOB is currently archived

**Table 82–5 DBMS\_LOB Constants - DBFS Cache Flags**

Constant	Definition	Value	Description
DBFS_LINK_CACHE	PLS_INTEGER	1	Put the LOB data to the archive, but keep the data in the RDBMS as a cached version
DBFS_LINK_NOCACHE	PLS_INTEGER	0	Put the LOB data to the archive, and remove the data from the RDBMS.

**Table 82–6 DBMS\_LOB Constants - Maximum 1-Byte ASCII Characters for Contenttype**

Constant	Definition	Value	Description
CONTENTTYPE_MAX_SIZE	PLS_INTEGER	128	Maximum number of bytes allowed in the content type string

## Datatypes

The DBMS\_LOB package uses the datatypes shown in [Table 82–7](#).

**Table 82–7 Datatypes Used by DBMS\_LOB**

Type	Description
BLOB	Source or destination binary LOB.
RAW	Source or destination RAW buffer (used with BLOB).
CLOB	Source or destination character LOB (including NCLOB).
VARCHAR2	Source or destination character buffer (used with CLOB and NCLOB).
INTEGER	Specifies the size of a buffer or LOB, the offset into a LOB, or the amount to access.
BFILE	Large, binary object stored outside the database.

The DBMS\_LOB package defines no special types.

An NCLOB is a CLOB for holding fixed-width and varying-width, multibyte national character sets.

The clause ANY\_CS in the specification of DBMS\_LOB subprograms for CLOBs enables the CLOB type to accept a CLOB or NCLOB locator variable as input.

## Operational Notes

All `DBMS_LOB` subprograms work based on `LOB` locators. For the successful completion of `DBMS_LOB` subprograms, you must provide an input locator that represents a `LOB` that already exists in the database tablespaces or external file system. See also Chapter 1 of *Oracle Database SecureFiles and Large Objects Developer's Guide*.

To use `LOBs` in your database, you must first use SQL data definition language (DDL) to define the tables that contain `LOB` columns.

- [Internal `LOBs`](#)
- [External `LOBs`](#)
- [Temporary `LOBs`](#)

### Internal `LOBs`

To populate your table with internal `LOBs` after `LOB` columns are defined in a table, you use the SQL data manipulation language (DML) to initialize or populate the locators in the `LOB` columns.

### External `LOBs`

For an external `LOB` (`BFILE`) to be represented by a `LOB` locator, you must:

- Ensure that a `DIRECTORY` object representing a valid, existing physical directory has been defined, and that physical files (the `LOBs` you plan to add) exist with read permission for the database. If your operating system uses case-sensitive path names, then be sure you specify the directory in the correct format.
- Pass the `DIRECTORY` object and the filename of the external `LOB` you are adding to the `BFILENAME` function to create a `LOB` locator for your external `LOB`.

Once you have completed these tasks, you can insert or update a row containing a `LOB` column using the specified `LOB` locator.

After the `LOBs` are defined and created, you can then `SELECT` from a `LOB` locator into a local PL/SQL `LOB` variable and use this variable as an input parameter to `DBMS_LOB` for access to the `LOB` value.

For details on the different ways to do this, you must refer to the section of the *Oracle Database SecureFiles and Large Objects Developer's Guide* that describes "Accessing External `LOBs` (`BFILES`)."

### Temporary `LOBs`

The database supports the definition, creation, deletion, access, and update of temporary `LOBs`. Your temporary tablespace stores the temporary `LOB` data. Temporary `LOBs` are not permanently stored in the database. Their purpose is mainly to perform transformations on `LOB` data.

For temporary `LOBs`, you must use the OCI, PL/SQL, or another programmatic interface to create or manipulate them. Temporary `LOBs` can be either `BLOBs`, `CLOBs`, or `NCLOBs`.

A temporary `LOB` is empty when it is created. By default, all temporary `LOBs` are deleted at the end of the session in which they were created. If a process dies unexpectedly or if the database crashes, then temporary `LOBs` are deleted, and the space for temporary `LOBs` is freed.



There is also an interface to let you group temporary LOBs together into a logical bucket. The duration represents this logical store for temporary LOBs. Each temporary LOB can have separate storage characteristics, such as `CACHE/ NOCACHE`. There is a default store for every session into which temporary LOBs are placed if you don't specify a specific duration. Additionally, you are able to perform a free operation on durations, which causes all contents in a duration to be freed.

There is no support for consistent read (CR), undo, backup, parallel processing, or transaction management for temporary LOBs. Because CR and rollbacks are not supported for temporary LOBs, you must free the temporary LOB and start over again if you encounter an error.

Because CR, undo, and versions are not generated for temporary LOBs, there is potentially a performance impact if you assign multiple locators to the same temporary LOB. Semantically, each locator should have its own copy of the temporary LOB.

A copy of a temporary LOB is created if the user modifies the temporary LOB while another locator is also pointing to it. The locator on which a modification was performed now points to a new copy of the temporary LOB. Other locators no longer see the same data as the locator through which the modification was made. A deep copy was not incurred by permanent LOBs in these types of situations, because CR snapshots and version pages enable users to see their own versions of the LOB cheaply.

You can gain pseudo-REF semantics by using pointers to locators in OCI and by having multiple pointers to locators point to the same temporary LOB locator, if necessary. In PL/SQL, you must avoid using more than one locator for each temporary LOB. The temporary LOB locator can be passed by reference to other procedures.

Because temporary LOBs are not associated with any table schema, there are no meanings to the terms in-row and out-of-row temporary LOBs. Creation of a temporary LOB instance by a user causes the engine to create and return a locator to the LOB data. The PL/SQL `DBMS_LOB` package, `PRO*C/C++`, OCI, and other programmatic interfaces operate on temporary LOBs through these locators just as they do for permanent LOBs.

There is no support for client side temporary LOBs. All temporary LOBs reside in the server.

Temporary LOBs do not support the `EMPTY_BLOB` or `EMPTY_CLOB` functions that are supported for permanent LOBs. The `EMPTY_BLOB` function specifies the fact that the LOB is initialized, but not populated with any data.

A temporary LOB instance can only be destroyed by using OCI or the `DBMS_LOB` package by using the appropriate `FREETEMPORARY` or `OCIDurationEnd` statement.

A temporary LOB instance can be accessed and modified using appropriate OCI and `DBMS_LOB` statements, just as for regular permanent internal LOBs. To make a temporary LOB permanent, you must explicitly use the OCI or `DBMS_LOB COPY` command, and copy the temporary LOB into a permanent one.

Security is provided through the LOB locator. Only the user who created the temporary LOB is able to see it. Locators are not expected to be able to pass from one user's session to another. Even if someone did pass a locator from one session to another, they would not access the temporary LOBs from the original session. Temporary LOB lookup is localized to each user's own session. Someone using a locator from somewhere else is only able to access LOBs within his own session that have the same LOB ID. Users should not try to do this, but if they do, they are not able to affect anyone else's data.

The database keeps track of temporary LOBs for each session in a `v$` view called `V$TEMPORARY_LOBS`, which contains information about how many temporary LOBs exist for each session. `v$` views are for DBA use. From the session, the database can determine which user owns the temporary LOBs. By using `V$TEMPORARY_LOBS` in conjunction with `DBA_SEGMENTS`, a DBA can see how much space is being used by a session for temporary LOBs. These tables can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

The following notes are specific to temporary LOBs:

1. All functions in `DBMS_LOB` return `NULL` if any of the input parameters are `NULL`. All procedures in `DBMS_LOB` raise an exception if the `LOB` locator is input as `NULL`.
2. Operations based on `CLOBs` do not verify if the character set IDs of the parameters (`CLOB` parameters, `VARCHAR2` buffers and patterns, and so on) match. It is the user's responsibility to ensure this.
3. Data storage resources are controlled by the DBA by creating different temporary tablespaces. DBAs can define separate temporary tablespaces for different users, if necessary.

**See Also:** *Oracle Database PL/SQL Language Reference* for more information on `NOCOPY` syntax

## Rules and Limits

- [General Rules and Limits](#)
- [Rules and Limits Specific to External Files \(BFILEs\)](#)
- [Maximum LOB Size](#)
- [Maximum Buffer Size](#)

### General Rules and Limits

- Oracle Database does not support constraints on columns or attributes whose type is a LOB, with the following exception: NOT NULL constraints are supported for a LOB column or attribute.
- The following rules apply in the specification of subprograms in this package:
  - length, offset, and amount parameters for subprograms operating on BLOBs and BFILEs must be specified in terms of *bytes*.
  - length, offset, and amount parameters for subprograms operating on CLOBs must be specified in terms of *characters*.

In multi-byte character sets, it is not possible to interpret these offsets correctly. As a result, SUBSTR raises the following error: ORA-22998: CLOB or NCLOB in multibyte character set not supported.

- A subprogram raises an INVALID\_ARGVAL exception if the following restrictions are not followed in specifying values for parameters (unless otherwise specified):
  1. Only positive, absolute offsets from the beginning of LOB data are permitted: Negative offsets from the tail of the LOB are not permitted.
  2. Only positive, nonzero values are permitted for the parameters that represent size and positional quantities, such as amount, offset, newlen, nth, and so on. Negative offsets and ranges observed in SQL string functions and operators are not permitted.
  3. The value of offset, amount, newlen, nth must not exceed the value lobmaxsize 18446744073709551615 ( $2^{64}$ ) in any DBMS\_LOB subprogram.
  4. For CLOBs consisting of fixed-width multibyte characters, the maximum value for these parameters must not exceed  $(lobmaxsize/character\_width\_in\_bytes)$  characters.

For example, if the CLOB consists of 2-byte characters, such as:

```
JA16SJISFIXED
```

Then, the maximum amount value should not exceed:

```
18446744073709551615/2 = 9223372036854775807
```

- PL/SQL language specifications stipulate an upper limit of 32767 bytes (not characters) for RAW and VARCHAR2 parameters used in DBMS\_LOB subprograms. For example, if you declare a variable to be:

```
charbuf VARCHAR2(3000)
```

Then, charbuf can hold 3000 single byte characters or 1500 2-byte fixed width characters. This has an important consequence for DBMS\_LOB subprograms for CLOBs and NCLOBs.

- The `%CHARSET` clause indicates that the form of the parameter with `%CHARSET` must match the form of the `ANY_CS` parameter to which it refers.

For example, in `DBMS_LOB` subprograms that take a `VARCHAR2` buffer parameter, the form of the `VARCHAR2` buffer must match the form of the `CLOB` parameter. If the input `LOB` parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input `LOB` parameter is of type `CLOB`, then the buffer must contain `CHAR` data.

For `DBMS_LOB` subprograms that take two `CLOB` parameters, both `CLOB` parameters must have the same form; that is, they must both be `NCLOBs`, or they must both be `CLOBs`.

- If the value of `amount` plus the `offset` exceeds the maximum `LOB` size allowed by the database, then access exceptions are raised.

Under these input conditions, read subprograms, such as `READ`, `COMPARE`, `INSTR`, and `SUBSTR`, read until `End of Lob/File` is reached. For example, for a `READ` operation on a `BLOB` or `BFILE`, if the user specifies `offset` value of 3 GB and an `amount` value of 2 GB on a `LOB` that is 4GB in size, then `READ` returns only 1GB (4GB-3GB) bytes.

- Functions with `NULL` or invalid input values for parameters return a `NULL`. Procedures with `NULL` values for destination `LOB` parameters raise exceptions.
- Operations involving patterns as parameters, such as `COMPARE`, `INSTR`, and `SUBSTR` do not support regular expressions or special matching characters (such as `%` in the `LIKE` operator in SQL) in the `pattern` parameter or substrings.
- The `End of LOB` condition is indicated by the `READ` procedure using a `NO_DATA_FOUND` exception. This exception is raised only upon an attempt by the user to read beyond the end of the `LOB`. The `READ` buffer for the last read contains 0 bytes.
- For consistent `LOB` updates, you must lock the row containing the destination `LOB` before making a call to any of the procedures (mutators) that modify `LOB` data.
- Unless otherwise stated, the default value for an `offset` parameter is 1, which indicates the first byte in the `BLOB` or `BFILE` data, and the first character in the `CLOB` or `NCLOB` value. No default values are specified for the `amount` parameter — you must input the values explicitly.
- You must lock the row containing the destination internal `LOB` before calling any subprograms that modify the `LOB`, such as `APPEND`, `COPY`, `ERASE`, `TRIM`, or `WRITE`. These subprograms do not implicitly lock the row containing the `LOB`.

### Rules and Limits Specific to External Files (BFILES)

- The subprograms `COMPARE`, `INSTR`, `READ`, `SUBSTR`, `FILECLOSE`, `FILECLOSEALL` and `LOADFROMFILE` operate only on an *opened* `BFILE` locator; that is, a successful `FILEOPEN` call must precede a call to any of these subprograms.
- For the functions `FILEEXISTS`, `FILEGETNAME` and `GETLENGTH`, a file's open/close status is unimportant; however, the file must exist physically, and you must have adequate privileges on the `DIRECTORY` object and the file.
- `DBMS_LOB` does not support any concurrency control mechanism for `BFILE` operations.
- In the event of several open files in the session whose closure has not been handled properly, you can use the `FILECLOSEALL` subprogram to close all files opened in the session and resume file operations from the beginning.

- If you are the creator of a DIRECTORY, or if you have system privileges, then use the CREATE OR REPLACE, DROP, and REVOKE statements in SQL with extreme caution.

If you, or other grantees of a particular directory object, have several open files in a session, then any of the preceding commands can adversely affect file operations. In the event of such abnormal termination, your only choice is to invoke a program or anonymous block that calls FILECLOSEALL, reopen your files, and restart your file operations.

- All files opened during a user session are implicitly closed at the end of the session. However, Oracle strongly recommends that you close the files after *both* normal and abnormal termination of operations on the BFILE.

In the event of normal program termination, proper file closure ensures that the number of files that are open simultaneously in the session remains less than SESSION\_MAX\_OPEN\_FILES.

In the event of abnormal program termination from a PL/SQL program, it is imperative that you provide an exception handler that ensures closure of all files opened in that PL/SQL program. This is necessary because after an exception occurs, only the exception handler has access to the BFILE variable in its most current state.

After the exception transfers program control outside the PL/SQL program block, all references to the open BFILES are lost. The result is a larger open file count which may or may not exceed the SESSION\_MAX\_OPEN\_FILES value.

For example, consider a READ operation past the end of the BFILE value, which generates a NO\_DATA\_FOUND exception:

```
-- This assumes a directory 'DDD' whose path is already known
DECLARE
    fil BFILE:= bfilename('DDD', 'filename.foo');
    pos INTEGER;
    amt BINARY_INTEGER;
    buf RAW(40);
BEGIN
    SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
    dbms_lob.open(fil, dbms_lob.lob_readonly);
    amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
    dbms_lob.read(fil, amt, pos, buf);
    dbms_output.put_line('Read F1 past EOF: ' ||
        utl_raw.cast_to_varchar2(buf));
    dbms_lob.close(fil);
END;
```

```
ORA-01403: no data found
ORA-06512: at "SYS.DBMS_LOB", line 373
ORA-06512: at line 10
```

After the exception has occurred, the BFILE locator variable file goes out of scope, and no further operations on the file can be done using that variable. Therefore, the solution is to use an exception handler:

```
DECLARE
    fil BFILE;
    pos INTEGER;
    amt BINARY_INTEGER;
    buf RAW(40);
BEGIN
```

```
SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
dbms_lob.open(fil, dbms_lob.lob_readonly);
amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
dbms_lob.read(fil, amt, pos, buf);
dbms_output.put_line('Read F1 past EOF: '||
    utl_raw.cast_to_varchar2(buf));
dbms_lob.close(fil);
exception
WHEN no_data_found
THEN
    BEGIN
        dbms_output.put_line('End of File reached. Closing file');
        dbms_lob.fileclose(fil);
        -- or dbms_lob.filecloseall if appropriate
    END;
END;
```

```
Statement processed.
End of File reached. Closing file
```

In general, you should ensure that files opened in a PL/SQL block using `DBMS_LOB` are closed before normal or abnormal termination of the block.

### Maximum LOB Size

The maximum size for LOBs supported by the database is equal to the value of the blocksize of the tablespace the LOB column resides in times the value  $2^{32}-1$  (4294967295). This allows for a maximum LOB size ranging from 8 terabytes to 128 terabytes.

### Maximum Buffer Size

The maximum buffer size, 32767 bytes, is represented by `maxbufsize`.

## Exceptions

**Table 82–8 DBMS\_LOB Exceptions**

Exception	Code	Description
ACCESS_ERROR	22925	You are trying to write too much data to the LOB: LOB size is limited to 4 gigabytes.
BUFFERING_ENABLED	22279	Cannot perform operation with LOB buffering enabled
CONTENTTYPE_TOOLONG	43859	The length of the <code>contenttype</code> string exceeds the defined maximum. Modify the length of the <code>contenttype</code> string and retry the operation.
CONTENTTYPEBUF_WRONG	43862	The length of the <code>contenttype</code> buffer is less than defined constant. Modify the length of the <code>contenttype</code> buffer and retry the operation.
INVALID_ARGVAL	21560	The argument is expecting a non-NULL, valid value but the argument value passed in is NULL, invalid, or out of range.
INVALID_DIRECTORY	22287	The directory used for the current operation is not valid if being accessed for the first time, or if it has been modified by the DBA since the last access.
NO_DATA_FOUND	1403	ENDOFLOB indicator for looping read operations. This is not a hard error.
NOEXIST_DIRECTORY	22285	The directory leading to the file does not exist.
NOPRIV_DIRECTORY	22286	The user does not have the necessary access privileges on the directory or the file for the operation.
OPEN_TOOMANY	22290	The number of open files has reached the maximum limit.
OPERATION_FAILED	22288	The operation attempted on the file failed.
QUERY_WRITE	14553	Cannot perform a LOB write inside a query or PDML slave
SECUREFILE_BADLOB	43856	A non-SECUREFILE LOB type was used in a SECUREFILE only call
SECUREFILE_BADPARAM	43857	An invalid argument was passed to a SECUREFILE subprogram
SECUREFILE_MARKERASED	43861	The mark provided to a FRAGMENT_* operation has been deleted
SECUREFILE_OUTOFBOUNDS	43883	Attempted to perform a FRAGMENT_* operation past the LOB end
UNOPENED_FILE	22289	The file is not open for the required operation to be performed.
VALUE_ERROR	6502	PL/SQL error for invalid values to subprogram's parameters.

---

## Summary of DBMS\_LOB Subprograms

**Table 82–9 DBMS\_LOB Package Subprograms**

Subprogram	Description
<a href="#">APPEND Procedures</a> on page 82-19	Appends the contents of the source LOB to the destination LOB
<a href="#">CLOSE Procedure</a> on page 82-20	Closes a previously opened internal or external LOB
<a href="#">COMPARE Functions</a> on page 82-21	Compares two entire LOBs or parts of two LOBs
<a href="#">CONVERTTOBLOB Procedure</a> on page 82-23	Reads character data from a source CLOB or NCLOB instance, converts the character data to the specified character, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets
<a href="#">CONVERTTOCLOB Procedure</a> on page 82-26	Takes a source BLOB instance, converts the binary data in the source instance to character data using the specified character, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets
<a href="#">COPY Procedures</a> on page 82-29	Copies all, or part, of the source LOB to the destination LOB
<a href="#">COPY_DBFS_LINK Procedures</a> on page 82-31	Copies the DBFS link in the source LOB to the destination LOB
<a href="#">COPY_FROM_DBFS_LINK</a> on page 82-32	Retrieves the data for the LOB from the DBFS store
<a href="#">CREATETEMPORARY Procedures</a> on page 82-33	Creates a temporary BLOB or CLOB and its corresponding index in the user's default temporary tablespace
<a href="#">DBFS_LINK_GENERATE_PATH Functions</a> on page 82-34	Returns a unique file path name for use in creating a DBFS Link
<a href="#">ERASE Procedures</a> on page 82-35	Erases all or part of a LOB
<a href="#">FILECLOSE Procedure</a> on page 82-37	Closes the file
<a href="#">FILECLOSEALL Procedure</a> on page 82-38	Closes all previously opened files
<a href="#">FILEEXISTS Function</a> on page 82-39	Checks if the file exists on the server
<a href="#">FILEGETNAME Procedure</a> on page 82-40	Gets the directory object name and file name
<a href="#">FILEISOPEN Function</a> on page 82-41	Checks if the file was opened using the input BFILE locators
<a href="#">FILEOPEN Procedure</a> on page 82-42	Opens a file
<a href="#">FRAGMENT_DELETE Procedure</a> on page 82-43	Deletes the data at the specified offset for the specified length from the LOB
<a href="#">FRAGMENT_INSERT Procedures</a> on page 82-44	Inserts the specified data (limited to 32K) into the LOB at the specified offset



**Table 82–9 (Cont.) DBMS\_LOB Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">FRAGMENT_MOVE Procedure</a> on page 82-45	Moves the amount of bytes (BLOB) or characters (CLOB/NCLOB) from the specified offset to the new offset specified
<a href="#">FRAGMENT_REPLACE Procedures</a> on page 82-46	Replaces the data at the specified offset with the specified data (not to exceed 32k)
<a href="#">FREETEMPORARY Procedures</a> on page 82-48	Frees the temporary BLOB or CLOB in the default temporary tablespace
<a href="#">GET_DBFS_LINK Functions</a> on page 82-49	Returns the DBFS Link path associated with the specified SecureFile
<a href="#">GET_DBFS_LINK_STATE Procedures</a> on page 82-50	Retrieves the current DBFS Link state of the specified SecureFile
<a href="#">GETCHUNKSIZE Functions</a> on page 82-53	Returns the amount of space used in the LOB chunk to store the LOB value
<a href="#">GETCONTENTTYPE Functions</a> on page 82-51	Returns the content ID string previously set by means of the <a href="#">SETCONTENTTYPE Procedure</a>
<a href="#">GETLENGTH Functions</a> on page 82-55	Gets the length of the LOB value
<a href="#">GETOPTIONS Functions</a> on page 82-56	Obtains settings corresponding to the <code>option_type</code> field for a particular LOB
<a href="#">GET_STORAGE_LIMIT Function</a> on page 82-52	Returns the storage limit for LOBs in your database configuration
<a href="#">INSTR Functions</a> on page 82-57	Returns the matching position of the <i>n</i> th occurrence of the pattern in the LOB
<a href="#">ISOPEN Functions</a> on page 82-59	Checks to see if the LOB was already opened using the input locator
<a href="#">ISTEMPORARY Functions</a> on page 82-60	Checks if the locator is pointing to a temporary LOB
<a href="#">LOADBLOBFROMFILE Procedure</a> on page 82-61	Loads BFILE data into an internal BLOB
<a href="#">LOADCLOBFROMFILE Procedure</a> on page 82-63	Loads BFILE data into an internal CLOB
<a href="#">LOADFROMFILE Procedure</a> on page 82-67	Loads BFILE data into an internal LOB
<a href="#">MOVE_TO_DBFS_LINK Procedures</a> on page 82-69	Writes the specified SecureFile data to the DBFS store
<a href="#">OPEN Procedures</a> on page 82-70	Opens a LOB (internal, external, or temporary) in the indicated mode
<a href="#">READ Procedures</a> on page 82-72	Reads data from the LOB starting at the specified offset
<a href="#">SET_DBFS_LINK Procedures</a> on page 82-74	Links the specified SecureFile to the specified path name. It does not copy the data to the path
<a href="#">SETCONTENTTYPE Procedure</a> on page 82-75	Sets the content type string for the data in the LOB
<a href="#">SETOPTIONS Procedures</a> on page 82-76	Enables CSCE features on a per-LOB basis, overriding the default LOB column settings
<a href="#">SUBSTR Functions</a> on page 82-77	Returns part of the LOB value starting at the specified offset

**Table 82–9 (Cont.) DBMS\_LOB Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">TRIM Procedures</a> on page 82-79	Trims the LOB value to the specified shorter length
<a href="#">WRITE Procedures</a> on page 82-81	Writes data to the LOB from a specified offset
<a href="#">WRITEAPPEND Procedures</a> on page 82-83	Writes a buffer to the end of a LOB

## APPEND Procedures

This procedure appends the contents of a source internal LOB to a destination LOB. It appends the complete source LOB.

### Syntax

```
DBMS_LOB.APPEND (
  dest_lob IN OUT NOCOPY BLOB,
  src_lob  IN          BLOB);

DBMS_LOB.APPEND (
  dest_lob IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  src_lob  IN          CLOB  CHARACTER SET dest_lob%CHARSET);
```

### Parameters

**Table 82–10 APPEND Procedure Parameters**

Parameter	Description
dest_lob	Locator for the internal LOB to which the data is to be appended.
src_lob	Locator for the internal LOB from which the data is to be read.

### Exceptions

**Table 82–11 APPEND Procedure Exceptions**

Exception	Description
VALUE_ERROR	Either the source or the destination LOB is NULL.
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML slave
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on either LOB

### Usage Notes

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

- If APPEND is called on a LOB that has been archived, it implicitly gets the LOB before the first byte is written

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## CLOSE Procedure

This procedure closes a previously opened internal or external LOB.

### Syntax

```
DBMS_LOB.CLOSE (
    lob_loc    IN OUT NOCOPY BLOB);

DBMS_LOB.CLOSE (
    lob_loc    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);

DBMS_LOB.CLOSE (
    file_loc   IN OUT NOCOPY BFILE);
```

### Parameters

**Table 82–12** *CLOSE Procedure Parameters*

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .

### Exceptions

No error is returned if the BFILE exists but is not opened. An error is returned if the LOB is not open.

### Usage Notes

CLOSE requires a round-trip to the server for both internal and external LOBs. For internal LOBs, CLOSE triggers other code that relies on the close call, and for external LOBs (BFILES), CLOSE actually closes the server-side operating system file.

It is not mandatory that you wrap all LOB operations inside the Open/Close interfaces. However, if you open a LOB, you must close it before you commit or rollback the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## COMPARE Functions

This function compares two entire LOBs or parts of two LOBs.

### Syntax

```
DBMS_LOB.COMPARE (
  lob_1          IN BLOB,
  lob_2          IN BLOB,
  amount         IN INTEGER := DBMS_LOB.LOBMAXSIZE,
  offset_1       IN INTEGER := 1,
  offset_2       IN INTEGER := 1)
RETURN INTEGER;
```

```
DBMS_LOB.COMPARE (
  lob_1          IN CLOB CHARACTER SET ANY_CS,
  lob_2          IN CLOB CHARACTER SET lob_1%CHARSET,
  amount         IN INTEGER := DBMS_LOB.LOBMAXSIZE,
  offset_1       IN INTEGER := 1,
  offset_2       IN INTEGER := 1)
RETURN INTEGER;
```

```
DBMS_LOB.COMPARE (
  lob_1          IN BFILE,
  lob_2          IN BFILE,
  amount         IN INTEGER,
  offset_1       IN INTEGER := 1,
  offset_2       IN INTEGER := 1)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(COMPARE, WNDS, WNPS, RNDS, RNPS);
```

### Parameters

**Table 82–13 COMPARE Function Parameters**

Parameter	Description
lob_1	LOB locator of first target for comparison.
lob_2	LOB locator of second target for comparison.
amount	Number of bytes (for BLOBs) or characters (for CLOBs/NCLOBs) to compare.
offset_1	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.
offset_2	Offset in bytes or characters on the second LOB (origin: 1) for the comparison.

### Return Values

- INTEGER: 0 if the comparison succeeds, nonzero if not.
- NULL, if any of amount, offset\_1 or offset\_2 is not a valid LOB offset value. A valid offset is within the range of 1 to LOBMAXSIZE inclusive.

## Usage Notes

- You can only compare LOBs of the same datatype (LOBs of BLOB type with other BLOBs, and CLOBs with CLOBs, and BFILEs with BFILEs). For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.
- COMPARE returns 0 if the data exactly matches over the range specified by the offset and amount parameters. COMPARE returns -1 if the first CLOB is less than the second, and 1 if it is greater.
- For fixed-width  $n$ -byte CLOBs, if the input amount for COMPARE is specified to be greater than  $(DBMS\_LOB.LOBMAXSIZE/n)$ , then COMPARE matches characters in a range of size  $(DBMS\_LOB.LOBMAXSIZE/n)$ , or  $\text{Max}(\text{length}(\text{clob1}), \text{length}(\text{clob2}))$ , whichever is lesser.
- If COMPARE is called on any LOB that has been archived, it implicitly gets the LOB before the compare begins.

## Exceptions

**Table 82–14 COMPARE Function Exceptions for BFILE operations**

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on either LOB

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## CONVERTTOBLOB Procedure

This procedure reads character data from a source CLOB or NCLOB instance, converts the character data to the character set you specify, writes the converted data to a destination BLOB instance in binary format, and returns the new offsets. You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

### Syntax

```
DBMS_LOB.CONVERTTOBLOB (
  dest_lob      IN OUT  NOCOPY  BLOB,
  src_clob      IN      CLOB CHARACTER SET ANY_CS,
  amount        IN      INTEGER,
  dest_offset   IN OUT  INTEGER,
  src_offset    IN OUT  INTEGER,
  blob_csid     IN      NUMBER,
  lang_context  IN OUT  INTEGER,
  warning       OUT     INTEGER);
```

### Parameters

**Table 82–15** *CONVERTTOBLOB Procedure Parameters*

Parameter	Description
dest_lob	LOB locator of the destination LOB instance.
src_clob	LOB locator of the source LOB instance.
amount	Number of characters to convert from the source LOB. If you want to copy the entire LOB, pass the constant DBMS_LOB.LOBBMAXSIZE. If you pass any other value, it must be less than or equal to the size of the LOB.
dest_offset	(IN) Offset in bytes in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. (OUT) The new offset in bytes after the end of the write.
src_offset	(IN) Offset in characters in the source LOB for the start of the read. (OUT) Offset in characters in the source LOB right after the end of the read.
blob_csid	Desired character set ID of the converted data.
lang_context	(IN) Language context, such as shift status, for the current conversion. (OUT) The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.

**Table 82–15 (Cont.) CONVERTTOBLOB Procedure Parameters**

Parameter	Description
warning	<p>(OUT) Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message.</p> <p>Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, '?') is used in place of the inconvertible character. The return value of this error message is defined as the constant <code>warn_inconvertible_char</code> in the <code>DBMS_LOB</code> package.</p>

## Usage Notes

### Preconditions

Before calling the `CONVERTTOBLOB` procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked. To lock the row, select the LOB using the `FOR UPDATE` clause of the `SELECT` statement.

### Constants and Defaults

All parameters are required. You must pass a variable for each `OUT` or `IN OUT` parameter. You must pass either a variable or a value for each `IN` parameter.

[Table 82–16](#) gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the `dbmslob.sql` package specification file.

**Table 82–16 DBMS\_LOB.CONVERTTOBLOB Typical Values**

Parameter	Value	Description
amount	<code>LOBMAXSIZE</code> (IN)	convert the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning
blob_csid	<code>DEFAULT_CSID</code> (IN)	default CSID, use same CSID as source LOB
lang_context	<code>DEFAULT_LANG_CTX</code> (IN)	default language context
warning	<code>NO_WARNING</code> (OUT)	no warning message, success
	<code>WARN_INCONVERTIBLE_CHAR</code> (OUT)	character in source cannot be properly converted

### General Notes

- You must specify the desired character set for the destination LOB in the `blob_csid` parameter. You can pass a zero value for `blob_csid`. When you do so, the database assumes that the desired character set is the same as the source LOB character set.



- You must specify the offsets for both the source and destination LOBs, and the number of characters to copy from the source LOB. The `amount` and `src_offset` values are in characters and the `dest_offset` is in bytes. To convert the entire LOB, you can specify `LOBMAXSIZE` for the `amount` parameter.
- `CONVERTTLOBLOB` gets the source and/or destination LOBs as necessary prior to conversion and write of the data.

## Exceptions

Table 82–17 gives possible exceptions this procedure can throw. The first column lists the exception string and the second column describes the error conditions that can cause the exception.

**Table 82–17** *CONVERTTLOBLOB Procedure Exceptions*

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	One or more of the following: <ul style="list-style-type: none"> <li>- <code>src_offset</code> or <code>dest_offset</code> &lt; 1.</li> <li>- <code>src_offset</code> or <code>dest_offset</code> &gt; <code>LOBMAXSIZE</code>.</li> <li>- <code>amount</code> &lt; 1.</li> <li>- <code>amount</code> &gt; <code>LOBMAXSIZE</code>.</li> </ul>

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information on using LOBs in application development

## CONVERTTOCLOB Procedure

This procedure takes a source BLOB instance, converts the binary data in the source instance to character data using the character set you specify, writes the character data to a destination CLOB or NCLOB instance, and returns the new offsets. You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

### Syntax

```
DBMS_LOB.CONVERTTOCLOB (
    dest_lob      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
    src_blob      IN              BLOB,
    amount        IN              INTEGER,
    dest_offset   IN OUT          INTEGER,
    src_offset    IN OUT          INTEGER,
    blob_csid     IN              NUMBER,
    lang_context  IN OUT          INTEGER,
    warning       OUT             INTEGER);
```

### Parameters

**Table 82–18** *CONVERTTOCLOB Procedure Parameters*

Parameter	Description
dest_lob	LOB locator of the destination LOB instance.
src_blob	LOB locator of the source LOB instance.
amount	Number of bytes to convert from the source LOB.  If you want to copy the entire BLOB, pass the constant <code>DBMS_LOB.LOBMAXSIZE</code> . If you pass any other value, it must be less than or equal to the size of the BLOB.
dest_offset	(IN) Offset in characters in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB.  (OUT) The new offset in characters after the end of the write. This offset always points to the beginning of the first complete character after the end of the write.
src_offset	(IN) Offset in bytes in the source LOB for the start of the read.  (OUT) Offset in bytes in the source LOB right after the end of the read.
blob_csid	The character set ID of the source data
lang_context	(IN) Language context, such as shift status, for the current conversion.  (OUT) The language context at the time when the current conversion is done.  This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.

**Table 82–18 (Cont.) CONVERTTOCLOB Procedure Parameters**

Parameter	Description
warning	Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message.  Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, '?') is used in place of the inconvertible character. The return value of this error message is defined as the constant <code>warn_inconvertible_char</code> in the <code>DBMS_LOB</code> package.

## Usage Notes

### Preconditions

Before calling the `CONVERTTOCLOB` procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked before calling the `CONVERTTOCLOB` procedure. To lock the row, select the LOB using the `FOR UPDATE` clause of the `SELECT` statement.

### Constants and Defaults

All parameters are required. You must pass a variable for each `OUT` or `IN OUT` parameter. You must pass either a variable or a value for each `IN` parameter.

[Table 82–19](#) gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the `dbmslob.sql` package specification file.

**Table 82–19 DBMS\_LOB.CONVERTTOCLOB Typical Values**

Parameter	Value	Description
amount	<code>LOBMAXSIZE</code> (IN)	convert the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning
csid	<code>DEFAULT_CSID</code> (IN)	default CSID, use destination CSID
lang_context	<code>DEFAULT_LANG_CTX</code> (IN)	default language context
warning	<code>NO_WARNING</code> (OUT)  <code>WARN_INCONVERTIBLE_CHAR</code> (OUT)	no warning message, success  character in source cannot be properly converted

### General Notes

- You must specify the desired character set for the destination LOB in the `blob_csid` parameter. You can pass a zero value for `blob_csid`. When you do so, the database assumes that the desired character set is the same as the source LOB character set.

- You must specify the offsets for both the source and destination LOBs, and the number of characters to copy from the source LOB. The `amount` and `src_offset` values are in characters and the `dest_offset` is in bytes. To convert the entire LOB, you can specify `LOBMAXSIZE` for the `amount` parameter.
- `CONVERTTOCLOB` gets the source and/or destination LOBs as necessary prior to conversion and write of the data.

## Exceptions

**Table 82–20** *CONVERTTOCLOB Procedure Exceptions*

Exception	Description
<code>VALUE_ERROR</code>	Any of the input parameters are <code>NULL</code> or <code>INVALID</code> .
<code>INVALID_ARGVAL</code>	One or more of the following: <ul style="list-style-type: none"> <li>- <code>src_offset</code> or <code>dest_offset</code> &lt; 1.</li> <li>- <code>src_offset</code> or <code>dest_offset</code> &gt; <code>LOBMAXSIZE</code>.</li> <li>- <code>amount</code> &lt; 1.</li> <li>- <code>amount</code> &gt; <code>LOBMAXSIZE</code>.</li> </ul>

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information on using LOBs in application development

## COPY Procedures

This procedure copies all, or a part of, a source internal LOB to a destination internal LOB. You can specify the offsets for both the source and destination LOBs, and the number of bytes or characters to copy.

### Syntax

```
DBMS_LOB.COPY (
  dest_lob   IN OUT NOCOPY BLOB,
  src_lob    IN           BLOB,
  amount     IN           INTEGER,
  dest_offset IN           INTEGER := 1,
  src_offset IN           INTEGER := 1);
```

```
DBMS_LOB.COPY (
  dest_lob   IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  src_lob    IN           CLOB CHARACTER SET dest_lob%CHARSET,
  amount     IN           INTEGER,
  dest_offset IN           INTEGER := 1,
  src_offset IN           INTEGER := 1);
```

### Parameters

**Table 82–21 COPY Procedure Parameters**

Parameter	Description
dest_lob	LOB locator of the copy target.
src_lob	LOB locator of source for the copy.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to copy.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.
src_offset	Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.

### Exceptions

**Table 82–22 COPY Procedure Exceptions**

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or invalid.
INVALID_ARGVAL	Either: <ul style="list-style-type: none"> <li>- src_offset or dest_offset &lt; 1</li> <li>- src_offset or dest_offset &gt; LOBMAXSIZE</li> <li>- amount &lt; 1</li> <li>- amount &gt; LOBMAXSIZE</li> </ul>
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML slave
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on either LOB

## Usage Notes

- If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.
- It is not an error to specify an amount that exceeds the length of the data in the source LOB. Thus, you can specify a large amount to copy from the source LOB, which copies data from the `src_offset` to the end of the source LOB.
- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.
- Prior to copy, the source and destination LOBs are gotten, if they are currently archived. For a complete over-write, the destination LOB is not be retrieved.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## COPY\_DBFS\_LINK Procedures

This procedure copies the DBFS link in the source LOB to the destination LOB.

### Syntax

```
DBMS_LOB.COPY_DBFS_LINK (
  lob_loc_dst   IN OUT BLOB,
  lob_loc_src   IN     BLOB,
  flags         IN     PLS_INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

```
DBMS_LOB.COPY_DBFS_LINK (
  lob_loc_dst   IN OUT CLOB CHARACTER SET ANY_CS,
  lob_loc_src   IN     CLOB CHARACTER SET ANY_CS,
  flags         IN     PLS_INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

### Parameters

**Table 82–23** COPY\_DBFS\_LINK Procedure Parameters

Parameter	Description
lob_loc_dst	LOB to be made to reference the same storage data as lob_loc_src
lob_loc_src	LOB from which to copy the reference
flags	Options to COPY_DBFS_LINK: <ul style="list-style-type: none"> <li>▪ DBFS_LINK_NOCACHE specifies to only copy the DBFS Link</li> <li>▪ DBFS_LINK_CACHE specifies to copy the DBFS Link and read the data into the database LOB specified by lob_loc_dst so that the data is cached</li> </ul>

### Exceptions

**Table 82–24** COPY\_DBFS\_LINK Procedure Exceptions

Exception	Description
SECUREFILE_BADLOB	Either lob_loc_src or lob_loc_dst is not a SECUREFILE
INVALID_ARGVAL	lob_loc_src LOB has not been archived
ORA-01555	If the source LOB has been retrieved, never archived, or if the LOB has been migrated in and out (modified or not) since the locator was gotten.

## COPY\_FROM\_DBFS\_LINK

This procedure reads the archived data from the DBFS store and writes it back into the LOB.

### Syntax

```
DBMS_LOB.COPY_FROM_DBFS_LINK (  
  lob_loc      IN OUT BLOB);
```

```
DBMS_LOB.COPY_FROM_DBFS_LINK (  
  lob_loc      IN OUT CLOB CHARACTER SET ANY_CS);
```

### Parameters

**Table 82–25** *COPY\_FROM\_DBFS\_LINK Procedure Parameters*

Parameter	Description
lob_loc	LOB to be retrieved from the archive

### Usage Note

Note that COPY\_FROM\_DBFS\_LINK does not remove the underlying DBFS file.

### Exceptions

**Table 82–26** *COPY\_FROM\_DBFS\_LINK Procedure Exceptions*

Exception	Description
SECUREFILE_ BADLOB	lob_loc is not a SECUREFILE
ORA-01555	If the LOB has already been retrieved and has been modified since retrieval, if the LOB has been migrated in and out (modified or not) since the locator was retrieved



## CREATETEMPORARY Procedures

This procedure creates a temporary BLOB or CLOB and its corresponding index in your default temporary tablespace.

### Syntax

```
DBMS_LOB.CREATETEMPORARY (
  lob_loc IN OUT NOCOPY BLOB,
  cache   IN           BOOLEAN,
  dur     IN           PLS_INTEGER := DBMS_LOB.SESSION);
```

```
DBMS_LOB.CREATETEMPORARY (
  lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  cache   IN           BOOLEAN,
  dur     IN           PLS_INTEGER := 10);
```

### Parameters

**Table 82–27 CREATETEMPORARY Procedure Parameters**

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
cache	Specifies if LOB should be read into buffer cache or not.
dur	1 of 3 predefined duration values (SESSION, TRANSACTION, or CALL) which specifies a hint as to whether the temporary LOB is cleaned up at the end of the session, transaction or call.  If dur is omitted, then the session duration is used.

#### See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure
- *Oracle Database PL/SQL Language Reference* for more information about NOCOPY and passing temporary lob as parameters

## DBFS\_LINK\_GENERATE\_PATH Functions

This subprogram returns a unique file path name for use in creating a DBFS Link.

### Syntax

```
DBMS_LOB.DBFS_LINK_GENERATE_PATH (  
  lob_loc      IN BLOB,  
  storage_dir  IN VARCHAR2)  
RETURN VARCHAR2;  
  
DBMS_LOB.DBFS_LINK_GENERATE_PATH (  
  lob_loc      IN CLOB CHARACTER SET ANY_CS,  
  storage_dir  IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 82–28** *DBFS\_LINK\_GENERATE\_PATH Function Parameters*

Parameter	Description
lob_loc	LOB to be retrieved from DBFS
storage_dir	DBFS directory that will be the parent directory of the file

### Exceptions

**Table 82–29** *DBFS\_LINK\_GENERATE\_PATH Function Exceptions*

Exception	Description
SECUREFILE_ WRONGTYPE	lob_loc is not a SECUREFILE

## ERASE Procedures

This procedure erases an entire internal LOB or part of an internal LOB.

### Syntax

```
DBMS_LOB.ERASE (
  lob_loc          IN OUT NOCOPY BLOB,
  amount          IN OUT NOCOPY INTEGER,
  offset          IN              INTEGER := 1);

DBMS_LOB.ERASE (
  lob_loc          IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  amount          IN OUT NOCOPY INTEGER,
  offset          IN              INTEGER := 1);
```

### Parameters

**Table 82–30 ERASE Procedure Parameters**

Parameter	Description
lob_loc	Locator for the LOB to be erased. For more information, see <a href="#">Operational Notes</a> .
amount	Number of bytes (for BLOBs or BFILES) or characters (for CLOBs or NCLOBs) to be erased.
offset	Absolute offset (origin: 1) from the beginning of the LOB in bytes (for BLOBs) or characters (CLOBs).

### Usage Notes

- When data is erased from the middle of a LOB, zero-byte fillers or spaces are written for BLOBs or CLOBs respectively.
- The actual number of bytes or characters erased can differ from the number you specified in the `amount` parameter if the end of the LOB value is reached before erasing the specified number. The actual number of characters or bytes erased is returned in the `amount` parameter.
- ERASE gets the LOB if it is archived, unless the erase covers the entire LOB.

---

**Note:** The length of the LOB is not decreased when a section of the LOB is erased. To decrease the length of the LOB value, see the "[TRIM Procedures](#)" on page 82-79.

---

### Exceptions

**Table 82–31 ERASE Procedure Exceptions**

Exception	Description
VALUE_ERROR	Any input parameter is NULL.
INVALID_ARGVAL	Either: <ul style="list-style-type: none"> <li>- amount &lt; 1 or amount &gt; LOBMAXSIZE</li> <li>- offset &lt; 1 or offset &gt; LOBMAXSIZE</li> </ul>

**Table 82–31 (Cont.) ERASE Procedure Exceptions**

Exception	Description
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML slave
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

## Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

### See Also:

- ["TRIM Procedures"](#) on page 82-79
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## FILECLOSE Procedure

This procedure closes a BFILE that has already been opened through the input locator.

---

**Note:** The database has only read-only access to BFILES. This means that BFILES cannot be written through the database.

---

### Syntax

```
DBMS_LOB.FILECLOSE (
    file_loc IN OUT NOCOPY BFILE);
```

### Parameters

**Table 82–32 FILECLOSE Procedure Parameters**

Parameter	Description
file_loc	Locator for the BFILE to be closed.

### Exceptions

**Table 82–33 FILECLOSE Procedure Exceptions**

Exception	Description
VALUE_ERROR	NULL input value for file_loc.
UNOPENED_FILE	File was not opened with the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

#### See Also:

- ["FILEOPEN Procedure"](#) on page 82-42
- ["FILECLOSEALL Procedure"](#) on page 82-38
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## FILECLOSEALL Procedure

This procedure closes all BFILEs opened in the session.

### Syntax

```
DBMS_LOB.FILECLOSEALL;
```

### Exceptions

**Table 82–34 FILECLOSEALL Procedure Exception**

Exception	Description
UNOPENED_FILE	No file has been opened in the session.

**See Also:**

- ["FILEOPEN Procedure"](#) on page 82-42
- ["FILECLOSE Procedure"](#) on page 82-37
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## FILEEXISTS Function

This function finds out if a specified BFILE locator points to a file that actually exists on the server's file system.

### Syntax

```
DBMS_LOB.FILEEXISTS (
    file_loc    IN    BFILE)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–35 FILEEXISTS Function Parameter**

Parameter	Description
file_loc	Locator for the BFILE.

### Return Values

**Table 82–36 FILEEXISTS Function Return Values**

Return	Description
0	Physical file does not exist.
1	Physical file exists.

### Exceptions

**Table 82–37 FILEEXISTS Function Exceptions**

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

#### See Also:

- ["FILEISOPEN Function"](#) on page 82-41.
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## FILEGETNAME Procedure

This procedure determines the directory object and filename, given a BFILE locator. This function only indicates the directory object name and filename assigned to the locator, not if the physical file or directory actually exists.

The maximum constraint values for the `dir_alias` buffer is 30, and for the entire path name, it is 2000.

### Syntax

```
DBMS_LOB.FILEGETNAME (  
    file_loc   IN    BFILE,  
    dir_alias  OUT   VARCHAR2,  
    filename  OUT   VARCHAR2);
```

### Parameters

**Table 82–38 FILEGETNAME Procedure Parameters**

Parameter	Description
<code>file_loc</code>	Locator for the BFILE
<code>dir_alias</code>	Directory object name
<code>filename</code>	Name of the BFILE

### Exceptions

**Table 82–39 FILEGETNAME Procedure Exceptions**

Exception	Description
<code>VALUE_ERROR</code>	Any of the input parameters are NULL or INVALID.
<code>INVALID_ARGVAL</code>	<code>dir_alias</code> or <code>filename</code> are NULL.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure



## FILEISOPEN Function

This function finds out whether a BFILE was opened with the specified FILE locator.

### Syntax

```
DBMS_LOB.FILEISOPEN (
    file_loc IN BFILE)
RETURN INTEGER;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(fileisopen, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–40 FILEISOPEN Function Parameter**

Parameter	Description
file_loc	Locator for the BFILE.

### Return Values

INTEGER: 0 = file is not open, 1 = file is open

### Usage Notes

If the input FILE locator was never passed to the FILEOPEN procedure, then the file is considered not to be opened by this locator. However, a different locator may have this file open. In other words, openness is associated with a specific locator.

### Exceptions

**Table 82–41 FILEISOPEN Function Exceptions**

Exception	Description
NOEXIST_ DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_ DIRECTORY	Directory has been invalidated after the file was opened.

#### See Also:

- ["FILEEXISTS Function"](#) on page 82-39
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## FILEOPEN Procedure

This procedure opens a BFILE for read-only access. BFILE data may not be written through the database.

### Syntax

```
DBMS_LOB.FILEOPEN (
    file_loc   IN OUT NOCOPY BFILE,
    open_mode  IN             BINARY_INTEGER := file_readonly);
```

### Parameters

**Table 82–42 FILEOPEN Procedure Parameters**

Parameter	Description
file_loc	Locator for the BFILE.
open_mode	File access is read-only.

### Exceptions

**Table 82–43 FILEOPEN Procedure Exceptions**

Exception	Description
VALUE_ERROR	file_loc or open_mode is NULL.
INVALID_ARGVAL	open_mode is not equal to FILE_READONLY.
OPEN_TOOMANY	Number of open files in the session exceeds session_max_open_files.
NOEXIST_DIRECTORY	Directory associated with file_loc does not exist.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

#### See Also:

- ["FILECLOSE Procedure"](#) on page 82-37
- ["FILECLOSEALL Procedure"](#) on page 82-38
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## FRAGMENT\_DELETE Procedure

This procedure deletes the data at the specified offset for the specified length from the LOB without having to rewrite all the data in the LOB following the specified offset.

### Syntax

```
DBMS_LOB.FRAGMENT_DELETE (
  lob_loc      IN OUT NOCOPY BLOB,
  amount       IN           INTEGER,
  offset       IN           INTEGER);

DBMS_LOB.FRAGMENT_DELETE (
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  amount       IN           INTEGER,
  offset       IN           INTEGER);
```

### Parameters

**Table 82–44** *FRAGMENT\_DELETE Procedure Parameters*

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to be removed from the LOB
offset	Offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to begin the deletion

### Exceptions

**Table 82–45** *FRAGMENT\_DELETE Procedure Exceptions*

Exception	Description
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled
SECUREFILE_BADLOB	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call
SECUREFILE_OUTOFBOUNDS	Attempted to perform a FRAGMENT_* operation past LOB end

## FRAGMENT\_INSERT Procedures

This procedure inserts the specified data (limited to 32K) into the LOB at the specified offset.

### Syntax

```
DBMS_LOB.FRAGMENT_INSERT (
  lob_loc      IN OUT NOCOPY BLOB,
  amount       IN           INTEGER,
  offset       IN           INTEGER,
  buffer       IN           RAW);

DBMS_LOB.FRAGMENT_INSERT (
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  amount       IN           INTEGER,
  offset       IN           INTEGER,
  buffer       IN           VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

### Parameters

**Table 82–46** *FRAGMENT\_INSERT Procedure Parameters*

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to be inserted into the LOB
offset	Offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to begin the insertion
buffer	Data to insert into the LOB

### Exceptions

**Table 82–47** *FRAGMENT\_INSERT Procedure Exceptions*

Exception	Description
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled
SECUREFILE_BADLOB	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call
SECUREFILE_OUTOFBOUNDS	Attempted to perform a FRAGMENT_* operation past LOB end

### Usage Notes

FRAGMENT\_INSERT gets the LOB, if necessary, before performing operations on the LOB.

## FRAGMENT\_MOVE Procedure

This procedure moves the amount of bytes (BLOB) or characters (CLOB/NCLOB) from the specified offset to the new offset specified.

### Syntax

```
DBMS_LOB.FRAGMENT_MOVE (
  lob_loc      IN OUT NOCOPY BLOB,
  amount       IN           INTEGER,
  src_offset   IN           INTEGER,
  dest_offset  IN           INTEGER);
```

```
DBMS_LOB.FRAGMENT_MOVE (
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  amount       IN           INTEGER,
  src_offset   IN           INTEGER,
  dest_offset  IN           INTEGER);
```

### Parameters

**Table 82–48 FRAGMENT\_MOVE Procedure Parameters**

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to be moved in the LOB
src_offset	Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to put the data
dest_offset	Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to remove the data

### Exceptions

**Table 82–49 FRAGMENT\_MOVE Procedure Exceptions**

Exception	Description
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled
SECUREFILE_BADLOB	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call
SECUREFILE_OUTOFBOUNDS	Attempted to perform a FRAGMENT_* operation past LOB end

### Usage Notes

- All offsets are pre-move offsets.
- Offsets of more than 1 past the end of the LOB are not permitted.
- FRAGMENT\_MOVE gets the LOB, if necessary, before performing operations on the LOB.

## FRAGMENT\_REPLACE Procedures

This procedure replaces the data at the specified offset with the specified data (not to exceed 32k).

### Syntax

```
DBMS_LOB.FRAGMENT_REPLACE (
  lob_loc      IN OUT NOCOPY BLOB,
  old_amount   IN             INTEGER,
  new_amount   IN             INTEGER,
  offset       IN             INTEGER,
  buffer       IN             RAW);

DBMS_LOB.FRAGMENT_REPLACE (
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  old_amount   IN             INTEGER,
  new_amount   IN             INTEGER,
  offset       IN             INTEGER,
  buffer       IN             VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

### Parameters

**Table 82–50 FRAGMENT\_REPLACE Function Parameters**

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
old_amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to be replaced in the LOB
new_amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to written to the LOB
offset	Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to put the data
buffer	Data to insert into the LOB

### Exceptions

**Table 82–51 FRAGMENT\_REPLACE Procedure Exceptions**

Exception	Description
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled
SECUREFILE_BADLOB	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call
SECUREFILE_OUTOFBOUNDS	Attempted to perform a FRAGMENT_* operation past LOB end

### Usage Notes

- Invoking this procedure is equivalent to deleting the old amount of bytes/characters at offset and then inserting the new amount of bytes/characters at offset.

- `FRAGMENT_REPLACE` gets the LOB, if necessary, before performing operations on the LOB.

## FREETEMPORARY Procedures

This procedure frees the temporary BLOB or CLOB in the default temporary tablespace.

### Syntax

```
DBMS_LOB.FREETEMPORARY (
  lob_loc IN OUT NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (
  lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);
```

### Parameters

**Table 82–52** *FREETEMPORARY Procedure Parameters*

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .

### Usage Notes

- When a new temporary LOB is created, and there is currently no temporary LOB in use with the same duration (session, transaction, call), a new temporary LOB segment is created. When the temporary LOB is freed, the space it consumed is released to the temporary segment. If there are no other temporary LOBs for the same duration, the temporary segment is also freed.
- After the call to `FREETEMPORARY`, the LOB locator that was freed is marked as invalid.
- If an invalid LOB locator is assigned to another LOB locator using `OCILobLocatorAssign` in OCI or through an assignment operation in PL/SQL, then the target of the assignment is also freed and marked as invalid.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure



## GET\_DBFS\_LINK Functions

This function returns the DBFS Link path associated with the specified SecureFile.

### Syntax

```
DBMS_LOB.GET_DBFS_LINK (
  lob_loc          IN      BLOB,
  storage_path     OUT VARCHA2(DBFS_LINK_PATH_MAX_SIZE) ,
  lob_length       OUT NUMBER);
```

```
DBMS_LOB.GET_DBFS_LINK (
  lob_loc          IN      CLOB CHARACTER SET ANY_CS,
  storage_path     OUT VARCHA2(DBFS_LINK_PATH_MAX_SIZE) ,
  lob_length       OUT NUMBER);
```

### Parameters

**Table 82–53** *GET\_DBFS\_LINK Function Parameters*

Parameter	Description
lob_loc	LOB to be retrieved from DBFS
storage_path	Path where the LOB is stored in DBFS
lob_length	LOB length at the time of write to DBFS

### Return Values

The Archive ID

### Exceptions

**Table 82–54** *GET\_DBFS\_LINK Function Exceptions*

Exception	Description
SECUREFILE_BADLOB	lob_loc is not a SECUREFILE
ORA-01555	If the LOB has already been retrieved and has been modified since retrieval, if the LOB has been migrated in and out (modified or not) since the locator was retrieved

## GET\_DBFS\_LINK\_STATE Procedures

This procedure retrieves the current DBFS Link state of the specified SecureFile.

### Syntax

```
DBMS_LOB.GET_DBFS_LINK_STATE (
  lob_loc      IN BLOB,
  storage_path OUT VARCHAR2 (DBFS_LINK_PATH_MAX_SIZE),
  state        OUT NUMBER,
  cached       OUT BOOLEAN);
```

```
DBMS_LOB.GET_DBFS_LINK_STATE (
  lob_loc      IN CLOB CHARACTER SET ANY_CS,
  storage_path OUT VARCHAR2 (DBFS_LINK_PATH_MAX_SIZE),
  state        OUT NUMBER,
  cached       OUT BOOLEAN);
```

### Parameters

**Table 82–55** GET\_DBFS\_LINK\_STATE Procedure Parameters

Parameter	Description
lob_loc	LOB to be retrieved from the archive
storage_path	Path where the LOB is stored in DBFS
state	One of DBFS_LINK_NEVER, DBFS_LINK_NO or DBFS_LINK_YES
cached	If the LOB is archived and the data was specified to be cached on put

### Exceptions

**Table 82–56** GET\_DBFS\_LINK\_STATE Procedure Exceptions

Exception	Description
SECUREFILE_BADLOB	lob_loc is not a SECUREFILE

### Usage Notes

- If the LOB has never been archived, state is set to DBMS\_LOB.DBFS\_LINK\_NEVER. If the LOB has been archived, state is set to DBMS\_LOB.DBFS\_LINK\_YES. If the LOB has been previously retrieved from the archive, state is set to DBFS\_LINK\_NO.
- If the LOB was archived, but the data was left in the RDBMS, cached is set to TRUE. Cached is set to FALSE if the data was removed after the put, and NULL if state is DBMS\_LOB.DBFS\_LINK\_NEVER.

## GETCONTENTTYPE Functions

This procedure returns the content type string previously set by means of the [SETCONTENTTYPE Procedure](#).

### Syntax

```
DBMS_LOB.GETCONTENTTYPE (
  lob_loc IN BLOB)
RETURN VARCHAR2;

DBMS_LOB.GETCONTENTTYPE (
  lob_loc IN CLOB CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(getcontenttype, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–57** *GETCONTENTTYPE Function Parameters*

Parameter	Description
lob_loc	LOB whose content type is to be retrieved

### Return Values

The returned content type.

### Exceptions

**Table 82–58** *GETCONTENTTYPE Function Exceptions*

Exception	Description
SECUREFILE_BADLOB	lob_loc is not a SECUREFILE

## GET\_STORAGE\_LIMIT Function

This function returns the LOB storage limit for the specified LOB.

### Syntax

```
DBMS_LOB.GET_STORAGE_LIMIT (
    lob_loc IN CLOB CHARACTER SET ANY_CS)
RETURN INTEGER;

DBMS_LOB.GET_STORAGE_LIMIT (
    lob_loc IN BLOB)
RETURN INTEGER;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(get_storage_limit, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–59** GET\_STORAGE\_LIMIT Function Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .

### Return Value

The value returned from this function is the maximum allowable size for specified LOB locator. For BLOBs, the return value depends on the block size of the tablespace the LOB resides in and is calculated as  $(2^{32})-1$  (4294967295) times the block size of the tablespace. For CLOBs/NCLOBs, the value returned is the  $(2^{32})-1$  (4294967295) times the block size of the tablespace divided by the character width of the CLOB/NCLOB.

### Usage

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for details on LOB storage limits

## GETCHUNKSIZE Functions

When creating the table, you can specify the chunking factor, a multiple of tablespace blocks in bytes. This corresponds to the chunk size used by the LOB data layer when accessing or modifying the LOB value. Part of the chunk is used to store system-related information, and the rest stores the LOB value.

This function returns the amount of space used in the LOB chunk to store the LOB value.

### Syntax

```
DBMS_LOB.GETCHUNKSIZE (
    lob_loc IN BLOB)
    RETURN INTEGER;

DBMS_LOB.GETCHUNKSIZE (
    lob_loc IN CLOB CHARACTER SET ANY_CS)
    RETURN INTEGER;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(getchunksizesize, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–60** GETCHUNKSIZE Function Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .

### Return Values

The return value is a usable chunk size in bytes.

### Usage Notes

- With regard to basic LOB files, performance is improved if you enter read/write requests using a multiple of this chunk size. For writes, there is an added benefit, because LOB chunks are versioned, and if all writes are done on a chunk basis, then no extra or excess versioning is done or duplicated. You could batch up the WRITE until you have enough for a chunk, instead of issuing several WRITE calls for the same chunk.

These tactics of performance improvement do not apply to SecureFiles.

- Note that chunk size is independent of LOB type (BLOB, CLOB, NCLOB, Unicode or other character set).

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## Exceptions

**Table 82–61** *GETCHUNKSIZE Procedure Exceptions*

<b>Exception</b>	<b>Description</b>
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

## GETLENGTH Functions

This function gets the length of the specified LOB. The length in bytes or characters is returned.

The length returned for a BFILE includes the EOF, if it exists. Any 0-byte or space filler in the LOB caused by previous ERASE or WRITE operations is also included in the length count. The length of an empty internal LOB is 0.

### Syntax

```
DBMS_LOB.GETLENGTH (
  lob_loc   IN BLOB)
RETURN INTEGER;

DBMS_LOB.GETLENGTH (
  lob_loc   IN CLOB CHARACTER SET ANY_CS)
RETURN INTEGER;

DBMS_LOB.GETLENGTH (
  file_loc  IN BFILE)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);
```

### Parameters

**Table 82–62** GETLENGTH Function Parameter

Parameter	Description
file_loc	The file locator for the LOB whose length is to be returned.

### Return Values

The length of the LOB in bytes or characters as an INTEGER. NULL is returned if the input LOB is NULL or if the input lob\_loc is NULL. An error is returned in the following cases for BFILES:

- lob\_loc does not have the necessary directory and operating system privileges
- lob\_loc cannot be read because of an operating system read error

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

### Exceptions

**Table 82–63** GETLENGTH Procedure Exceptions

Exception	Description
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

## GETOPTIONS Functions

This function obtains settings corresponding to the `option_type` field for a particular LOB.

### Syntax

```
DBMS_LOB.GETOPTIONS (
  lob_loc          IN      BLOB,
  option_types     IN      PLS_INTEGER)
RETURN PLS_INTEGER;

DBMS_LOB.GETOPTIONS (
  lob_loc          IN      CLOB CHARACTER SET ANY_CS,
  option_types     IN      PLS_INTEGER)
RETURN PLS_INTEGER;
```

### Parameters

**Table 82–64** *GETOPTIONS Function Parameter*

Parameter	Description
<code>lob_loc</code>	Locator for the LOB to be examined. For more information, see <a href="#">Operational Notes</a> .
<code>option_type</code>	See <a href="#">DBMS_LOB Constants - Option Types</a> on page 82-5

### Return Values

The return values are a combination of `COMPRESS_ON`, `ENCRYPT_ON` and `DEDUPLICATE_ON` (see [DBMS\\_LOB Constants - Option Values](#) on page 82-5) depending on which option types (see [DBMS\\_LOB Constants - Option Types](#) on page 82-5) are passed in.

### Exceptions

**Table 82–65** *GETOPTIONS Procedure Exceptions*

Exception	Description
<code>INVALID_ARGVAL</code>	A parameter value was invalid
<code>QUERY_WRITE</code>	Cannot perform operation during a query
<code>BUFFERING_ENABLED</code>	Cannot perform operation with LOB buffering enabled
<code>SECUREFILE_BADLOB</code>	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call

### Usage Notes

You cannot turn compression or deduplication on or off for a SecureFile column that does not have those features on. The `GetOptions` Functions and [SETOPTIONS Procedures](#) work on individual SecureFiles. You can turn off a feature on a particular SecureFile and turn on a feature that has already been turned off by `SetOptions`, but you cannot turn on an option that has not been given to the SecureFile when the table was created.



## INSTR Functions

This function returns the matching position of the *n*th occurrence of the pattern in the LOB, starting from the offset you specify.

### Syntax

```
DBMS_LOB.INSTR (
  lob_loc    IN    BLOB,
  pattern    IN    RAW,
  offset     IN    INTEGER := 1,
  nth        IN    INTEGER := 1)
RETURN INTEGER;
```

```
DBMS_LOB.INSTR (
  lob_loc    IN    CLOB          CHARACTER SET ANY_CS,
  pattern    IN    VARCHAR2     CHARACTER SET lob_loc%CHARSET,
  offset     IN    INTEGER := 1,
  nth        IN    INTEGER := 1)
RETURN INTEGER;
```

```
DBMS_LOB.INSTR (
  file_loc   IN    BFILE,
  pattern    IN    RAW,
  offset     IN    INTEGER := 1,
  nth        IN    INTEGER := 1)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);
```

### Parameters

**Table 82–66 INSTR Function Parameters**

Parameter	Description
lob_loc	Locator for the LOB to be examined. For more information, see <a href="#">Operational Notes</a> .
file_loc	The file locator for the LOB to be examined.
pattern	Pattern to be tested for. The pattern is a group of RAW bytes for BLOBs, and a character string (VARCHAR2) for CLOBs. The maximum size of the pattern is 16383 bytes.
offset	Absolute offset in bytes (BLOBs) or characters (CLOBs) at which the pattern matching is to start. (origin: 1)
nth	Occurrence number, starting at 1.

### Return Values

**Table 82–67 INSTR Function Return Values**

Return	Description
INTEGER	Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.

**Table 82–67 (Cont.) INSTR Function Return Values**

Return	Description
NULL	Either: -any one or more of the IN parameters was NULL or INVALID. -offset < 1 or offset > LOBMAXSIZE. -nth < 1. -nth > LOBMAXSIZE.

## Usage Notes

The form of the VARCHAR2 buffer (the `pattern` parameter) must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For BFILES, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

Operations that accept RAW or VARCHAR2 parameters for pattern matching, such as INSTR, do not support regular expressions or special matching characters (as in the case of SQL LIKE) in the pattern parameter or substrings.

## Exceptions

**Table 82–68 INSTR Function Exceptions for BFILES**

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

### See Also:

- ["SUBSTR Functions"](#) on page 82-77
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## ISOPEN Functions

This function checks to see if the LOB was already opened using the input locator. This subprogram is for internal and external LOBs.

### Syntax

```
DBMS_LOB.ISOPEN (
  lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
  lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
  file_loc IN BFILE)
  RETURN INTEGER;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(isopen, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–69 ISOPEN Function Parameters**

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
file_loc	File locator.

### Return Values

The return value is 1 if the LOB is open, 0 otherwise.

### Usage Notes

For BFILES, openness is associated with the locator. If the input locator was never passed to OPEN, the BFILE is not considered to be opened by this locator. However, a different locator may have opened the BFILE. More than one OPEN can be performed on the same BFILE using different locators.

For internal LOBs, openness is associated with the LOB, not with the locator. If locator1 opened the LOB, then locator2 also sees the LOB as open. For internal LOBs, ISOPEN requires a round-trip, because it checks the state on the server to see if the LOB is indeed open.

For external LOBs (BFILES), ISOPEN also requires a round-trip, because that's where the state is kept.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## ISTEMPORARY Functions

This function determines whether a LOB instance is temporary.

### Syntax

```
DBMS_LOB.ISTEMPORARY (
    lob_loc IN BLOB)
RETURN INTEGER;

DBMS_LOB.ISTEMPORARY (
    lob_loc IN CLOB CHARACTER SET ANY_CS)
RETURN INTEGER;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 82–70** *ISTEMPORARY Procedure Parameters*

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .

### Return Values

The return value is 1 if the LOB is temporary and exists; 0 if the LOB is not temporary or does not exist; NULL if the given locator is NULL.

### Usage Notes

When you free a Temporary LOB with `FREETEMPORARY`, the LOB locator is not set to NULL. Consequently, `ISTEMPORARY` will return 0 for a locator that has been freed but not explicitly reset to NULL.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## LOADBLOBFROMFILE Procedure

This procedure loads data from BFILE to internal BLOB. This achieves the same outcome as LOADFROMFILE, and returns the new offsets.

### Syntax

```
DBMS_LOB.LOADBLOBFROMFILE (
  dest_lob    IN OUT NOCOPY BLOB,
  src_bfile   IN          BFILE,
  amount      IN          INTEGER,
  dest_offset IN OUT     INTEGER,
  src_offset  IN OUT     INTEGER);
```

### Parameters

**Table 82–71** *LOADBLOBFROMFILE Procedure Parameters*

Parameter	Description
dest_lob	BLOB locator of the target for the load.
src_bfile	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE. You can also use DBMS_LOB.LOBMAXSIZE to load until the end of the BFILE.
dest_offset	(IN) Offset in bytes in the destination BLOB (origin: 1) for the start of the write. (OUT) New offset in bytes in the destination BLOB right after the end of this write, which is also where the next write should begin.
src_offset	(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.

### Usage Notes

- You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src\_offset, because they refer to the BFILE, are in terms of bytes, and the dest\_offset is in bytes for BLOBs.
- If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.
- There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is LOBMAXSIZE which you can specify to continue loading until the end of the BFILE is reached).
- It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely

affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

- LOADFROMFILE gets the destination LOB prior to the load unless the load covers the entire LOB.

### Constants and Defaults

There is no easy way to omit parameters. You must either declare a variable for IN/OUT parameter or provide a default value for the IN parameter. Here is a summary of the constants and the defaults that can be used.

**Table 82–72 Suggested Values of the Parameter**

Parameter	Default Value	Description
amount	DBMS_LOB.LOBMAXSIZE (IN)	Load the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning

Constants defined in DBMSLOB.SQL

```
lobmaxsize          CONSTANT INTEGER          := DBMS_LOB.LOBMAXSIZE;
```

### Exceptions

**Table 82–73 LOADBLOBFROMFILE Procedure Exceptions**

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: <ul style="list-style-type: none"> <li>- src_offset or dest_offset &lt; 1.</li> <li>- src_offset or dest_offset &gt; LOBMAXSIZE.</li> <li>- amount &lt; 1.</li> <li>- amount &gt; LOBMAXSIZE.</li> </ul>
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the BLOB

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## LOADCLOBFROMFILE Procedure

This procedure loads data from a BFILE to an internal CLOB/NCLOB with necessary character set conversion and returns the new offsets.

### Syntax

```
DBMS_LOB.LOADCLOBFROMFILE (
  dest_lob      IN OUT NOCOPY  NOCOPY CLOB CHARACTER SET ANY_CS,
  src_bfile     IN              BFILE,
  amount        IN              INTEGER,
  dest_offset   IN OUT         INTEGER,
  src_offset    IN OUT         INTEGER,
  bfile_csid    IN              NUMBER,
  lang_context  IN OUT         INTEGER,
  warning       OUT            INTEGER);
```

### Parameters

**Table 82–74** *LOADCLOBFROMFILE Procedure Parameters*

Parameter	Description
dest_lob	CLOB/NCLOB locator of the target for the load.
src_bfile	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE. Use DBMS_LOB.LOBBMAXSIZE of load until the end of the BFILE.
dest_offset	(IN) Offset in characters in the destination CLOB (origin: 1) for the start of the write. (OUT) The new offset in characters right after the end of this load, which is also where the next load should start. It always points to the beginning of the first complete character after the end of load. If the last character is not complete, offset goes back to the beginning of the partial character.
src_offset	(IN) Offset in bytes in the source BFILE (origin: 1) for the start of the read. (OUT) Offset in bytes in the source BFILE right after the end of this read, which is also where the next read should begin.
bfile_csid	Character set id of the source (BFILE) file.
lang_context	(IN) Language context, such as shift status, for the current load.  (OUT) The language context at the time when the current load stopped, and what the next load should be using if continuing loading from the same source. This information is returned to the user so that they can use it for the continuous load without losing or misinterpreting any source data. For the very first load or if do not care, simply use the default 0. The details of this language context is hidden from the user. One does not need to know what it is or what's in it in order to make the call

**Table 82–74 (Cont.) LOADCLOBFROMFILE Procedure Parameters**

Parameter	Description
warning	(OUT) Warning message. This indicates something abnormal happened during the loading. It may or may not be caused by the user's mistake. The loading is completed as required, and it's up to the user to check the warning message. Currently, the only possible warning is the inconvertible character. This happens when the character in the source cannot be properly converted to a character in destination, and the default replacement character (for example, '?') is used in place. The message is defined the constant value <code>DBMS_LOB.WARN_INCONVERTIBLE_CHAR</code> .

## Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and `src_offset`, because they refer to the BFILE, are in terms of bytes, and the `dest_offset` is in characters for CLOBs.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination CLOB. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE (unless the amount specified is `LOBMAXSIZE` which you can specify to continue loading until the end of the BFILE is reached).

Note the following requirements:

- The destination character set is always the same as the database character set in the case of CLOB and national character set in the case of NCLOB.
- `csid=0` indicates the default behavior that uses database `csid` for CLOB and national `csid` for NCLOB in the place of source `csid`. Conversion is still necessary if it is of varying width
- It is not mandatory that you wrap the LOB operation inside the OPEN/CLOSE operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the OPEN/CLOSE, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

The source BFILE can contain data in the Unicode character set. The Unicode standard defines many encoding schemes that provide mappings from Unicode characters to sequences of bytes. [Table 82–75, "Supported Unicode Encoding Schemes"](#) lists Unicode encodings schemes supported by this subprogram.

**Table 82–75 Supported Unicode Encoding Schemes**

Encoding Scheme	Oracle Name	bfile_csid Value
UTF-8	AL32UTF8	873
UTF-16BE	AL16UTF16	2000



**Table 82–75 (Cont.) Supported Unicode Encoding Schemes**

Encoding Scheme	Oracle Name	bfile_csid Value
UTF-16LE	AL16UTF16LE	2002
CESU-8	UTF8	871
UTF-EBCDIC	UTFE	872
UTF-16	UTF16	1000

All three UTF-16 encoding schemes encode Unicode characters as 2-byte unsigned integers. Integers can be stored in big-endian or in little-endian byte order. The UTF-16BE encoding scheme defines big-endian data. The UTF-16LE scheme defines little-endian data. The UTF-16 scheme requires that the source BFILE contains the Byte Order Mark (BOM) character in the first two bytes to define the byte order. The BOM code is 0xFEFF. If the code is stored as {0xFE, 0xFF}, the data is interpreted as big-endian. If it is stored as {0xFF, 0xFE}, the data is interpreted as little-endian.

In UTF-8 and in CESU-8 encodings the Byte Order Mark is stored as {0xEF, 0xBB, 0xBF}. With any of the Unicode encodings, the corresponding BOM sequence at the beginning of the file is recognized and not loaded into the destination LOB.

## Constants

Here is a summary of the constants and the suggested values that can be used.

**Table 82–76 Suggested Values of the Parameter**

Parameter	Suggested Value	Description
amount	DBMS_LOB.LOBMAXSIZE (IN)	Load the entire file
dest_offset	1 (IN)	start from the beginning
src_offset	1 (IN)	start from the beginning
csid	0 (IN)	default csid, use destination csid
lang_context	0 (IN)	default language context
warning	0 (OUT)	no warning message, everything is ok

Constants defined in DBMSLOB.SQL

lobmaxsize	CONSTANT INTEGER	:= 18446744073709551615;
warn_inconvertible_char	CONSTANT INTEGER	:= 1;
default_csid	CONSTANT INTEGER	:= 0;
default_lang_ctx	CONSTANT INTEGER	:= 0;
no_warning	CONSTANT INTEGER	:= 0;

## Exceptions

**Table 82–77 LOADCLOBFROMFILE Procedure Exceptions**

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.

**Table 82–77 (Cont.) LOADCLOBFROMFILE Procedure Exceptions**

Exception	Description
INVALID_ARGVAL	Either: <ul style="list-style-type: none"><li>- src_offset or dest_offset &lt; 1.</li><li>- src_offset or dest_offset &gt; LOBMAXSIZE.</li><li>- amount &lt; 1.</li><li>- amount &gt; LOBMAXSIZE.</li></ul>
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the CLOB

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## LOADFROMFILE Procedure

This procedure copies all, or a part of, a source external LOB (BFILE) to a destination internal LOB.

### Syntax

```
DBMS_LOB.LOADFROMFILE (
  dest_lob   IN OUT NOCOPY BLOB,
  src_file   IN           BFILE,
  amount     IN           INTEGER,
  dest_offset IN           INTEGER := 1,
  src_offset IN           INTEGER := 1);
```

### Parameters

**Table 82–78** *LOADFROMFILE Procedure Parameters*

Parameter	Description
dest_lob	LOB locator of the target for the load.
src_file	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the load.
src_offset	Offset in bytes in the source BFILE (origin: 1) for the start of the load.

### Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The `amount` and `src_offset`, because they refer to the BFILE, are in terms of bytes, and the `dest_offset` is either in bytes or characters for BLOBs and CLOBs respectively.

---

**Note:** The input BFILE must have been opened prior to using this procedure. No character set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this.

---

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE.

---



---

**Note:** If the character set is varying width, UTF-8 for example, the LOB value is stored in the fixed-width UCS2 format. Therefore, if you are using `DBMS_LOB.LOADFROMFILE`, the data in the BFILE should be in the UCS2 character set instead of the UTF-8 character set. However, you should use `sql*loader` instead of `LOADFROMFILE` to load data into a CLOB or NCLOB because `sql*loader` provides the necessary character set conversions.

---



---

It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.

## Exceptions

**Table 82–79** *LOADFROMFILE Procedure Exceptions*

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.
INVALID_ARGVAL	Either: <ul style="list-style-type: none"> <li>- <code>src_offset</code> or <code>dest_offset</code> &lt; 1.</li> <li>- <code>src_offset</code> or <code>dest_offset</code> &gt; LOBMAXSIZE.</li> <li>- <code>amount</code> &lt; 1.</li> <li>- <code>amount</code> &gt; LOBMAXSIZE.</li> </ul>

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## MOVE\_TO\_DBFS\_LINK Procedures

This procedure writes the specified SecureFile data to the DBFS store.

### Syntax

```
DBMS_LOB.MOVE_TO_DBFS_LINK (
  lob_loc      IN OUT BLOB,
  storage_path IN    VARCHAR2(dbfs_link_path_max_size),
  flags        IN    BINARY_INTEGER DEFAULT DBFS_LINK_NOCACHE);

DBMS_LOB.MOVE_TO_DBFS_LINK (
  lob_loc      IN OUT CLOB CHARACTER SET ANY_CS,
  storage_path IN    VARCHAR2(dbfs_link_path_max_size),
  flags        IN    BINARY_INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

### Parameters

**Table 82–80 MOVE\_TO\_DBFS\_LINK Procedure Parameters**

Parameter	Description
lob_loc	LOB to be archived
storage_path	Path where the LOB will be stored
flags	Either DBFS_LINK_CACHE or DBFS_LINK_NOCACHE. If DBFS_LINK_CACHE is specified, the LOB data continues to be stored in the RDBMS as well as being written to the DBFS store. DBFS_LINK_NOCACHE specifies that the LOB data should be deleted from the RDBMS once written to the DBFS.

### Exceptions

**Table 82–81 MOVE\_TO\_DBFS\_LINK Procedure Exceptions**

Exception	Description
SECUREFILE_BADLOB	lob_loc is not a SECUREFILE

### Usage Notes

- If the LOB is already archived, the procedure silently returns as if the put was successful. In that case, if DBFS\_LINK\_NOCACHE is specified, or flags is defaulted, the LOB data is removed from the RDBMS.
- Calling this procedure multiple times on the same LOB with the same flags has no effect.
- Calling the procedure on a LOB that is already archived causes the LOB to be cached (DBFS\_LINK\_CACHE) or removed (DBFS\_LINK\_NOCACHE) according to the flag setting.

## OPEN Procedures

This procedure opens a LOB, internal or external, in the indicated mode. Valid modes include read-only, and read/write.

### Syntax

```
DBMS_LOB.OPEN (
  lob_loc  IN OUT NOCOPY BLOB,
  open_mode IN          BINARY_INTEGER);

DBMS_LOB.OPEN (
  lob_loc  IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  open_mode IN          BINARY_INTEGER);

DBMS_LOB.OPEN (
  file_loc IN OUT NOCOPY BFILE,
  open_mode IN          BINARY_INTEGER := file_readonly);
```

### Parameters

**Table 82–82 OPEN Procedure Parameters**

Parameter	Description
lob_loc	LOB locator. For more information, see <a href="#">Operational Notes</a> .
open_mode	Mode in which to open. For BLOB and CLOB types, the mode can be either: LOB_READONLY or LOB_READWRITE. For BFILE types, the mode must be FILE_READONLY.

### Usage Notes

---

**Note:** If the LOB was opened in read-only mode, and if you try to write to the LOB, then an error is returned. BFILE can only be opened with read-only mode.

---

OPEN requires a round-trip to the server for both internal and external LOBs. For internal LOBs, OPEN triggers other code that relies on the OPEN call. For external LOBs (BFILES), OPEN requires a round-trip because the actual operating system file on the server side is being opened.

It is not mandatory that you wrap all LOB operations inside the Open/Close interfaces. However, if you open a LOB, you must close it before you commit or rollback the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## READ Procedures

This procedure reads a piece of a LOB, and returns the specified amount into the buffer parameter, starting from an absolute offset from the beginning of the LOB.

The number of bytes or characters actually read is returned in the amount parameter. If the input `offset` points past the End of LOB, then amount is set to 0, and a `NO_DATA_FOUND` exception is raised.

### Syntax

```
DBMS_LOB.READ (
  lob_loc   IN          BLOB,
  amount    IN OUT NOCOPY INTEGER,
  offset    IN          INTEGER,
  buffer    OUT         RAW);

DBMS_LOB.READ (
  lob_loc   IN          CLOB CHARACTER SET ANY_CS,
  amount    IN OUT NOCOPY INTEGER,
  offset    IN          INTEGER,
  buffer    OUT         VARCHAR2 CHARACTER SET lob_loc%CHARSET);

DBMS_LOB.READ (
  file_loc  IN          BFILE,
  amount    IN OUT NOCOPY INTEGER,
  offset    IN          INTEGER,
  buffer    OUT         RAW);
```

### Parameters

**Table 82–83 READ Procedure Parameters**

Parameter	Description
<code>lob_loc</code>	Locator for the LOB to be read. For more information, see <a href="#">Operational Notes</a> .
<code>file_loc</code>	The file locator for the LOB to be examined.
<code>amount</code>	Number of bytes (for BLOBs) or characters (for CLOBs) to read, or number that were read.
<code>offset</code>	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).
<code>buffer</code>	Output buffer for the read operation.

### Exceptions

[Table 82–84](#) lists exceptions that apply to any LOB instance. [Table 82–85](#) lists exceptions that apply only to BFILES.

**Table 82–84 READ Procedure Exceptions**

Exception	Description
<code>VALUE_ERROR</code>	Any of <code>lob_loc</code> , <code>amount</code> , or <code>offset</code> parameters are NULL.



**Table 82–84 (Cont.) READ Procedure Exceptions**

Exception	Description
INVALID_ARGVAL	Either: <ul style="list-style-type: none"> <li>- amount &lt; 1</li> <li>- amount &gt; MAXBUFSIZE</li> <li>- offset &lt; 1</li> <li>- offset &gt; LOBMAXSIZE</li> <li>- amount is greater, in bytes or characters, than the capacity of buffer.</li> </ul>
NO_DATA_FOUND	End of the LOB is reached, and there are no more bytes or characters to read from the LOB: amount has a value of 0.

**Table 82–85 READ Procedure Exceptions for BFILES**

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

## Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.
- When calling DBMS\_LOB.READ from the client (for example, in a BEGIN/END block from within SQL\*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.
- READ gets the LOB, if necessary, before the read.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## SET\_DBFS\_LINK Procedures

This function links the specified SecureFile to the specified path name. It does not copy the data to the path.

### Syntax

```
DBMS_LOB.SET_DBFS_LINK (
  lob_loc      IN OUT BLOB,
  archive_id   IN     RAW(1024));

DBMS_LOB.SET_DBFS_LINK(
  lob_loc_dst  IN OUT CLOB CHARACTER SET ANY_CS,
  archive_id   IN     RAW(1024));
```

### Parameters

**Table 82–86** SET\_DBFS\_LINK Procedure Parameters

Parameter	Description
lob_loc	LOB for which to store the reference value
archive_id	Archive ID as returned by calling either of the <a href="#">GET_DBFS_LINK Functions</a> Functions

### Exceptions

**Table 82–87** SET\_DBFS\_LINK Procedure Exceptions

Exception	Description
SECUREFILE_BADLOB	lob_loc is not a SECUREFILE

## SETCONTENTTYPE Procedure

This procedure sets the content type string for the data in the LOB.

### Syntax

```
DBMS_LOB.SETCONTENTTYPE (
  lob_loc      IN OUT NOCOPY BLOB,
  contenttype  IN          VARCHAR2);

DBMS_LOB.SETCONTENTTYPE (
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  contenttype  IN          VARCHAR2);
```

### Parameters

**Table 82–88** SETCONTENTTYPE Procedure Parameters

Parameter	Description
lob_loc	LOB to be assigned the content type
contenttype	String to be assigned

### Exceptions

**Table 82–89** SETCONTENTTYPE Procedure Exceptions

Exception	Description
SECUREFILE_BADLOB	lob_loc is not a SECUREFILE

### Usage Notes

To clear an existing contenttype associated with a SECUREFILE, invoke SETCONTENTTYPE with contenttype set to empty string.

## SETOPTIONS Procedures

This procedure enables/disables CSCE features on a per-LOB basis, overriding the default LOB column settings.

### Syntax

```

DBMS_LOB.SETOPTIONS (
  lob_loc          IN      BLOB,
  option_types     IN      PLS_INTEGER,
  options          IN      PLS_INTEGER);

DBMS_LOB.SETOPTIONS (
  lob_loc          IN      CLOB CHARACTER SET ANY_CS,
  option_types     IN      PLS_INTEGER,
  options          IN      PLS_INTEGER);

```

### Parameters

**Table 82–90** SETOPTIONS Procedure Parameter

Parameter	Description
lob_loc	Locator for the LOB to be examined. For more information, see <a href="#">Operational Notes</a> .
option_type	See <a href="#">DBMS_LOB Constants - Option Types</a> on page 82-5
options	See <a href="#">DBMS_LOB Constants - Option Values</a> on page 82-5

### Exceptions

**Table 82–91** SETOPTIONS Procedure Exceptions

Exception	Description
SECUREFILE_BADLOB	Unsupported object type for the operation
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled

### Usage Notes

- DBMS\_LOB.SETOPTIONS cannot be used to enable or disable encryption on individual LOBs.
- You cannot turn compression or deduplication on or off for a SecureFile column that does not have those features on. The [GETOPTIONS Functions](#) and [SetOptions Procedures](#) work on individual SecureFiles. You can turn off a feature on a particular SecureFile and turn on a feature that has already been turned off by [SetOptions](#), but you cannot turn on an option that has not been given to the SecureFile when the table was created.

## SUBSTR Functions

This function returns amount bytes or characters of a LOB, starting from an absolute offset from the beginning of the LOB.

For fixed-width n-byte CLOBs, if the input amount for SUBSTR is greater than  $(32767/n)$ , then SUBSTR returns a character buffer of length  $(32767/n)$ , or the length of the CLOB, whichever is lesser. For CLOBs in a varying-width character set, n is the maximum byte-width used for characters in the CLOB.

### Syntax

```
DBMS_LOB.SUBSTR (
  lob_loc      IN      BLOB,
  amount       IN      INTEGER := 32767,
  offset       IN      INTEGER := 1)
RETURN RAW;

DBMS_LOB.SUBSTR (
  lob_loc      IN      CLOB CHARACTER SET ANY_CS,
  amount       IN      INTEGER := 32767,
  offset       IN      INTEGER := 1)
RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;

DBMS_LOB.SUBSTR (
  file_loc     IN      BFILE,
  amount       IN      INTEGER := 32767,
  offset       IN      INTEGER := 1)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

### Parameters

**Table 82–92 SUBSTR Function Parameters**

Parameter	Description
lob_loc	Locator for the LOB to be read. For more information, see <a href="#">Operational Notes</a> .
file_loc	The file locator for the LOB to be examined.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to be read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).

### Return Values

**Table 82–93 SUBSTR Function Return Values**

Return	Description
RAW	Function overloading that has a BLOB or BFILE in parameter.
VARCHAR2	CLOB version.

**Table 82–93 (Cont.) SUBSTR Function Return Values**

Return	Description
NULL	Either: <ul style="list-style-type: none"> <li>- any input parameter is NULL</li> <li>- amount &lt; 1</li> <li>- amount &gt; 32767</li> <li>- offset &lt; 1</li> <li>- offset &gt; LOBMAXSIZE</li> </ul>

## Exceptions

**Table 82–94 SUBSTR Function Exceptions for BFILE operations**

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

## Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.
- When calling DBMS\_LOB.SUBSTR from the client (for example, in a BEGIN/END block from within SQL\*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.
- DBMS\_LOB.SUBSTR will return 8191 or more characters based on the characters stored in the LOBs. If all characters are not returned as a consequence of the character byte size exceeding the available buffer, the user should either call DBMS\_LOB.SUBSTR with a new offset to read the remaining characters, or call the subprogram on loop until all the data is extracted.
- SUBSTR gets the LOB, if necessary, before read.

### See Also:

- ["INSTR Functions"](#) on page 82-57
- ["READ Procedures"](#) on page 82-72
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## TRIM Procedures

This procedure trims the value of the internal LOB to the length you specify in the `newlen` parameter. Specify the length in bytes for BLOBs, and specify the length in characters for CLOBs.

---

**Note:** The TRIM procedure decreases the length of the LOB to the value specified in the `newlen` parameter.

---

If you attempt to TRIM an empty LOB, then nothing occurs, and TRIM returns no error. If the new length that you specify in `newlen` is greater than the size of the LOB, then an exception is raised.

### Syntax

```
DBMS_LOB.TRIM (
  lob_loc      IN OUT NOCOPY BLOB,
  newlen      IN          INTEGER);

DBMS_LOB.TRIM (
  lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  newlen      IN          INTEGER);
```

### Parameters

**Table 82–95 TRIM Procedure Parameters**

Parameter	Description
<code>lob_loc</code>	Locator for the internal LOB whose length is to be trimmed. For more information, see <a href="#">Operational Notes</a> .
<code>newlen</code>	New, trimmed length of the LOB value in bytes for BLOBs or characters for CLOBs.

### Exceptions

**Table 82–96 TRIM Procedure Exceptions**

Exception	Description
VALUE_ERROR	<code>lob_loc</code> is NULL.
INVALID_ARGVAL	Either: - <code>new_len</code> < 0 - <code>new_len</code> > LOBMAXSIZE
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML slave
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

### Usage Notes

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close

it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the OPEN or CLOSE statement.
- TRIM gets the LOB, if necessary, before altering the length of the LOB, unless the new length specified is '0'

**See Also:**

- ["ERASE Procedures"](#) on page 82-35
- ["WRITEAPPEND Procedures"](#) on page 82-83
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure



## WRITE Procedures

This procedure writes a specified amount of data into an internal LOB, starting from an absolute offset from the beginning of the LOB. The data is written from the `buffer` parameter.

WRITE replaces (overwrites) any data that already exists in the LOB at the offset, for the length you specify.

### Syntax

```
DBMS_LOB.WRITE (
  lob_loc  IN OUT NOCOPY BLOB,
  amount   IN           INTEGER,
  offset   IN           INTEGER,
  buffer   IN           RAW);

DBMS_LOB.WRITE (
  lob_loc  IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  amount   IN           INTEGER,
  offset   IN           INTEGER,
  buffer   IN           VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

### Parameters

**Table 82–97** WRITE Procedure Parameters

Parameter	Description
<code>lob_loc</code>	Locator for the internal LOB to be written to. For more information, see <a href="#">Operational Notes</a>
<code>amount</code>	Number of bytes (for BLOBs) or characters (for CLOBs) to write
<code>offset</code>	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.
<code>buffer</code>	Input buffer for the write

### Exceptions

**Table 82–98** WRITE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of <code>lob_loc</code> , <code>amount</code> , or <code>offset</code> parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: <ul style="list-style-type: none"> <li>- <code>amount</code> &lt; 1</li> <li>- <code>amount</code> &gt; MAXBUFSIZE</li> <li>- <code>offset</code> &lt; 1</li> <li>- <code>offset</code> &gt; LOBMAXSIZE</li> </ul>
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML slave
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

**Table 82–98 (Cont.) WRITE Procedure Exceptions**

Exception	Description
SECUREFILE_OUTOFBOUNDS	Attempted to perform a write operation past the end of a LOB having <code>FRAGMENT_*</code> on it

## Usage Notes

- There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer is written to the LOB. If the offset you specify is beyond the end of the data currently in the LOB, then zero-byte fillers or spaces are inserted in the BLOB or CLOB respectively.
- The form of the `VARCHAR2` buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input LOB parameter is of type `CLOB`, then the buffer must contain `CHAR` data.
- When calling `DBMS_LOB.WRITE` from the client (for example, in a `BEGIN/END` block from within `SQL*Plus`), the buffer must contain data in the client's character set. The database converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.
- It is not mandatory that you wrap the LOB operation inside the `Open/Close` interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit or rollback the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the `Open/Close` API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.
- `WRITE` gets the LOB, if necessary, before writing the LOB, unless the write is specified to overwrite the entire LOB.

### See Also:

- ["APPEND Procedures"](#) on page 82-19
- ["COPY Procedures"](#) on page 82-29
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

## WRITEAPPEND Procedures

This procedure writes a specified amount of data to the end of an internal LOB. The data is written from the `buffer` parameter.

### Syntax

```
DBMS_LOB.WRITEAPPEND (
  lob_loc IN OUT NOCOPY BLOB,
  amount  IN           INTEGER,
  buffer  IN           RAW);

DBMS_LOB.WRITEAPPEND (
  lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
  amount  IN           INTEGER,
  buffer  IN           VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

### Parameters

**Table 82–99** *WRITEAPPEND Procedure Parameters*

Parameter	Description
<code>lob_loc</code>	Locator for the internal LOB to be written to. For more information, see <a href="#">Operational Notes</a>
<code>amount</code>	Number of bytes (for BLOBs) or characters (for CLOBs) to write
<code>buffer</code>	Input buffer for the write

### Usage Notes

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer are written to the end of the LOB.

### Exceptions

**Table 82–100** *WRITEAPPEND Procedure Exceptions*

Exception	Description
VALUE_ERROR	Any of <code>lob_loc</code> , <code>amount</code> , or <code>offset</code> parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - <code>amount</code> < 1 - <code>amount</code> > MAXBUFSIZE
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML slave
BUFFERING_ENABLED	Cannot perform operation with LOB buffering enabled if buffering is enabled on the LOB

### Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

- When calling `DBMS_LOB.WRITEAPPEND` from the client (for example, in a `BEGIN/END` block from within `SQL*Plus`), the buffer must contain data in the client's character set. The database converts the client-side buffer to the server's character set before it writes the buffer data to the `LOB`.
- It is not mandatory that you wrap the `LOB` operation inside the `Open/Close` interfaces. If you did not open the `LOB` before performing the operation, the functional and domain indexes on the `LOB` column are updated during the call. However, if you opened the `LOB` before performing the operation, you must close it before you commit or rollback the transaction. When an internal `LOB` is closed, it updates the functional and domain indexes on the `LOB` column.
- If you do not wrap the `LOB` operation inside the `Open/Close` API, the functional and domain indexes are updated each time you write to the `LOB`. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the `LOB` within the `OPEN` or `CLOSE` statement.
- `WRITEAPPEND` gets the `LOB`, if necessary, before appending to the `LOB`.

**See Also:**

- ["APPEND Procedures"](#) on page 82-19
- ["COPY Procedures"](#) on page 82-29
- ["WRITE Procedures"](#) on page 82-81
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

The DBMS\_LOCK package provides an interface to Oracle Lock Management services. You can request a lock of a specific mode, give it a unique name recognizable in another procedure in the same or another instance, change the lock mode, and release it.

**See Also:** For more information, and an example of how to use the DBMS\_LOCK package, see "About User Locks" in *Oracle Database Advanced Application Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_LOCK](#)
  - Overview
  - Security Model
  - Constants
  - Rules and Limits
  - Operational Notes
- [Summary of DBMS\\_LOCK Subprograms](#)

---

## Using DBMS\_LOCK

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Rules and Limits](#)
- [Operational Notes](#)

## Overview

Some uses of user locks:

- Providing exclusive access to a device, such as a terminal
- Providing application-level enforcement of read locks
- Detecting when a lock is released and cleanup after the application
- Synchronizing applications and enforcing sequential processing

## Security Model

There might be operating system-specific limits on the maximum number of total locks available. This *must* be considered when using locks or making this package available to other users. Consider granting the `EXECUTE` privilege only to specific users or roles.

A better alternative would be to create a cover package limiting the number of locks used and grant `EXECUTE` privilege to specific users. An example of a cover package is documented in the `DBMS_LOCK.SQL` package specification file. The abbreviations for these locks as they appear in Enterprise Manager monitors are in parentheses.



## Constants

The DBMS\_LOCK package uses the constants shown in [Table 83-1](#).

**Table 83-1 DBMS\_LOCK Constants**

Name	Alternate Name(s)	Type	Value	OEM Abbreviation	Description
NL_MODE	NuLl	INTEGER	1	-	-
SS_MODE	Sub Shared	INTEGER	2	ULRS	This can be used on an aggregate object to indicate that share locks are being acquired on subparts of the object.
SX_MODE	<ul style="list-style-type: none"> <li>■ Sub eXclusive</li> <li>■ Row Exclusive Mode</li> </ul>	INTEGER	3	ULRX	This can be used on an aggregate object to indicate that exclusive locks are being acquired on sub-parts of the object.
S_MODE	<ul style="list-style-type: none"> <li>■ Shared</li> <li>■ Row Exclusive Mode</li> <li>■ Intended Exclusive</li> </ul>	INTEGER	4	ULRSX	-
SSX_MODE	<ul style="list-style-type: none"> <li>■ Shared Sub eXclusive</li> <li>■ Share Row Exclusive Mode</li> </ul>	INTEGER	5	-	This indicates that the entire aggregate object has a share lock, but some of the sub-parts may additionally have exclusive locks.
X_MODE	Exclusive	INTEGER	6	ULX	-

These are the various lock modes (nl -> "NuLl", ss -> "Sub Shared", sx -> "Sub eXclusive", s -> "Shared", ssx -> "Shared Sub eXclusive", x -> "eXclusive").

## Rules and Limits

When another process holds "held", an attempt to get "get" does the following:

**Table 83–2 Lock Compatibility**

<b>HELD MODE</b>	<b>GET NL</b>	<b>GET SS</b>	<b>GET SX</b>	<b>GET S</b>	<b>GET SSX</b>	<b>GET X</b>
NL	Success	Success	Success	Success	Success	Success
SS	Success	Success	Success	Success	Success	Fail
SX	Success	Success	Success	Fail	Fail	Fail
S	Success	Success	Fail	Success	Fail	Fail
SSX	Success	Success	Fail	Fail	Fail	Fail
X	Success	Fail	Fail	Fail	Fail	Fail

```
maxwait constant integer := 32767;
```

The constant `maxwait` waits forever.

## Operational Notes

User locks never conflict with Oracle locks because they are identified with the prefix "UL". You can view these locks using the Enterprise Manager lock monitor screen or the appropriate fixed views. User locks are automatically released when a session terminates. The lock identifier is a number in the range of 0 to 1073741823.

Because a reserved user lock is the same as an Oracle lock, it has all the functionality of an Oracle lock, such as deadlock detection. Be certain that any user locks used in distributed transactions are released upon `COMMIT`, or an undetected deadlock may occur.

DBMS\_LOCK is most efficient with a limit of a few hundred locks for each session. Oracle strongly recommends that you develop a standard convention for using these locks in order to avoid conflicts among procedures trying to use the same locks. For example, include your company name as part of your lock names.

## Summary of DBMS\_LOCK Subprograms

**Table 83–3** *DBMS\_LOCK Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">ALLOCATE_UNIQUE Procedure</a> on page 83-9	Allocates a unique lock ID to a named lock
<a href="#">CONVERT Function</a> on page 83-11	Converts a lock from one mode to another
<a href="#">RELEASE Function</a> on page 83-12	Releases a lock
<a href="#">REQUEST Function</a> on page 83-13	Requests a lock of a specific mode.
<a href="#">SLEEP Procedure</a> on page 83-14	Puts a session to sleep for a specific time

## ALLOCATE\_UNIQUE Procedure

This procedure allocates a unique lock identifier (in the range of 1073741824 to 1999999999) a specified lock name. Lock identifiers are used to enable applications to coordinate their use of locks. This is provided because it may be easier for applications to coordinate their use of locks based on lock names rather than lock numbers.

### Syntax

```
DBMS_LOCK.ALLOCATE_UNIQUE (
    lockname          IN  VARCHAR2,
    lockhandle       OUT VARCHAR2,
    expiration_secs  IN  INTEGER  DEFAULT 864000);
```

### Parameters

**Table 83–4** *ALLOCATE\_UNIQUE Procedure Parameters*

Parameter	Description
lockname	Name of the lock for which you want to generate a unique ID. Do not use lock names beginning with ORA\$; these are reserved for products supplied by Oracle.
lockhandle	Returns the handle to the lock ID generated by <code>ALLOCATE_UNIQUE</code> .  You can use this handle in subsequent calls to <code>REQUEST</code> , <code>CONVERT</code> , and <code>RELEASE</code> .  A handle is returned instead of the actual lock ID to reduce the chance that a programming error accidentally creates an incorrect, but valid, lock ID. This provides better isolation between different applications that are using this package.  LOCKHANDLE can be up to VARCHAR2 (128).  All sessions using a lock handle returned by <code>ALLOCATE_UNIQUE</code> with the same lock name are referring to the same lock. Therefore, do not pass lock handles from one session to another.
expiration_secs	Number of seconds to wait after the last <code>ALLOCATE_UNIQUE</code> has been performed on a specified lock, before permitting that lock to be deleted from the <code>DBMS_LOCK_ALLOCATED</code> table.  The default waiting period is 10 days. You should not delete locks from this table. Subsequent calls to <code>ALLOCATE_UNIQUE</code> may delete expired locks to recover space.

### Usage Notes

If you choose to identify locks by name, you can use `ALLOCATE_UNIQUE` to generate a unique lock identification number for these named locks.

The first session to call `ALLOCATE_UNIQUE` with a new lock name causes a unique lock ID to be generated and stored in the `dbms_lock_allocated` table. Subsequent calls (usually by other sessions) return the lock ID previously generated.

A lock name is associated with the returned lock ID for at least `expiration_secs` (defaults to 10 days) past the last call to `ALLOCATE_UNIQUE` with the specified lock name. After this time, the row in the `dbms_lock_allocated` table for this lock name may be deleted in order to recover space. `ALLOCATE_UNIQUE` performs a commit.

**Note:** Named user locks may be less efficient, because Oracle uses SQL to determine the lock associated with a specified name.

### Exceptions

ORA-20000, ORU-10003: Unable to find or insert lock <lockname> into catalog dbms\_lock\_allocated.

## CONVERT Function

This function converts a lock from one mode to another. CONVERT is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE\_UNIQUE procedure.

### Syntax

```
DBMS_LOCK.CONVERT(
  id          IN INTEGER ||
  lockhandle  IN VARCHAR2,
  lockmode   IN INTEGER,
  timeout     IN NUMBER DEFAULT MAXWAIT)
RETURN INTEGER;
```

### Parameters

**Table 83–5** CONVERT Function Parameters

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change
lockmode	New mode that you want to assign to the specified lock. For the available modes and their associated integer identifiers, see <a href="#">Constants</a> on page 83-5.
timeout	Number of seconds to continue trying to change the lock mode.  If the lock cannot be converted within this time period, then the call returns a value of 1 (timeout).

### Return Values

**Table 83–6** CONVERT Function Return Values

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Don't own lock specified by id or lockhandle
5	Illegal lock handle

## RELEASE Function

This function explicitly releases a lock previously acquired using the `REQUEST` function. Locks are automatically released at the end of a session. `RELEASE` is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the `ALLOCATE_UNIQUE` procedure.

### Syntax

```
DBMS_LOCK.RELEASE (  
    id          IN INTEGER)  
RETURN INTEGER;
```

```
DBMS_LOCK.RELEASE (  
    lockhandle IN VARCHAR2)  
RETURN INTEGER;
```

### Parameters

**Table 83–7** *RELEASE Function Parameter*

Parameter	Description
<code>id</code> or <code>lockhandle</code>	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by <code>ALLOCATE_UNIQUE</code> , of the lock mode you want to change

### Return Values

**Table 83–8** *RELEASE Function Return Values*

Return Value	Description
0	Success
3	Parameter error
4	Do not own lock specified by <code>id</code> or <code>lockhandle</code>
5	Illegal lock handle



## REQUEST Function

This function requests a lock with a specified mode. `REQUEST` is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the `ALLOCATE_UNIQUE` procedure.

### Syntax

```
DBMS_LOCK.REQUEST(
  id                IN  INTEGER ||
  lockhandle        IN  VARCHAR2,
  lockmode          IN  INTEGER DEFAULT X_MODE,
  timeout           IN  INTEGER DEFAULT MAXWAIT,
  release_on_commit IN  BOOLEAN DEFAULT FALSE)
RETURN INTEGER;
```

The current default values, such as `X_MODE` and `MAXWAIT`, are defined in the `DBMS_LOCK` package specification.

### Parameters

**Table 83–9** *REQUEST Function Parameters*

Parameter	Description
<code>id</code> or <code>lockhandle</code>	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by <code>ALLOCATE_UNIQUE</code> , of the lock mode you want to change
<code>lockmode</code>	Mode that you are requesting for the lock. For the available modes and their associated integer identifiers, see <a href="#">Constants</a> on page 83-5.
<code>timeout</code>	Number of seconds to continue trying to grant the lock. If the lock cannot be granted within this time period, then the call returns a value of 1 ( <code>timeout</code> ).
<code>release_on_commit</code>	Set this parameter to <code>TRUE</code> to release the lock on commit or roll-back. Otherwise, the lock is held until it is explicitly released or until the end of the session.

### Return Values

**Table 83–10** *REQUEST Function Return Values*

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Already own lock specified by <code>id</code> or <code>lockhandle</code>
5	Illegal lock handle

## SLEEP Procedure

This procedure suspends the session for a specified period of time.

### Syntax

```
DBMS_LOCK.SLEEP (  
    seconds IN NUMBER);
```

### Parameters

**Table 83–11 SLEEP Procedure Parameters**

Parameter	Description
seconds	Amount of time, in seconds, to suspend the session. The smallest increment can be entered in hundredths of a second; for example, 1.95 is a legal time value.

The DBMS\_LOGMNR package, one of a set of LogMiner packages, contains the subprograms you use to initialize the LogMiner tool and to begin and end a LogMiner session.

**See Also:** *Oracle Database Utilities* for information regarding LogMiner.

This chapter contains the following topics:

- [Using DBMS\\_LOGMNR](#)
  - Overview
  - Security Model
  - Constants
  - Views
  - Operational Notes
- [Summary of DBMS\\_LOGMNR Subprograms](#)

## Using DBMS\_LOGMNR

This section contains the following topics, which relate to using the DBMS\_LOGMNR package:

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)
- [Operational Notes](#)

## Overview

Oracle LogMiner, which is part of Oracle Database, enables you to query online and archived redo log files through a SQL interface. The `DBMS_LOGMNR` package provides the majority of the tools needed to start and stop LogMiner and specify the redo log files of interest.

All changes made to user data or to the database dictionary are recorded in the Oracle redo log files so that database recovery operations can be performed. You can take advantage of the data recorded in the redo log files to accomplish other tasks, such as:

- Pinpointing when a logical corruption to a database, such as errors made at the application level, may have begun
- Determining what actions you would have to take to perform fine-grained recovery at the transaction level.
- Performance tuning and capacity planning through trend analysis.
- Track any data manipulation language (DML) and data definition language (DDL) statements executed on the database, the order in which they were executed, and who executed them.

**See Also:** [Chapter 85, "DBMS\\_LOGMNR\\_D"](#) for information on the package subprograms that extract a LogMiner dictionary and re-create LogMiner tables in alternate tablespaces

## Security Model

You must have the EXECUTE\_CATALOG\_ROLE role to use the DBMS\_LOGMNR package.

## Constants

The DBMS\_LOGMNR package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS\_LOGMNR.NEW.

Table 84–1 describes the constants for the ADD\_LOGFILE options flag in the DBMS\_LOGMNR package.

**Table 84–1 Constants for ADD\_LOGFILE Options Flag**

Constant	Description
NEW	Implicitly calls the DBMS_LOGMNR.END_LOGMNR procedure to end the current LogMiner session and then creates a new session. The new session starts a new list of redo log files to be analyzed, beginning with the redo log file you specify.
ADDFILE	Adds the specified redo log file to the list of redo log files to be analyzed. Any attempt to add a duplicate file raises an exception (ORA-01289). This is the default if no options flag is specified.

Table 84–2 describes the constants for the START\_LOGMNR options flag in the DBMS\_LOGMNR package.

**Table 84–2 Constants for START\_LOGMNR Options Flag**

Constant	Description
COMMITTED_DATA_ONLY	<p>If set, DML statements corresponding to committed transactions are returned. DML statements corresponding to a committed transaction are grouped together. Transactions are returned in their commit order. Transactions that are rolled back or in-progress are filtered out, as are internal redo records (those related to index operations, management, and so on).</p> <p>If this option is not set, all rows for all transactions (committed, rolled back, and in-progress) are returned in the order in which they are found in the redo logs (in order of SCN values).</p>
SKIP_CORRUPTION	<p>Directs a select operation on the V\$LOGMNR_CONTENTS view to skip any corruptions in the redo log file being analyzed and continue processing. This option works only when a block in the redo log file (and not the header of the redo log file) is corrupt. You should check the INFO column in the V\$LOGMNR_CONTENTS view to determine the corrupt blocks skipped by LogMiner. When a corruption in the redo log file is skipped, the OPERATION column contains the value CORRUPTED_BLOCKS, and the STATUS column contains the value 1343.</p>
DDL_DICT_TRACKING	<p>If the LogMiner dictionary in use is a flat file or in the redo log files, LogMiner updates its internal dictionary if a DDL event occurs. This ensures that correct SQL_REDO and SQL_UNDO information is maintained for objects that are modified after the LogMiner internal dictionary is built. The database to which LogMiner is connected must be open.</p> <p>This option cannot be used in conjunction with the DICT_FROM_ONLINE_CATALOG option and cannot be used when the LogMiner dictionary being used is one that was extracted to a flat file prior to Oracle9i.</p>

**Table 84–2 (Cont.) Constants for START\_LOGMNR Options Flag**

Constant	Description
<p>DICT_FROM_ONLINE_CATALOG</p>	<p>Directs LogMiner to use the current online database dictionary rather than a LogMiner dictionary contained in a flat file or in the redo log files being analyzed.</p> <p>This option cannot be used in conjunction with the DDL_DICT_TRACKING option. The database to which LogMiner is connected must be the same one that generated the redo log files.</p> <p>Expect to see a value of 2 in the STATUS column of the V\$LOGMNR_CONTENTS view if the table definition in the database does not match the table definition in the redo log file.</p>
<p>DICT_FROM_REDO_LOGS</p>	<p>If set, LogMiner expects to find a LogMiner dictionary in the redo log files that were specified. The redo log files are specified with the DBMS_LOGMNR.ADD_LOGFILE procedure or with the DBMS_LOGMNR.START_LOGMNR procedure with the CONTINUOUS_MINE option.</p>
<p>NO_SQL_DELIMITER</p>	<p>If set, the SQL delimiter (a semicolon) is not placed at the end of reconstructed SQL statements. This is helpful for applications that open a cursor and then execute the reconstructed statements.</p>
<p>NO_ROWID_IN_STMT</p>	<p>If set, the ROWID clause is not included in the reconstructed SQL statements. The redo log file may already contain logically unique identifiers for modified rows if supplemental logging is enabled.</p> <p>When using this option, you must be sure that supplemental logging was enabled in the source database at the appropriate level and that no duplicate rows exist in the tables of interest. LogMiner does not make any guarantee regarding the uniqueness of logical row identifiers.</p>
<p>PRINT_PRETTY_SQL</p>	<p>If set, LogMiner formats the reconstructed SQL statements for ease of reading. These reconstructed SQL statements are not executable.</p>
<p>CONTINUOUS_MINE</p>	<p>Directs LogMiner to automatically add redo log files, as needed, to find the data of interest. You only need to specify the first log to start mining, or just the starting SCN or date to indicate to LogMiner where to begin mining logs. You are not required to specify any redo log files explicitly. LogMiner automatically adds and mines the (archived and online) redo log files for the data of interest. This option requires that LogMiner is connected to the same database instance that is generating the redo log files. It also requires that the database be mounted and that archiving be enabled.</p> <p>Beginning with Oracle Database release 10.1, the CONTINUOUS_MINE options is supported for use in an Oracle Real Application Clusters (Oracle RAC) environment.</p>
<p>STRING_LITERALS_IN_STMT</p>	<p>If set, SQL_REDO and SQL_UNDO use literals for numbers and datetime and interval column types.</p>



## Views

The DBMS\_LOGMNR package uses the views listed in the section on Accessing LogMiner Operational Information in Views in *Oracle Database Utilities*.

## Operational Notes

A **LogMiner session** begins with a call to `DBMS_LOGMNR.ADD_LOGFILE` or `DBMS_LOGMNR.START_LOGMNR` (the former if you plan to specify log files explicitly; the latter if you plan to use continuous mining). The session ends with a call to `DBMS_LOGMNR.END_LOGMNR`. Within a LogMiner session, you can specify the redo log files to be analyzed and the SCN or time range of interest; then you can issue `SQL SELECT` statements against the `V$LOGMNR_CONTENTS` view to retrieve the data of interest.

---

## Summary of DBMS\_LOGMNR Subprograms

**Table 84–3 DBMS\_LOGMNR Package Subprograms**

Subprogram	Description
<a href="#">ADD_LOGFILE Procedure</a> on page 84-10	Adds a redo log file to the existing or newly created list of redo log files for LogMiner to process, so that if a new list is created, this marks the beginning of a LogMiner session
<a href="#">COLUMN_PRESENT Function</a> on page 84-12	Call this function for any row returned from the V\$LOGMNR_CONTENTS view to determine if undo or redo column values exist for the column specified by the <code>column_name</code> input parameter to this function
<a href="#">END_LOGMNR Procedure</a> on page 84-14	Finishes a LogMiner session
<a href="#">MINE_VALUE Function</a> on page 84-15	Call this function for any row returned from the V\$LOGMNR_CONTENTS view to retrieve the undo or redo column value of the column specified by the <code>column_name</code> input parameter to this function
<a href="#">REMOVE_LOGFILE Procedure</a> on page 84-17	Removes a redo log file from the list of redo log files for LogMiner to process
<a href="#">START_LOGMNR Procedure</a> on page 84-18	Initializes the LogMiner utility and starts LogMiner (unless the session was already started with a call to <code>DBMS_LOGMNR.ADD_LOGFILE</code> )

## ADD\_LOGFILE Procedure

This procedure adds a file to an existing or newly created list of log files for LogMiner to process.

### Syntax

```
DBMS_LOGMNR.ADD_LOGFILE (
  LogFileName      IN VARCHAR2,
  options          IN BINARY_INTEGER default ADDFILE );
```

### Parameters

**Table 84–4 ADD\_LOGFILE Procedure Parameters**

Parameter	Description
LogFileName	Specifies the name of the redo log file to add to the list of redo log files to be analyzed during this session.
options	Does one of the following: <ul style="list-style-type: none"> <li>■ Starts a new LogMiner session and a new list of redo log files for analysis (DBMS_LOGMNR.NEW)</li> <li>■ Adds a file to an existing list of redo log files for analysis (DBMS_LOGMNR.ADDFILE)</li> </ul> See <a href="#">Table 84–1, "Constants for ADD_LOGFILE Options Flag"</a> .

### Exceptions

**Table 84–5 ADD\_LOGFILE Procedure Exceptions**

Exception	Description
ORA-01284	Specified file cannot be opened.
ORA-01287	Specified file is from a different database incarnation.
ORA-01289	Specified file has already been added to the list. Duplicate redo log files cannot be added.
ORA-01290	Specified file is not in the current list and therefore cannot be removed from the list.
ORA-01324	Specified file cannot be added to the list because there is a DB_ID mismatch.

### Usage Notes

- Before querying the V\$LOGMNR\_CONTENTS view, you must make a successful call to the DBMS\_LOGMNR.START\_LOGMNR procedure (within the current LogMiner session).
- Unless you specify the CONTINUOUS\_MINE option, the LogMiner session must be set up with a list of redo log files to be analyzed. Use the ADD\_LOGFILE procedure to specify the list of redo log files to analyze.
- If you are not using the CONTINUOUS\_MINE option and you want to analyze more than one redo log file, you must call the ADD\_LOGFILE procedure separately for each redo log file. The redo log files do not need to be registered in any particular order.

- Both archived and online redo log files can be mined.
- After you have added the first redo log file to the list, each additional redo log file that you add to the list must be associated with the same database and database RESETLOGS SCN as the first redo log file. (The database RESETLOGS SCN uniquely identifies each execution of an ALTER DATABASE OPEN RESETLOGS statement. When the online redo logs are reset, Oracle creates a new and unique incarnation of the database.)
- To analyze the redo log files from a different database (or a database incarnation with a different database RESETLOGS SCN) than that with which the current list of redo log files is associated, use the END\_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD\_LOGFILE procedure.
- LogMiner matches redo log files by the log sequence number. Thus, two redo log files with different names but with the same log sequence number will return the ORA-01289 exception. For instance, the online counterpart of an archived redo log file has a different name from the archived redo log file, but attempting to register it with LogMiner after registering the archived counterpart will result in the ORA-01289 exception being returned.

## COLUMN\_PRESENT Function

This function is designed to be used in conjunction with the `MINE_VALUE` function.

If the `MINE_VALUE` function returns a `NULL` value, it can mean either:

- The specified column is not present in the redo or undo portion of the data.
- The specified column is present and has a `NULL` value.

To distinguish between these two cases, use the `COLUMN_PRESENT` function, which returns a 1 if the column is present in the redo or undo portion of the data. Otherwise, it returns a 0.

### Syntax

```
DBMS_LOGMNR.COLUMN_PRESENT (
    sql_redo_undo      IN RAW,
    column_name        IN VARCHAR2 default '') RETURN NUMBER;
```

### Parameters

**Table 84–6** *COLUMN\_PRESENT Function Parameters*

Parameter	Description
<code>sql_redo_undo</code>	Specifies either the <code>REDO_VALUE</code> or the <code>UNDO_VALUE</code> column in the <code>V\$logmnr_contents</code> view from which to extract data values. See the Usage Notes for more information.
<code>column_name</code>	Specifies the fully qualified name ( <code>schema.table.column</code> ) of the column for which this function will return information.

### Return Values

[Table 84–7](#) describes the return values for the `COLUMN_PRESENT` function. The `COLUMN_PRESENT` function returns 1 if the self-describing record (the first parameter) contains the column specified in the second parameter. This can be used to determine the meaning of `NULL` values returned by the `DBMS_LOGMNR.MINE_VALUE` function.

**Table 84–7** *Return Values for COLUMN\_PRESENT Function*

Return	Description
0	Specified column is not present in this row of <code>V\$logmnr_contents</code> .
1	Column is present in this row of <code>V\$logmnr_contents</code> .

### Exceptions

**Table 84–8** *COLUMN\_PRESENT Function Exceptions*

Exception	Description
ORA-01323	Currently, a LogMiner dictionary is not associated with the LogMiner session. You must specify a LogMiner dictionary for the LogMiner session.
ORA-00904	Value specified for the <code>column_name</code> parameter is not a fully qualified column name.

## Usage Notes

- To use the COLUMN\_PRESENT function, you must have successfully started LogMiner.
- The COLUMN\_PRESENT function must be invoked in the context of a select operation on the V\$logmnr\_contents view.
- The COLUMN\_PRESENT function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- The value for the sql\_redo\_undo parameter depends on the operation performed and the data of interest:
  - If an update operation was performed and you want to know what the value was prior to the update operation, specify UNDO\_VALUE.
  - If an update operation was performed and you want to know what the value is after the update operation, specify REDO\_VALUE.
  - If an insert operation was performed, typically you would specify REDO\_VALUE (because the value of a column prior to an insert operation will always be NULL).
  - If a delete operation was performed, typically you would specify UNDO\_VALUE (because the value of a column after a delete operation will always be NULL).

## END\_LOGMNR Procedure

This procedure finishes a LogMiner session. Because this procedure performs cleanup operations that may not otherwise be done, you must use it to properly end a LogMiner session. This procedure is called automatically when you log out of a database session or when you call `DBMS_LOGMNR.ADD_LOGFILE` and specify the `NEW` option.

### Syntax

```
DBMS_LOGMNR.END_LOGMNR;
```

### Exceptions

**Table 84–9** *END\_LOGMNR Procedure Exception*

Exception	Description
ORA-01307	No LogMiner session is currently active. The <code>END_LOGMNR</code> procedure was called without adding any log files or before the <code>START_LOGMNR</code> procedure was called



## MINE\_VALUE Function

This function facilitates queries based on a column's data value. This function takes two arguments. The first one specifies whether to mine the redo (REDO\_VALUE) or undo (UNDO\_VALUE) portion of the data. The second argument is a string that specifies the fully qualified name of the column to be mined. The MINE\_VALUE function always returns a string that can be converted back to the original datatype.

### Syntax

```
DBMS_LOGMNR.MINE_VALUE (
    sql_redo_undo      IN RAW,
    column_name        IN VARCHAR2 default '') RETURN VARCHAR2;
```

### Parameters

**Table 84–10 MINE\_VALUE Function Parameters**

Parameter	Description
sql_redo_undo	Specifies either the REDO_VALUE or the UNDO_VALUE column in the V\$LOGMNR_CONTENTS view from which to extract data values. See the Usage Notes for more information.
column_name	Specifies the fully qualified name (schema.table.column) of the column for which this function will return information.

### Return Values

**Table 84–11 Return Values for MINE\_VALUE Function**

Return	Description
NULL	The column is not contained within the self-describing record, or the column value is NULL. To distinguish between the two different null possibilities, use the DBMS_LOGMNR.COLUMN_PRESENT function.
NON-NULL	The column is contained within the self-describing record; the value is returned in string format.

### Exceptions

**Table 84–12 MINE\_VALUE Function Exceptions**

Exception	Description
ORA-01323	Invalid state. Currently, a LogMiner dictionary is not associated with the LogMiner session. You must specify a LogMiner dictionary for the LogMiner session.
ORA-00904	Invalid identifier. The value specified for the column_name parameter was not a fully qualified column name.

### Usage Notes

- To use the MINE\_VALUE function, you must have successfully started LogMiner.
- The MINE\_VALUE function must be invoked in the context of a select operation from the V\$LOGMNR\_CONTENTS view.

- The MINE\_VALUE function does not support LONG, LOB, ADT, or COLLECTION datatypes.
- The value for the `sql_redo_undo` parameter depends on the operation performed and the data of interest:
  - If an update operation was performed and you want to know what the value was prior to the update operation, specify `UNDO_VALUE`.
  - If an update operation was performed and you want to know what the value is after the update operation, specify `REDO_VALUE`.
  - If an insert operation was performed, typically you would specify `REDO_VALUE` (because the value of a column prior to an insert operation will always be null).
  - If a delete operation was performed, typically you would specify `UNDO_VALUE` (because the value of a column after a delete operation will always be null).

## REMOVE\_LOGFILE Procedure

This procedure removes a redo log file from an existing list of redo log files for LogMiner to process.

---

**Note:** This procedure replaces the REMOVEFILE constant that was an option on the ADD\_LOGFILE procedure prior to Oracle Database 10g.

---

### Syntax

```
DBMS_LOGMNR.REMOVE_LOGFILE (
    LogFileName      IN VARCHAR2);
```

### Parameters

**Table 84–13 REMOVE\_LOGFILE Procedure Parameters**

Parameter	Description
LogFileName	Specifies the name of the redo log file to be removed from the list of redo log files to be analyzed during this session.

### Exceptions

**Table 84–14 REMOVE\_LOGFILE Procedure Exception**

Exception	Description
ORA-01290	Cannot remove unlisted log file

### Usage Notes

- Before querying the V\$LOGMNR\_CONTENTS view, you must make a successful call to the DBMS\_LOGMNR.START\_LOGMNR procedure (within the current LogMiner session).
- You can use this procedure to remove a redo log file from the list of redo log files for LogMiner to process if you know that redo log file does not contain any data of interest.
- Multiple redo log files can be removed by calling this procedure repeatedly.
- The redo log files do not need to be removed in any particular order.
- To start a new list of redo log files for analysis, use the END\_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD\_LOGFILE procedure.
- Even if you remove all redo log files from the list, any subsequent calls you make to the ADD\_LOGFILE procedure must match the database ID and RESETLOGS SCN of the removed redo log files. Therefore, to analyze the redo log files from a different database (or a database incarnation with a different database RESETLOGS SCN) than that with which the current list of redo log files is associated, use the END\_LOGMNR procedure to end the current LogMiner session, and then build a new list using the ADD\_LOGFILE procedure.

## START\_LOGMNR Procedure

This procedure starts LogMiner by loading the dictionary that LogMiner will use to translate internal schema object identifiers to names.

### Syntax

```
DBMS_LOGMNR.START_LOGMNR (
  startScn          IN NUMBER default 0,
  endScn            IN NUMBER default 0,
  startTime         IN DATE default '01-jan-1988',
  endTime           IN DATE default '31-dec-2110',
  DictFileName      IN VARCHAR2 default '',
  Options           IN BINARY_INTEGER default 0 );
```

### Parameters

**Table 84–15** START\_LOGMNR Procedure Parameters

Parameter	Description
startScn	Directs LogMiner to return only redo records with an SCN greater than or equal to the <code>startScn</code> specified. This fails if there is no redo log file containing the specified <code>startScn</code> value. (You can query the <code>FILENAME</code> , <code>LOW_SCN</code> , and <code>NEXT_SCN</code> columns in the <code>V\$LOGMNR_LOGS</code> view for each redo log file to determine the range of SCN values contained in each redo log file.)
endScn	Directs LogMiner to return only redo records with an SCN less than or equal to the <code>endScn</code> specified. If you specify an <code>endScn</code> value that is beyond the value in any redo log file, then LogMiner uses the greatest <code>endScn</code> value in the redo log file that contains the most recent changes. (You can query the <code>FILENAME</code> , <code>LOW_SCN</code> , and <code>NEXT_SCN</code> columns in the <code>V\$LOGMNR_LOGS</code> view for each redo log file to determine the range of SCN values contained in each redo log file.)
startTime	Directs LogMiner to return only redo records with a timestamp greater than or equal to the <code>startTime</code> specified. This fails if there is no redo log file containing the specified <code>startTime</code> value. (You can query the <code>FILENAME</code> , <code>LOW_TIME</code> , and <code>HIGH_TIME</code> columns in the <code>V\$LOGMNR_LOGS</code> view for each redo log file to determine the range of time covered in each redo log file.)  This parameter is ignored if <code>startScn</code> is specified. See the Usage Notes for additional information.
endTime	Directs LogMiner to return only redo records with a timestamp less than or equal to the <code>endTime</code> specified. If you specify an <code>endTime</code> value that is beyond the value in any redo log file, then LogMiner will use the greatest <code>endTime</code> in the redo log file that contains the most recent changes. You can query the <code>FILENAME</code> , <code>LOW_TIME</code> , and <code>HIGH_TIME</code> columns in the <code>V\$LOGMNR_LOGS</code> view for each redo log file to determine the range of time covered in each redo log file.)  This parameter is ignored if <code>endScn</code> is specified. See the Usage Notes for additional information.

**Table 84–15 (Cont.) START\_LOGMNR Procedure Parameters**

Parameter	Description
DictFileName	Specifies the flat file that contains the LogMiner dictionary. It is used to reconstruct <code>SQL_REDO</code> and <code>SQL_UNDO</code> columns in <code>V\$LOGMNR_CONTENTS</code> , as well as to fully translate <code>SEG_NAME</code> , <code>SEG_OWNER</code> , <code>SEG_TYPE_NAME</code> , <code>TABLE_NAME</code> , and <code>TABLE_SPACE</code> columns. The fully qualified path name for the LogMiner dictionary file must be specified. (This file must have been created previously through the <code>DBMS_LOGMNR_D.BUILD</code> procedure.)  You need to specify this parameter only if neither <code>DICT_FROM_REDO_LOGS</code> nor <code>DICT_FROM_ONLINE_CATALOG</code> is specified.
options	See <a href="#">Table 84–2, "Constants for START_LOGMNR Options Flag"</a> .

## Exceptions

**Table 84–16 START\_LOGMNR Procedure Exceptions**

Exception	Description
ORA-01280	Internal error encountered.
ORA-01281	<code>startScn</code> or <code>endScn</code> parameter value is not a valid SCN, or <code>endScn</code> is less than <code>startScn</code> .
ORA-01282	value for the <code>startTime</code> parameter was greater than the value specified for the <code>endTime</code> parameter, or there was no redo log file that was compatible with the date range specified with the <code>startTime</code> and <code>endTime</code> parameters.
ORA-01283	Options parameter specified is invalid.
ORA-01284	LogMiner dictionary file specified in the <code>DictFileName</code> parameter has a full path length greater than 256 characters, or the file cannot be opened.
ORA-01285	Error reading specified file.
ORA-01291	Redo log files that are needed to satisfy the user's requested SCN or time range are missing.
ORA-01292	No log file has been specified for the current LogMiner session.
ORA-01293	Mounted database required for specified LogMiner options.
ORA-01294	Error occurred while processing information in the specified dictionary file, possible corruption.
ORA-01295	Specified LogMiner dictionary does not correspond to the database that produced the log files being analyzed.
ORA-01296	Character set mismatch between specified LogMiner dictionary and log files.
ORA-01297	Redo version mismatch between LogMiner dictionary and log files.
ORA-01299	Specified LogMiner dictionary corresponds to a different database incarnation.
ORA-01300	Writable database required for specified LogMiner options.

## Usage Notes

- LogMiner can use a dictionary that you previously extracted to the redo log files or to a flat file, or you can specify that LogMiner use the online catalog if LogMiner is mining data from the source system. See *Oracle Database Utilities* and

Chapter 85, "DBMS\_LOGMNR\_D" in this manual for more information about the LogMiner dictionary.

- After executing the `START_LOGMNR` procedure, you can query the following views:
  - `V$LOGMNR_CONTENTS` - contains history of information in redo log files
  - `V$LOGMNR_DICTIONARY` - contains current information about the LogMiner dictionary file extracted to a flat file
  - `V$LOGMNR_PARAMETERS` - contains information about the LogMiner session(You can query the `V$LOGMNR_LOGS` view after a redo log file list has been added to the list of files that LogMiner is to mine.)
- Parameters and options are not persistent across calls to `DBMS_LOGMNR.START_LOGMNR`. You must specify all desired parameters and options (including SCN and time ranges) each time you call `DBMS_LOGMNR.START_LOGMNR`
- Be aware that specifying redo log files using a timestamp is not precise.
- The `CONTINUOUS_MINE` option directs LogMiner to automatically add redo log files, as needed, to find the data of interest. You need to specify only the first log to start mining, or just the starting SCN or date to indicate to LogMiner where to begin mining logs. Keep the following in mind when using the `CONTINUOUS_MINE` option:
  - The database control file will hold information about a limited number of archived redo log files, although the number of entries can be quite large. Query the `V$ARCHIVED_LOGS` view to determine which redo log file entries will be found by LogMiner.

Even if an entry is listed in the database control file (and the `V$ARCHIVED_LOGS` view), the archived redo log file may not be accessible by LogMiner for various reasons. For example, the archived redo log file may have been deleted or moved from its location (maybe because of a backup operation to tape), or the directory where it resides may not be available.
  - If you specify the `CONTINUOUS_MINE` option and an ending time or SCN that will occur in the future (or you do not specify an end time or SCN), a query of the `V$LOGMNR_CONTENTS` view will not finish until the database has generated redo log files beyond the specified time or SCN. In this scenario, LogMiner will automatically add archived redo log files to the LogMiner redo log file list as they are generated. In addition, in this scenario only, LogMiner may automatically remove redo log files from the list to keep it at 50 processed redo files. This is to save PGA memory as LogMiner automatically adds redo log files to the list. If LogMiner did not perform automated removal, memory could eventually be exhausted.
  - LogMiner can mine online redo logs. However, if the `CONTINUOUS_MINE` option is not specified, it is possible that the database is writing to the online redo log file at the same time that LogMiner is reading the online redo log file. If a log switch occurs while LogMiner is reading an online redo log file, the database will overwrite what LogMiner is attempting to read. The data that LogMiner returns if the file it is trying to read gets overwritten by the database is unpredictable.
- Keep the following in mind regarding starting and ending times or SCN ranges:

- If you specify neither a `startTime` nor a `startScn` parameter, LogMiner will set the `startScn` parameter to use the lowest SCN value from the redo log file that contains the oldest changes.
- If you specify both time and SCN values, LogMiner uses the SCN value or values and ignores the time values.
- If you specify starting and ending time or SCN values and they are found in the LogMiner redo log file list, then LogMiner mines the logs indicated by those values.
- If you specify starting and ending times or SCN values that are not in the LogMiner redo log file list, and you specify `DBMS_LOGMNR.START_LOGMNR` without the `CONTINUOUS_MINE` option, and you specify:
  - \* 0 for the `startTime` or `startScn` value, then the lowest SCN in the LogMiner redo log file list will be used as the `startScn`
  - \* A nonzero number for the `startTime` or `startScn` value, then an error is returned
  - \* 0 or a nonzero number for the `endTime` or `endScn` value, then the highest SCN in the LogMiner redo log file list will be used as the `endScn`
- If you specify starting and ending times or SCN values and they are not found in the LogMiner redo log file list, and you specify `DBMS_LOGMNR.START_LOGMNR` with the `CONTINUOUS_MINE` option, and you specify:
  - \* 0 for the `startTime` or `startScn` value, then an error is returned.
  - \* A `startTime` or `startScn` value that is greater than any value in the database's archived redo log files, then LogMiner starts mining in the online redo log file. LogMiner will continue to process the online redo log file until it finds a change at, or beyond, the requested starting point before it returns rows from the `V$LOGMNR_CONTENTS` view.
  - \* An `endTime` or `endScn` parameter value that indicates a time or SCN in the future, then LogMiner includes the online redo log files when it mines. When you query the `V$LOGMNR_CONTENTS` view, rows will be returned from this view as changes are made to the database, and will not stop until LogMiner sees a change beyond the requested ending point.
  - \* 0 for the `endTime` or `endScn` parameter value, then LogMiner includes the online redo log files when it mines. When you query the `V$LOGMNR_CONTENTS` view, rows will be returned from this view as changes are made to the database, and will not stop until you enter `CTL+C` or you terminate the PL/SQL cursor.





---

---

## DBMS\_LOGMNR\_D

The DBMS\_LOGMNR\_D package, one of a set of LogMiner packages, contains two subprograms:

- The BUILD procedure extracts the LogMiner data dictionary to either the redo log files or to a flat file. This information is saved in preparation for future analysis of redo log files using the LogMiner tool.
- The SET\_TABLESPACE procedure re-creates all LogMiner tables in an alternate tablespace.

The **LogMiner data dictionary** consists of the memory data structures and the database tables that are used to store and retrieve information about objects and their versions. It is referred to as the **LogMiner dictionary** throughout the LogMiner documentation.

**See Also:** *Oracle Database Utilities* for information regarding LogMiner.

This chapter contains the following topics:

- [Using DBMS\\_LOGMNR\\_D](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_LOGMNR\\_D Subprograms](#)

## Using DBMS\_LOGMNR\_D

This section contains the following topics, which relate to using the DBMS\_LOGMNR\_D package:

- [Overview](#)
- [Security Model](#)

## Overview

LogMiner requires a dictionary to translate object IDs into object names when it returns redo data to you. LogMiner gives you three options for supplying the dictionary:

- Using the online catalog
- Extracting a LogMiner dictionary to the redo log files
- Extracting a LogMiner dictionary to a flat file

Use the `BUILD` procedure to extract the LogMiner dictionary to the redo log files or a flat file. If you want to specify the online catalog as the dictionary source, you do so when you start LogMiner with the `DBMS_LOGMNR.START_LOGMNR` package.

Use the `SET_TABLESPACE` procedure if you want LogMiner tables to use a tablespace other than the default `SYSAUX` tablespace.

**See Also:** [DBMS\\_LOGMNR](#) for information on the package subprograms used in running a LogMiner session.

## Security Model

You must have the EXECUTE\_CATALOG\_ROLE role to use the DBMS\_LOGMNR\_D package.

## Summary of DBMS\_LOGMNR\_D Subprograms

**Table 85-1 DBMS\_LOGMNR\_D Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">BUILD Procedure</a> on page 85-6	Extracts the LogMiner dictionary to either a flat file or one or more redo log files
<a href="#">SET_TABLESPACE Procedure</a> on page 85-9	Re-creates all LogMiner tables in an alternate tablespace

## BUILD Procedure

This procedure extracts the LogMiner data dictionary to either the redo log files or to a flat file.

### Syntax

```
DBMS_LOGMNR_D.BUILD (
    dictionary_filename IN VARCHAR2,
    dictionary_location IN VARCHAR2,
    options             IN NUMBER);
```

### Parameters

**Table 85–2 BUILD Procedure Parameters**

Parameter	Description
dictionary_filename	Specifies the name of the LogMiner dictionary file.
dictionary_location	Specifies the path to the LogMiner dictionary file directory.
options	Specifies that the LogMiner dictionary is written to either a flat file (STORE_IN_FLAT_FILE) or the redo log files (STORE_IN_REDO_LOGS).

### Exceptions

**Table 85–3 BUILD Procedure Exceptions**

Exception	Description
ora-01302	Dictionary build options are missing or incorrect. This error is returned under the following conditions: <ul style="list-style-type: none"> <li>■ If the value of the OPTIONS parameter is not one of the supported values (STORE_IN_REDO_LOGS, STORE_IN_FLAT_FILE) or is not specified</li> <li>■ If the STORE_IN_REDO_LOGS option is not specified and neither the dictionary_filename nor the dictionary_location parameter is specified</li> <li>■ If the STORE_IN_REDO_LOGS option is specified and either the dictionary_filename or the dictionary_location parameter is specified</li> </ul>
ora-01308	Initialization parameter UTL_FILE_DIR is not set.
ora-01336	Specified dictionary file cannot be opened. This error is returned under the following conditions: <ul style="list-style-type: none"> <li>■ The specified value for the dictionary_location does not exist.</li> <li>■ The UTL_FILE_DIR initialization parameter is not set to have access to the dictionary_location</li> <li>■ The dictionary file is read-only.</li> </ul>

### Usage Notes

- To extract the LogMiner dictionary to a flat file, you must supply a filename and location.

To extract the LogMiner dictionary to the redo log files, specify only the `STORE_IN_REDO_LOGS` option. The size of the LogMiner dictionary may cause it to be contained in multiple redo log files.

The combinations of parameters used result in the following behavior:

- If you do not specify any parameters, an error is returned.
- If you specify a filename and location, without any options, the LogMiner dictionary is extracted to a flat file with that name.
- If you specify a filename and location, as well as the `STORE_IN_FLAT_FILE` option, the LogMiner dictionary is extracted to a flat file with the specified name.
- If you do not specify a filename and location, but do specify the `STORE_IN_REDO_LOGS` option, the LogMiner dictionary is extracted to the redo log files.
- If you specify a filename and location, as well as the `STORE_IN_REDO_LOGS` option, an error is returned.
- If you do not specify a filename and location, but do specify the `STORE_IN_FLAT_FILE` option, an error is returned.
- Ideally, the LogMiner dictionary file will be created after all database dictionary changes have been made and prior to the creation of any redo log files that are to be analyzed. As of Oracle9i release 1 (9.0.1), you can use LogMiner to dump the LogMiner dictionary to the redo log files or a flat file, perform DDL operations, and dynamically apply the DDL changes to the LogMiner dictionary.
- Do not run the `DBMS_LOGMNR_D.BUILD` procedure if there are any ongoing DDL operations.
- The database must be open when you run the `DBMS_LOGMNR_D.BUILD` procedure.
- When extracting a LogMiner dictionary to a flat file, the procedure queries the dictionary tables of the current database and creates a text-based file containing the contents of the tables. To extract a LogMiner dictionary to a flat file, the following conditions must be met:
  - You must specify a directory for use by the PL/SQL procedure. To do so, set the initialization parameter `UTL_FILE_DIR` in the initialization parameter file. For example:
 

```
UTL_FILE_DIR = /oracle/dictionary
```

After setting the parameter, you must shut down and restart the database for this parameter to take effect. If you do not set this parameter, the procedure will fail.
  - You must ensure that no DDL operations occur while the LogMiner dictionary build is running. Otherwise, the LogMiner dictionary file may not contain a consistent snapshot of the database dictionary.

Be aware that the `DDL_DICT_TRACKING` option to the `DBMS_LOGMNR.START_LOGMNR` procedure is not supported for flat file dictionaries created prior to Oracle9i. If you attempt to use the `DDL_DICT_TRACKING` option with a LogMiner database extracted to a flat file prior to Oracle9i, the ORA-01330 error (problem loading a required build table) is returned.

- To extract a LogMiner dictionary file to the redo log files, the following conditions must be met:

- The `DBMS_LOGMNR_D.BUILD` procedure must be run on a system that is running Oracle9i or later.
- Archivelog mode must be enabled in order to generate usable redo log files.
- The `COMPATIBLE` parameter in the initialization parameter file must be set to 9.2.0 or higher.
- The database to which LogMiner is attached must be Oracle9i or later.

In addition, supplemental logging (at least the minimum level) should be enabled to ensure that you can take advantage of all the features that LogMiner offers. See *Oracle Database Utilities* for information about using supplemental logging with LogMiner.

## Examples

### Example 1: Extracting the LogMiner Dictionary to a Flat File

The following example extracts the LogMiner dictionary file to a flat file named `dictionary.ora` in a specified path (`/oracle/database`).

```
SQL> EXECUTE dbms_logmnr_d.build('dictionary.ora', -  
    '/oracle/database/', -  
    options => dbms_logmnr_d.store_in_flat_file);
```

### Example 2: Extracting the LogMiner Dictionary to the Redo Log Files

The following example extracts the LogMiner dictionary to the redo log files.

```
SQL> EXECUTE dbms_logmnr_d.build( -  
    options => dbms_logmnr_d.store_in_redo_logs);
```



## SET\_TABLESPACE Procedure

By default, all LogMiner tables are created to use the SYS\_AUX tablespace. However, it may be desirable to have LogMiner tables use an alternate tablespace. Use this procedure to move LogMiner tables to an alternate tablespace.

### Syntax

```
DBMS_LOGMNR_D.SET_TABLESPACE (
    new_tablespace      IN VARCHAR2);
```

### Parameters

**Table 85–4 SET\_TABLESPACE Parameter**

Parameter	Description
new_tablespace	A string naming a preexisting tablespace. To move all LogMiner tables to employ this tablespace, supply this parameter.

### Usage Notes

- Users upgrading from earlier versions of Oracle Database may find LogMiner tables in the SYSTEM tablespace. Oracle encourages such users to consider using the SET\_TABLESPACE procedure to move the tables to the SYS\_AUX tablespace once they are confident that they will not be downgrading to an earlier version of Oracle Database.
- Users of this routine must supply an existing tablespace.

**See Also:** *Oracle Database Concepts* and *Oracle Database SQL Language Reference* for information about tablespaces and how to create them

### Example: Using the DBMS\_LOGMNR\_D.SET\_TABLESPACE Procedure

The following example shows the creation of an alternate tablespace and execution of the DBMS\_LOGMNR\_D.SET\_TABLESPACE procedure.

```
SQL> CREATE TABLESPACE logmnrts$ datafile '/usr/oracle/dbs/logmnrts.f'
      SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

SQL> EXECUTE dbms_logmnr_d.set_tablespace('logmnrts$');
```



---

---

## DBMS\_LOGSTDBY

The DBMS\_LOGSTDBY package provides subprograms for configuring and managing the logical standby database environment.

**See Also:** *Oracle Data Guard Concepts and Administration* for more information about SQL Apply and logical standby databases

This chapter contains the following topics:

- [Using DBMS\\_LOGSTDBY](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_LOGSTDBY Subprograms](#)

---

## Using DBMS\_LOGSTDBY

This section contains topics which relate to using the DBMS\_LOGSTDBY package.

- [Overview](#)
- [Security Model](#)

## Overview

The DBMS\_LOGSTDBY package helps you manage the SQL Apply (logical standby database) environment. The subprograms in the DBMS\_LOGSTDBY package help you to accomplish the following main objectives:

- Manage configuration parameters used by SQL Apply.  
For example, controlling how transactions are applied on the logical standby database, how much shared pool is used, and how many processes are used by SQL Apply to mine and apply the changes.
- Ensure an appropriate level of supplemental logging is enabled, and a LogMiner dictionary is built correctly for logical standby database creation.
- Provide a way to skip the application of changes to selected tables or entire schemas in the logical standby database, and specify ways to handle exceptions encountered by SQL Apply.
- Allow controlled access to tables in the logical standby database that may require maintenance.

## Security Model

You must have the DBA role to use the DBMS\_LOGSTDBY package.

A prototype role, LOGSTDBY\_ADMINISTRATOR, is created by default with RESOURCE, and EXECUTE privileges on DBMS\_LOGSTDBY. If you choose to use this role, consider granting ALTER DATABASE and ALTER SESSION privileges to the role so that the grantee can start and stop SQL Apply and can enable and disable the database guard.

The procedures associated with skipping transactions (SKIP and UNSKIP, SKIP\_ERROR and UNSKIP\_ERROR, and SKIP\_TRANSACTION and UNSKIP\_TRANSACTION) all require DBA privileges to execute because their scope may contain wildcard schemas. Oracle recommends that where SKIP procedures are specified, these be owned by a secure account with appropriate privileges on the schemas they act on (for example, SYS).

## Summary of DBMS\_LOGSTDBY Subprograms

**Table 86–1 DBMS\_LOGSTDBY Package Subprograms**

Subprogram	Description
<a href="#">APPLY_SET Procedure</a> on page 86-7	Sets the values of various parameters that configure and maintain SQL Apply
<a href="#">APPLY_UNSET Procedure</a> on page 86-10	Restores the default values of various parameters that configure and maintain SQL Apply
<a href="#">BUILD Procedure</a> on page 86-11	Ensures supplemental logging is enabled properly and builds the LogMiner dictionary
<a href="#">INSTANTIATE_TABLE Procedure</a> on page 86-12	Creates and populates a table in the standby database from a corresponding table in the primary database
<a href="#">IS_APPLY_SERVER Function</a> on page 86-14	This function returns TRUE if it is executed from PL/SQL in the context of a logical standby apply server process. This function is used in conjunction with triggers that have the <code>fire_once</code> parameter in the <code>DBMS_DDL.SET_TRIGGER FIRING_PROPERTY</code> subprogram set to FALSE (the default is TRUE). Such triggers are executed when the relevant target is updated by an apply process. This function can be used within the body of the trigger to ensure that the trigger takes different (or no) actions on the primary or on the standby.
<a href="#">MAP_PRIMARY_SCN Function</a> on page 86-15	Maps an SCN relevant to the primary database to a corresponding SCN at the logical standby database. The mapped SCN is conservative in nature, and can thus be used to flash back the logical standby database to compensate for a flashback database operation performed at the primary database.
<a href="#">PREPARE_FOR_NEW_PRIMARY Procedure</a> on page 86-16	Used after a failover, this procedure ensures a local logical standby database that was not involved in the failover has not processed more redo than the new primary database and reports the set of archive redo log files that must be replaced to ensure consistency
<a href="#">PURGE_SESSION Procedure</a> on page 86-18	Identifies the archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply
<a href="#">REBUILD Procedure</a> on page 86-19	Records relevant metadata (including the LogMiner dictionary) in the redo stream in case a database that has recently changed its role to a primary database following a failover operation fails to do so during the failover process
<a href="#">SET_TABLESPACE Procedure</a> on page 86-20	Moves metadata tables required by SQL Apply to the user-specified tablespace. By default, the metadata tables are created in the <code>SYSAUX</code> tablespace.
<a href="#">SKIP Procedure</a> on page 86-21	Specifies rules that control database operations that should not be applied to the logical standby database
<a href="#">SKIP_ERROR Procedure</a> on page 86-29	Specifies rules regarding what action to take upon encountering errors

**Table 86–1 (Cont.) DBMS\_LOGSTDBY Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SKIP_TRANSACTION Procedure</a> on page 86-33	Specifies transactions that should not be applied on the logical standby database. Be careful in using this procedure, because not applying specific transactions may cause data corruption at the logical standby database.
<a href="#">UNSKIP Procedure</a> on page 86-35	Deletes rules specified by the SKIP procedure
<a href="#">UNSKIP_ERROR Procedure</a> on page 86-37	Deletes rules specified by the SKIP_ERROR procedure
<a href="#">UNSKIP_TRANSACTION Procedure</a> on page 86-39	Deletes rules specified by the SKIP_TRANSACTION procedure



## APPLY\_SET Procedure

Use this procedure to set values of parameters that configure and manage SQL Apply in a logical standby database environment. All parameters, except for `PRESERVE_COMMIT_ORDER`, can be changed without having to stop SQL Apply.

### Syntax

```
DBMS_LOGSTDBY.APPLY_SET (
    inname          IN VARCHAR,
    value           IN VARCHAR);
```

### Parameters

**Table 86–2 APPLY\_SET Procedure Parameters**

Parameter	Description
<code>APPLY_SERVERS</code>	Controls the number of <code>APPLIER</code> processes used to apply changes. The maximum number allowed is 1024, provided the <code>MAX_SERVERS</code> parameter is set to accommodate this.
<code>EVENT_LOG_DEST</code>	<p>Controls where SQL Apply records the occurrence of an interesting event. It takes the following values:</p> <ul style="list-style-type: none"> <li>▪ <code>DEST_ALL</code> - All events will be recorded in the <code>DBA_LOGSTDBY_EVENTS</code> view and in the alert log.</li> <li>▪ <code>DEST_EVENTS_TABLE</code> - All events that contain information about user data will be recorded only in the <code>DBA_LOGSTDBY_EVENTS</code> view. This is the default value.</li> </ul> <p>For example, if SQL Apply receives an <code>ORA-1403</code> error, the whole event is recorded in the <code>DBA_LOGSTDBY_EVENTS</code> view. Whereas, the alert log records only that SQL Apply stopped because of <code>ORA-1403</code>. No information regarding the user table or offending statement is logged in the alert log. However, if you stop the SQL Apply engine, it gets recorded in both the <code>DBA_LOGSTDBY_EVENTS</code> view and in the alert log.</p> <p>Note that this parameter affects the behavior of the following parameters: <code>RECORD_APPLIED_DDL</code>, <code>RECORD_SKIP_DDL</code>, <code>RECORD_SKIP_ERRORS</code>, and <code>RECORD_UNSUPPORTED_OPERATIONS</code>. For example, if <code>RECORD_APPLIED_DDL</code> is set to <code>TRUE</code>, but <code>EVENT_LOG_DEST</code> is set to <code>DEST_EVENTS_TABLE</code>, then the applied DDL string will only be recorded in the <code>DBA_LOGSTDBY_EVENTS</code> view.</p>
<code>LOG_AUTO_DEL_RETENTION_TARGET</code>	This parameter setting is only meaningful if <code>LOG_AUTO_DELETE</code> has been set to <code>TRUE</code> . The value you supply for this parameter controls how long (in minutes) a remote archived log that is received from the primary database will be retained at the logical standby database once all redo records contained in the log have been applied at the logical standby database. The default value is 1440 minutes.
<code>LOG_AUTO_DELETE</code>	Automatically deletes foreign archived redo log files as soon as they have been applied on the logical standby database. By default, a foreign archived redo log file is not deleted until 24 hours (the default value of <code>LOG_AUTO_DEL_RETENTION_TARGET</code> parameter) after it has been applied at the logical standby database. Set to <code>TRUE</code> to enable automatic deletion of archived redo log files. Set to <code>FALSE</code> to disable automatic deletion. The default value is <code>TRUE</code> .

**Table 86–2 (Cont.) APPLY\_SET Procedure Parameters**

Parameter	Description
MAX_EVENTS_RECORDED	Number of recent events that will be visible through the DBA_LOGSTDBY_EVENTS view. To record all events encountered by SQL Apply, use the DBMS_LOGSTDBY.MAX_EVENTS constant as the number value. The default value is 10,000.
MAX_SERVERS	Number of processes that SQL Apply uses to read and apply redo. The default value is 9. The maximum number allowed is 2048.
MAX_SGA	Number of megabytes from shared pool in System Global Area (SGA) that SQL Apply will use. The default value is 30 megabytes or one quarter of the value set for SHARED_POOL_SIZE, whichever is lower. The maximum size allowed is 4095 megabytes.
PREPARE_SERVERS	Controls the number of PREPARER processes used to prepare changes. The maximum number allowed is 1024, provided the MAX_SERVERS parameter is set to accommodate this.
PRESERVE_COMMIT_ORDER	<p>TRUE: Transactions are applied to the logical standby database in the exact order in which they were committed on the primary database. This is the default parameter setting.</p> <p>FALSE: Transactions containing non-overlapping sets of rows may be committed in a different order than they were committed on the primary database.</p> <p>Regardless of the level chosen, modifications done to the same row are always applied in the same order as they happened on the primary database. See the Usage Notes for details and recommendations.</p> <p>You cannot modify this parameter while SQL Apply is running.</p>
RECORD_APPLIED_DDL	<p>Controls whether DDL statements that have been applied to the logical standby database are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values:</p> <p>TRUE: Indicates that DDL statements applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.</p> <p>FALSE: Indicates that applied DDL statements are not recorded. This is the default parameter setting.</p>
RECORD_SKIP_DDL	<p>Controls whether skipped DDL statements are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values:</p> <p>TRUE: Skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting.</p> <p>FALSE: Skipped DDL statements are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.</p>

**Table 86–2 (Cont.) APPLY\_SET Procedure Parameters**

Parameter	Description
RECORD_SKIP_ERRORS	Controls whether skipped errors (as described by the SKIP_ERROR procedure) are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values:  TRUE: Skipped errors are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting.  FALSE: Skipped errors are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.
RECORD_UNSUPPORTED_OPERATIONS	Captures information about transactions running on the primary database that will not be supported by a logical standby database. This procedure records its information as events in the DBA_LOGSTDBY_EVENTS table.

If a parameter is changed while SQL Apply is running, the change will take effect at some point in the future. In such a case, an informational row is inserted into the DBA\_LOGSTDBY\_EVENTS view at the time the parameter change takes effect.

Additionally, if you are modifying a parameter while SQL Apply is running on an Oracle RAC configuration, you must be connected to the same instance where SQL Apply is running.

## Exceptions

**Table 86–3 APPLY\_SET Procedure Exceptions**

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

## Usage Notes

- Use the APPLY\_UNSET procedure to restore the default settings of a parameter.
- See *Oracle Data Guard Concepts and Administration* for help with tuning SQL Apply and for information about setting appropriate values for different parameters.

## Examples

To record DDLs in the DBA\_LOGSTDBY\_EVENTS view and in the alert log, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('RECORD_APPLIED_DDL', TRUE);
```

## APPLY\_UNSET Procedure

Use the `APPLY_UNSET` procedure to restore the default values of the parameters that you changed with the `APPLY_SET` procedure.

### Syntax

```
DBMS_LOGSTDBY.APPLY_UNSET (
    inname          IN VARCHAR);
```

### Parameters

The parameter information for the `APPLY_UNSET` procedure is the same as that described for the `APPLY_SET` procedure. See [Table 86–2](#) for complete parameter information.

### Exceptions

**Table 86–4** *APPLY\_UNSET Procedure Exceptions*

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

### Usage Notes

- Use the `APPLY_SET` procedure to specify a nondefault value for a parameter.

### Examples

If you previously specified that applied DDLs show up in the `DBA_LOGSTDBY_EVENTS` view and the alert log, you can restore the default behavior of SQL Apply regarding applied DDL statements with the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('RECORD_APPLIED_DDL');
```

## BUILD Procedure

Use this procedure on the primary database to record relevant metadata (LogMiner dictionary) information in the redo log, which will subsequently be used by SQL Apply. This procedure will enable database-wide primary- and unique-key supplemental logging, if necessary.

---

**Note:** In databases created using Oracle Database 11g release 2 (11.2) or later, supplemental logging information is automatically propagated to any existing physical standby databases. However, for databases in earlier releases, or if the database was created using an earlier release and then upgraded to 11.2, you must check whether supplemental logging is enabled at the physical standby(s) if it is also enabled at the primary database. If it is not enabled at the physical standby(s), then before performing a switchover or failover, you must enable supplemental logging on all existing physical standby databases. To do so, issue the following SQL command on each physical standby:

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX) COLUMNS;
```

If you do not do this, then any logical standby that is also in the same Data Guard configuration will be unusable if a switchover or failover is performed to one of the physical standby databases. If a switchover or failover has already occurred and supplemental logging was not enabled, then you must recreate all logical standby databases.

---

## Syntax

```
DBMS_LOGSTDBY.BUILD;
```

## Usage Notes

- Supplemental log information includes extra information in the redo logs that uniquely identifies a modified row in the logical standby database, and also includes information that helps efficient application of changes to the logical standby database.
- LogMiner dictionary information allows SQL Apply to interpret data in the redo logs.
- DBMS\_LOGSTDBY.BUILD should be run only once for each logical standby database you want to create. You do not need to use DBMS\_LOGSTDBY.BUILD for each Oracle RAC instance.
- DBMS\_LOGSTDBY.BUILD waits for all transactions (including distributed transactions) that are active at the time of the procedure invocation to complete before returning. See *Oracle Database Administrator's Guide* for information about how to handle in-doubt transactions.

## Examples

To build the LogMiner dictionary in the redo stream of the primary database and to record additional information so that a logical standby database can be instantiated, issue the following SQL statement at the primary database

```
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```

## INSTANTIATE\_TABLE Procedure

This procedure creates and populates a table in the standby database from a corresponding table in the primary database. The table requires the name of the database link (`dblink`) as an input parameter. If the table already exists in the logical standby database, it will be dropped and re-created based on the table definition at the primary database. This procedure only brings over the data associated with the table, and not the associated indexes and constraints.

Use the `INSTANTIATE_TABLE` procedure to:

- Add a table to a standby database.
- Re-create a table in a standby database.

### Syntax

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE (
    schema_name      IN VARCHAR2,
    table_name       IN VARCHAR2,
    dblink           IN VARCHAR2);
```

### Parameters

**Table 86–5** INSTANTIATE\_TABLE Procedure Parameters

Parameter	Description
<code>schema_name</code>	Name of the schema
<code>table_name</code>	Name of the table to be created or re-created in the standby database
<code>dblink</code>	Name of the database link account that has privileges to read and lock the table in the primary database, as well as the <code>SELECT_CATALOG_ROLE</code> on the primary database

### Exceptions

**Table 86–6** INSTANTIATE\_TABLE Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16236	Logical Standby metadata operation in progress
ORA-16276	Specified database link does not correspond to primary database
ORA-16277	Specified table is not supported by logical standby database
ORA-16278	Specified table has a multi-object skip rule defined

### Usage Notes

- Use this procedure to create and populate a table in a way that keeps the data on the standby database transactionally consistent with the primary database.
- This table will not be synchronized with the rest of the tables being maintained by SQL Apply and SQL Apply will not start to maintain it until SQL Apply encounters redo that occurred after the table was instantiated from the primary. The SCN at which the table was instantiated from the primary database is available in the `DBA_LOGSTDBY_EVENTS` view.

- The specified table must be a table that is supported by logical standby (that is, it does not appear in the `DBA_LOGSTDBY_UNSUPPORTED_TABLES` view on the primary database).
- If there are any skip rules that specifically name this table (without any wildcards), those skip rules will be dropped as part of `INSTANTIATE_TABLE`, so that the table will be properly maintained by SQL Apply in the future. If there are skip rules that indirectly reference this table (match a skip rule with a wildcard in the `schema_name` or `table_name`, and have a `TABLE`, `DML`, or `SCHEMA_DDL` statement type), `INSTANTIATE_TABLE` will fail with an ORA-16278 error. Any multiobject skip rules that pertain to the table must be dropped or changed before re-attempting the `INSTANTIATE_TABLE` call.

## Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE (-  
    SCHEMA_NAME => 'HR', TABLE_NAME => 'EMPLOYEES', -  
    DBLINK => 'INSTANTIATE_TBL_LINK');
```

## IS\_APPLY\_SERVER Function

This function returns `TRUE` if it is executed from PL/SQL in the context of a logical standby apply server process. This function is used in conjunction with triggers that have the `fire_once` parameter in the `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY` subprogram set to `FALSE` (the default is `TRUE`). Such triggers are executed when the relevant target is updated by an apply process. This function can be used within the body of the trigger to ensure that the trigger takes different (or no) actions on the primary or on the standby.

**See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY` subprogram.

### Syntax

```
DBMS_LOGSTDBY.IS_APPLY_SERVER  
RETURN BOOLEAN;
```

### Parameters

None



## MAP\_PRIMARY\_SCN Function

Returns an SCN on the standby that predates the supplied SCN from the primary database by at least 5 minutes. This function can be used to determine a safe SCN to use in a compensating flashback database operation at the logical standby database, following a flashback database operation or a point-in-time recovery operation at the primary database.

### Syntax

```
DBMS_LOGSTDBY.MAP_PRIMARY_SCN(primary_scn NUMBER) RETURN NUMBER;
```

### Exceptions

**Table 86–7 MAP\_PRIMARY\_SCN Function Exceptions**

Exception	Description
ORA-20001	Primary SCN is before mapped range
ORA-20002	SCN mapping requires PRESERVE_COMMIT_ORDER to be TRUE

### Usage Notes

Use this function to get a conservative SCN at the logical standby database that corresponds to an SCN at the primary database. This function is useful in the context of doing compensating flashback database operations at the logical standby following a flashback database or a point-in-time recovery operation done at the primary database.

## PREPARE\_FOR\_NEW\_PRIMARY Procedure

The `PREPARE_FOR_NEW_PRIMARY` procedure must be invoked at a logical standby database following a failover if that standby database was not the target of the failover operation. Such a standby database must process the exact same set of redo logs processed at the new primary database. This routine ensures that the local logical standby database has not processed more redo than the new primary database and reports the set of archive logs that must be replaced to ensure consistency. The set of replacement logs will be reported in the `alert.log`. These logs must be copied to the logical standby and registered using the `ALTER DATABASE REGISTER LOGICAL LOGFILE` statement.

### Syntax

```
DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (
    FORMER_STANDBY_TYPE    IN VARCHAR2,
    DBLINK                  IN VARCHAR2);
```

### Parameters

**Table 86–8** *PREPARE\_FOR\_NEW\_PRIMARY Procedure Parameters*

Parameter	Description
FORMER_STANDBY_TYPE	The type of standby database that was the target of the failover operation to become the new primary database. Valid values are 'PHYSICAL' if the new primary was formerly a physical standby, and 'LOGICAL' if the new primary database was formerly a logical standby database.
DBLINK	The name of a database link to the new primary database

### Exceptions

**Table 86–9** *PREPARE\_FOR\_NEW\_PRIMARY Procedure Exceptions*

Exception	Description
ORA-16104	Invalid Logical Standby option.
ORA-16109	Failed to apply log data from previous primary.

### Usage Notes

- This routine is intended only for logical standby systems.
- This routine will fail if the new primary database was formerly a logical standby database and the LogMiner dictionary build has not completed successfully.
- Log files displayed in the alert log will be referred to as *terminal logs*. Users should keep in mind that file paths are relative to the new primary database and may not resolve locally.
- Upon manual registration of the terminal logs, users should complete the process by calling either `START LOGICAL STANDBY APPLY` if the new primary database was formerly a physical standby database or `START LOGICAL STANDBY APPLY NEW PRIMARY` if the new primary database was formerly a logical standby database.
- See the alert log for more details regarding the reasons for any exception.

**Examples**

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY ( -  
        FORMER_STANDBY_TYPE => 'LOGICAL',      -  
        DBLINK => 'dblink_to_newprimary');
```

## PURGE\_SESSION Procedure

Identifies all archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply. Once identified, you can issue operating system commands to delete some or all of the unnecessary archived redo log files.

### Syntax

```
DBMS_LOGSTDBY.PURGE_SESSION;
```

### Exceptions

**Table 86–10 PURGE\_SESSION Procedure Exceptions**

Exception	Description
ORA-01309	Invalid session

### Usage Notes

- This procedure does not delete the archived redo log files. You must issue operating system commands to delete unneeded files.
- This procedure updates the DBA\_LOGMNR\_PURGED\_LOG view that displays the archived redo log files that have been applied to the logical standby database.
- In Oracle Database 10g Release 2, metadata related to the archived redo log files (and the actual archived redo log files) are purged automatically based on the default setting of the LOG\_AUTO\_DELETE parameter described in the DBMS\_LOGSTDBY.APPLY\_SET procedure described on page 86-7.

### Example

To identify and remove unnecessary files:

1. Enter the following statement on the logical standby database:

```
SQL> EXECUTE DBMS_LOGSTDBY.PURGE_SESSION;
```

2. Query the DBA\_LOGMNR\_PURGED\_LOG view to list the archived redo log files that can be removed:

```
SQL> SELECT * FROM DBA_LOGMNR_PURGED_LOG;
```

```
FILE_NAME
-----
/boston/arc_dest/arc_1_40_509538672.log
/boston/arc_dest/arc_1_41_509538672.log
/boston/arc_dest/arc_1_42_509538672.log
/boston/arc_dest/arc_1_43_509538672.log
/boston/arc_dest/arc_1_44_509538672.log
/boston/arc_dest/arc_1_45_509538672.log
/boston/arc_dest/arc_1_46_509538672.log
/boston/arc_dest/arc_1_47_509538672.log
```

3. Use operating system-specific commands to delete archived redo log files from the file system.

## REBUILD Procedure

This procedure is used if a database that has recently changed its role to a primary database following a failover operation fails to record relevant metadata (including the LogMiner dictionary) in the redo stream required for other logical standby databases.

### Syntax

```
DBMS_LOGSTDBY.REBUILD;
```

### Usage Notes

- LogMiner dictionary information is logged in the redo log files.
- The standby redo log files (if present) are archived.

### Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.REBUILD;
```

## SET\_TABLESPACE Procedure

Moves metadata tables required by SQL Apply to the user-specified tablespace. By default, the metadata tables are created in the `SYSAUX` tablespace. SQL Apply cannot be running when you invoke this procedure.

### Syntax

```
DBMS_LOGSTDBY.SET_TABLESPACE (
    NEW_TABLESPACE IN VARCHAR2)
```

### Parameters

**Table 86–11 SET\_TABLESPACE Procedure Parameters**

Parameter	Description
<code>NEW_TABLESPACE</code>	Name of the new tablespace where metadata tables will reside.

### Exceptions

**Table 86–12 SET\_TABLESPACE Procedure Exceptions**

Exception	Description
<code>ORA-16103</code>	Logical Standby apply must be stopped to allow this operation
<code>ORA-16236</code>	Logical Standby metadata operation in progress

### Examples

To move metadata tables to a new tablespace named `LOGSTDBY_TBS`, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.SET_TABLESPACE (new_tablespace => 'LOGSTDBY_TBS');
```

## SKIP Procedure

The `SKIP` procedure can be used to define rules that will be used by SQL Apply to skip the application of certain changes to the logical standby database. For example, the `SKIP` procedure can be used to skip changes to a subset of tables in the logical standby database. It can also be used to specify DDL statements that should not be applied at the logical standby database or should be modified before they are applied in the logical standby database. One reason why a DDL statement may need to be modified is to accommodate a different directory structure on the logical standby database.

### Syntax

```
DBMS_LOGSTDBY.SKIP (
    stmt                IN VARCHAR2,
    schema_name        IN VARCHAR2 DEFAULT NULL,
    object_name        IN VARCHAR2 DEFAULT NULL,
    proc_name          IN VARCHAR2 DEFAULT NULL,
    use_like           IN BOOLEAN DEFAULT TRUE,
    esc                IN CHAR1 DEFAULT NULL);
```

### Parameters

**Table 86–13** SKIP Procedure Parameters

Parameter	Description
<code>stmt</code>	<p>Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration since keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. <a href="#">Table 86–14</a> shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.</p> <p>The keyword <code>PL/SQL</code> is used for the execution of Oracle-supplied packages which are supported for replication. See <i>Oracle Data Guard Concepts and Administration</i> for information about supported packages.</p>
<code>schema_name</code>	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the <code>stmt</code> parameter. If not applicable, this value must be set to <code>NULL</code> .
<code>object_name</code>	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the <code>stmt</code> . If not applicable, this value must be set to <code>NULL</code> .

**Table 86–13 (Cont.) SKIP Procedure Parameters**

Parameter	Description
<code>proc_name</code>	<p>Name of a stored procedure to call when SQL Apply determines that a particular statement matches the filter defined by the <code>stmt</code>, <code>schema_name</code>, and <code>object_name</code> parameters. Specify the procedure in the following format:</p> <pre>' schema . package . procedure '</pre> <p>This procedure returns a value that directs SQL Apply to perform one of the following: execute the statement, skip the statement, or execute a replacement statement.</p> <p>For DDLs, SQL Apply calls the stored procedure with the following call signature:</p> <ul style="list-style-type: none"> <li>▪ <code>IN STATEMENT VARCHAR2</code> -- The SQL statement that matches the filter</li> <li>▪ <code>IN STATEMENT_TYPE VARCHAR2</code> -- The <code>stmt</code> of the filter</li> <li>▪ <code>IN SCHEMA VARCHAR2</code> -- The <code>schema_name</code> of the filter, if applicable</li> <li>▪ <code>IN NAME VARCHAR2</code> -- The <code>object_name</code> of the filter, if applicable</li> <li>▪ <code>IN XIDUSN NUMBER</code> -- Transaction ID part 1</li> <li>▪ <code>IN XIDSLT NUMBER</code> -- Transaction ID part 2</li> <li>▪ <code>IN XIDSQN NUMBER</code> -- Transaction ID part 3</li> <li>▪ <code>OUT SKIP_ACTION NUMBER</code> -- Action to be taken by SQL Apply upon completion of this routine. Valid values are: <ul style="list-style-type: none"> <li><code>SKIP_ACTION_APPLY</code> -- Execute the statement</li> <li><code>SKIP_ACTION_SKIP</code> -- Skip the statement</li> <li><code>SKIP_ACTION_ERROR</code> -- Halt apply so the DBA can take appropriate steps (for example, take compensating action)</li> <li><code>SKIP_ACTION_REPLACE</code> -- Execute the replacement statement supplied in the <code>NEW_STATEMENT</code> output parameter</li> </ul> </li> </ul> <p>For PL/SQL, SQL Apply calls the stored procedure with the following call signature:</p> <ul style="list-style-type: none"> <li>▪ <code>IN STATEMENT VARCHAR2</code> -- The SQL statement that matches the filter</li> <li>▪ <code>IN PACKAGE_SCHEMA VARCHAR2</code> -- The schema of the package being skipped (for example, <code>SYS</code> or <code>XDB</code>)</li> <li>▪ <code>IN PACKAGE_NAME VARCHAR2</code> -- The name of the package being skipped (for example, <code>DEMS_RLS</code>)</li> <li>▪ <code>IN PROCEDURE_NAME VARCHAR2</code> -- The name of the procedure being skipped (for example, <code>ADD_POLICY</code>)</li> <li>▪ <code>IN CURRENT_SCHEMA VARCHAR2</code> -- The name of the current schema in which the PL/SQL was executed on the primary</li> <li>▪ <code>IN XIDUSN NUMBER</code> -- Transaction ID part 1</li> <li>▪ <code>IN XIDSLT NUMBER</code> -- Transaction ID part 2</li> <li>▪ <code>IN XIDSQN NUMBER</code> -- Transaction ID part 3</li> <li>▪ <code>IN EXIT_STATUS</code> -- 0 (Zero) if the PL/SQL succeeded on the primary, or 1</li> </ul>



**Table 86–13 (Cont.) SKIP Procedure Parameters**

Parameter	Description
proc_name (cont.)	<ul style="list-style-type: none"> <li>OUT SKIP_ACTION NUMBER -- Action to be taken by SQL Apply upon completion of this routine. Valid return values are:  SKIP_ACTION_APPLY -- Execute the statement  SKIP_ACTION_APPLY -- Execute the statement  SKIP_ACTION_ERROR – Raise an error which halts apply so that the DBA can take appropriate steps</li> </ul> <p><b>Note 1:</b> SKIP_ACTION_REPLACE is not supported for PL/SQL.</p> <p><b>Note 2:</b> SQL Apply calls the skip handler when the procedure's exit is processed.</p> <p><b>Note 3:</b> The use_like parameter must be set to FALSE for PL/SQL since wildcarding PL/SQL is not supported.</p> <p>The following example shows how to have a conditional skip rule on DBMS_RLS.DROP_POLICY:</p> <pre> Create or replace procedure sec_mgr.skip_drop_policy ( statement      in varchar2, pkgown         in varchar2, pkgname        in varchar2, procnm         in varchar2, cuser          in varchar2, xidusn         in number, xidslt         in number, xidsgn         in number, exstatus       in number, skip_action    out number) Is Begin   If 0 = exstatus Then     Insert Into sec_mgr.logit Values       ('Success: '    pkgown    '.'    pkgname    '.'    procnm    ' by '    cuser);      If cuser != 'TESTSCHEMA' Then       skip_action := DBMS_LOGSTDBY.SKIP_ACTION_APPLY;     Else       skip_action := DBMS_LOGSTDBY.SKIP_ACTION_SKIP;     End If;   End If; End skip_drop_policy;  EXECUTE DBMS_LOGSTDBY.SKIP( -   stmt =&gt; 'PL/SQL', -   schema_name =&gt; 'SYS', -   object_name =&gt; 'DBMS_RLS.DROP_POLICY', -   proc_name =&gt; 'SEC_MGR.SKIP_DROP_POLICY' -   use_like=&gt; FALSE); </pre>
use_like	<p>Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in the <i>Oracle Database SQL Language Reference</i>.</p>

**Table 86–13 (Cont.) SKIP Procedure Parameters**

Parameter	Description
esc	Identifies an escape character (such as the character "/" ) that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. See <i>Oracle Database SQL Language Reference</i> for more information about pattern matching.

## Usage Notes

- This procedure requires DBA privileges to execute.
- You cannot associate a stored procedure to be invoked in the context of a DML statement. For example, the following statement returns the ORA-16104 :  
invalid Logical Standby option requested error:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
      stmt => 'DML', -
      schema_name => 'HR', -
      object_name => 'EMPLOYEES', -
      proc_name => 'DML_HANDLER');
```

Also, if an event matches multiple rules either because of the use of wildcards while specifying the rule or because of a specification of overlapping rules. For example, if you specify a rule for the SCHEMA\_DDL event for the HR.EMPLOYEES table, and a rule for the ALTER TABLE event for the HR.EMPLOYEES table, only one of the matching procedures will be invoked (alphabetically, by procedure). In the following code example, consider the following rules:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
      stmt => 'SCHEMA DDL', -
      schema_name => 'HR', -
      object_name => 'EMPLOYEES', -
      proc_name => 'SCHEMA_DDL_HANDLER');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
      stmt => 'ALTER TABLE', -
      schema_name => 'HR', -
      object_name => 'EMPLOYEES', -
      proc_name => 'TABLE_ALTER_HANDLER');
```

On encountering an ALTER TABLE statement, the schema\_ddl\_handler procedure will be invoked because its name will be at the top of an alphabetically sorted list of procedures that are relevant to the statement.

Collisions on a rule set because of a specification containing wildcard entries are resolved in a similar fashion. For example, the rules in the following example will result in the empddl\_handler procedure being invoked upon encountering the ALTER TABLE HR.EMPLOYEES ADD COLUMN RATING NUMBER statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
      stmt => 'ALTER TABLE', -
      schema_name => 'HR', -
      object_name => 'EMP%', -
      proc_name => 'EMPDDL_HANDLER');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
      stmt => 'ALTER TABLE', -
      schema_name => 'HR', -
      object_name => 'EMPLOYEES', -
      proc_name => 'EMPLOYEE_DDL_HANDLER');
```

- Use the `SKIP` procedure with caution, particularly when skipping DDL statements. If a `CREATE TABLE` statement is skipped, for example, you must also specify other DDL statements that refer to that table in the `SKIP` procedure. Otherwise, the statements will fail and cause an exception. When this happens, SQL Apply stops running.
- Before calling the `SKIP` procedure, SQL Apply must be halted. Do this by issuing an `ALTER DATABASE STOP LOGICAL STANDBY APPLY` statement. Once all desired filters have been specified, issue an `ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE` statement to start SQL Apply using the new filter settings.
- See the `UNSKIP` procedure on page 86-35 for information about reversing (undoing) the settings of the `SKIP` procedure.
- For `USER` statements, the `SCHEMA_NAME` parameter will be the user and specify '%' for the `OBJECT_NAME` parameter.
- If the `PROC_NAME` parameter is supplied, it must already exist in `DBA_PROCEEDURES` and it must execute with `DEFINER` rights. If the procedure is declared with `INVOKER` rights, the `ORA-1031: insufficient privileges` message will be returned.
- If the procedure returns a `REPLACEMENT` statement, the `REPLACEMENT` statement will be executed using the `SYSTEM` and `OBJECT` privileges of the owner of the procedure.
- The PL/SQL block of a `SKIP` procedure cannot contain transaction control statements (for example, `COMMIT`, `ROLLBACK`, `SAVEPOINT`, and `SET CONSTRAINT`) unless the block is declared to be an autonomous transaction.

### Skip Statement Options

Table 86-14 lists the supported values for the `stmt` parameter of the `SKIP` procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. In addition, any of the SQL statements listed in the `sys.audit_actions` table (shown in the right column of Table 86-14) are also valid values. Note that keywords are generally defined by database object.

**Table 86-14 Supported Values for the `stmt` Parameter**

Keyword	Associated SQL Statements
There is no keyword for this group of SQL statements.	GRANT REVOKE ANALYZE TABLE ANALYZE INDEX ANALYZE CLUSTER
CLUSTER	AUDIT CLUSTER CREATE CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK CREATE PUBLIC DATABASE LINK DROP DATABASE LINK DROP PUBLIC DATABASE LINK

**Table 86–14 (Cont.) Supported Values for the stmt Parameter**

<b>Keyword</b>	<b>Associated SQL Statements</b>
DIMENSION	ALTER DIMENSION CREATE DIMENSION DROP DIMENSION
DIRECTORY <sup>1</sup>	CREATE DIRECTORY DROP DIRECTORY
DML	Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)
INDEX	ALTER INDEX CREATE INDEX DROP INDEX
NON_SCHEMA_DDL	<i>All DDL that does not pertain to a particular schema</i> <b>Note:</b> SCHEMA_NAME and OBJECT_NAME must be null
PL/SQL <sup>2</sup>	Execute Oracle-supplied package.
PROCEDURE <sup>3</sup>	ALTER FUNCTION ALTER PACKAGE ALTER PACKAGE BODY ALTER PROCEDURE CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PACKAGE BODY DROP PROCEDURE
PROFILE	ALTER PROFILE CREATE PROFILE DROP PROFILE
ROLE	ALTER ROLE CREATE ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	ALTER ROLLBACK SEGMENT CREATE ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SCHEMA_DDL	<i>All DDL statements that create, modify, or drop schema objects (for example: tables, indexes, and columns)</i> <b>Note:</b> SCHEMA_NAME and OBJECT_NAME must <i>not</i> be null
SEQUENCE	ALTER SEQUENCE CREATE SEQUENCE DROP SEQUENCE
SYNONYM	CREATE PUBLIC SYNONYM CREATE SYNONYM DROP PUBLIC SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT <i>SQL_statements</i> NOAUDIT <i>SQL_statements</i>

**Table 86–14 (Cont.) Supported Values for the `stmt` Parameter**

<b>Keyword</b>	<b>Associated SQL Statements</b>
TABLE	CREATE TABLE ALTER TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE ALTER TABLESPACE
TRIGGER	ALTER TRIGGER CREATE TRIGGER DISABLE ALL TRIGGERS DISABLE TRIGGER DROP TRIGGER ENABLE ALL TRIGGERS ENABLE TRIGGER
TYPE	ALTER TYPE ALTER TYPE BODY CREATE TYPE CREATE TYPE BODY DROP TYPE DROP TYPE BODY
USER	ALTER USER CREATE USER DROP USER
VIEW	CREATE VIEW DROP VIEW
VIEW	CREATE VIEW DROP VIEW

<sup>1</sup> All directory objects are owned by `SYS`, but for the purpose of filtering them with a skip directive the schema should be specified as `%`.

<sup>2</sup> See *Oracle Data Guard Concepts and Administration* for information about supported packages.

<sup>3</sup> Java schema objects (sources, classes, and resources) are considered the same as procedure for purposes of skipping (ignoring) SQL statements.

## Exceptions

**Table 86–15 DBMS\_LOGSTDBY.SKIP Procedure Exceptions**

<b>Exception</b>	<b>Description</b>
ORA-01031	Insufficient privileges: <ul style="list-style-type: none"> <li>■ Procedure used <code>INVOKER</code> rights</li> <li>■ Procedure needs <code>DBA</code> privileges</li> </ul>
ORA-16103	Logical standby apply must be stopped to allow this operation.
ORA-16104	Invalid logical standby option requested.
ORA-16203	"Unable to interpret <code>SKIP</code> procedure return values." Indicates that a <code>SKIP</code> procedure has either generated an exception or has returned ambiguous values. You can identify the offending procedure by examining the <code>DBA_LOGSTDBY_EVENTS</code> view.
ORA-16236	Logical standby metadata operation in progress.

## Examples

### Example 1 Skipping all DML and DDL changes made to a schema

The following example shows how to specify rules so that SQL Apply will skip both DDL and DML statements made to the HR schema.

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'SCHEMA DDL', -
    schema_name => 'HR', -
    object_name => '%', -
    proc_name => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'DML', -
    schema_name => 'HR', -
    object_name => '%', -
    proc_name => null);
```

### Example 2 Creating a procedure to handle different file system organization

For example, if the file system organization in the logical standby database is different than that in the primary database, you can write a `SKIP` procedure to handle DDL statements with file specifications transparently.

The following procedure can handle DDL statements as long as you follow a specific naming convention for the file specification string.

1. Create the `SKIP` procedure to handle tablespace DDL statements:

```
CREATE OR REPLACE PROCEDURE sys.handle_tbs_ddl (

    old_stmt IN VARCHAR2,
    stmt_typ IN VARCHAR2,
    schema IN VARCHAR2,
    name IN VARCHAR2,
    xidusn IN NUMBER,
    xidslt IN NUMBER,
    xidsqn IN NUMBER,
    action OUT NUMBER,
    new_stmt OUT VARCHAR2
) AS
BEGIN

    -- All primary file specification that contains a directory
    -- /usr/orcl/primary/dbs
    -- should go to /usr/orcl/standby directory specification

    new_stmt = replace(old_stmt,
        '/usr/orcl/primary/dbs',
        '/usr/orcl/standby');

    action := DBMS_LOGSTDBY.SKIP_ACTION_REPLACE;

EXCEPTION
    WHEN OTHERS THEN
        action := DBMS_LOGSTDBY.SKIP_ACTION_ERROR;
        new_stmt := NULL;
END handle_tbs_ddl;
```

2. Register the `SKIP` procedure with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'TABLESPACE', -
    proc_name => 'SYS.HANDLE_TBS_DDL');
```

## SKIP\_ERROR Procedure

Upon encountering an error, the logical standby database uses the criteria contained in this procedure to determine a course of action. The default action when a match is found is to skip the error and continue with applying changes. However, if a procedure is supplied, then `SKIP_ERROR` can take other actions depending on the situation. It can do nothing, which causes SQL Apply to stop, or it can change the error message text and stop SQL Apply, or it can actually skip the error.

### Syntax

```
DBMS_LOGSTDBY.SKIP_ERROR (
    stmt                IN VARCHAR2,
    schema_name         IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2 DEFAULT NULL,
    proc_name           IN VARCHAR2 DEFAULT NULL,
    use_like             IN BOOLEAN  DEFAULT NULL,
    esc                 IN CHAR1    DEFAULT NULL);
```

### Parameters

**Table 86–16** SKIP\_ERROR Procedure Parameters

Parameter	Description
<code>stmt</code>	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration because keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. <a href="#">Table 86–14</a> shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.
<code>schema_name</code>	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the <code>stmt</code> parameter. If not applicable, this value must be set to <code>NULL</code> .
<code>object_name</code>	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the <code>stmt</code> . If not applicable, this value must be set to <code>NULL</code> .

**Table 86–16 (Cont.) SKIP\_ERROR Procedure Parameters**

Parameter	Description
proc_name	<p>Name of a stored procedure to call when SQL Apply encounters an error and determines a particular statement matches the filter defined by the <code>stmt</code>, <code>schema_name</code>, and <code>object_name</code> parameters. Specify the procedure in the following format:</p> <pre>' "schema" . "package" . "procedure" '</pre> <p>This procedure returns an error message that directs SQL Apply to perform one of the following actions:</p> <ul style="list-style-type: none"> <li>■ Silently skip the error and continue with SQL Apply</li> <li>■ Replace the error message that would have been created with a custom one, and stop SQL Apply</li> <li>■ Do nothing, causing SQL Apply to stop and the original error message to be logged</li> </ul> <p>SQL Apply calls the stored procedure with the following call signature:</p> <ul style="list-style-type: none"> <li>■ IN STATEMENT VARCHAR(4000) -- The first 4K of the statement</li> <li>■ IN STATEMENT_TYPE VARCHAR2 -- The <code>stmt</code> of the filter</li> <li>■ IN SCHEMA VARCHAR2 -- The <code>schema_name</code> of the filter, if applicable</li> <li>■ IN NAME VARCHAR2 -- The <code>object_name</code> of the filter, if applicable</li> <li>■ IN XIDUSN NUMBER -- Transaction ID part 1</li> <li>■ IN XIDSLT NUMBER -- Transaction ID part 2</li> <li>■ IN XIDSQN NUMBER -- Transaction ID part 3</li> <li>■ IN ERROR VARCHAR(4000) -- Text of the original error to be recorded</li> <li>■ OUT NEW_ERROR VARCHAR(4000) -- Null or modified error text</li> </ul>
use_like	<p>Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The <code>use_like</code> parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in the <i>Oracle Database SQL Language Reference</i>.</p>
esc	<p>Identifies an escape character (such as the characters "%" or "_") that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. See <i>Oracle Database SQL Language Reference</i> for more information about pattern matching.</p>

## Usage Notes

- A stored procedure provided to the `SKIP_ERROR` procedure is called when SQL Apply encounters an error that could shut down the application of redo logs to the standby database.
- Running this stored procedure affects the error being written in the `STATUS` column of the `DBA_LOGSTDBY_EVENTS` table. The `STATUS_CODE` column remains unchanged. If the stored procedure is to have no effect, that is, apply will be



stopped, then the `NEW_ERROR` is written to the events table. To truly have no effect, set `NEW_ERROR` to `ERROR` in the procedure.

- If the stored procedure requires that a shutdown be avoided, then you must set `NEW_ERROR` to `NULL`.
- This procedure requires `DBA` privileges to execute.
- For `USER` statements, the `SCHEMA_NAME` parameter will be the user and you should specify `'%'` for the `OBJECT_NAME` parameter.
- If the `PROC_NAME` parameter is specified, it must already exist in `DBA_PROCEEDURES` and it must execute with `DEFINERS` rights. If the procedure is declared with `INVOKERS` rights, the `ORA-1031: insufficient privileges` message will be returned.
- The PL/SQL block of a `SKIP_ERROR` procedure cannot contain transaction control statements (for example: `COMMIT`, `ROLLBACK`, `SAVEPOINT`, and `SET CONSTRAINT`) unless the block is declared to be an autonomous transaction using the following syntax:

```
PRAGMA AUTONOMOUS_TRANSACTION
```

## Exceptions

**Table 86–17** *SKIP\_ERROR Procedure Exceptions*

Exception	Description
ORA-01031	Insufficient privileges: <ul style="list-style-type: none"> <li>■ Procedure used <code>INVOKER</code> rights</li> <li>■ Procedure needs <code>DBA</code> privileges</li> </ul>
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

## Example 1

The following example shows how to specify rules so that SQL Apply will skip any error raised from any `GRANT DDL` command.

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR('GRANT')
```

## Example 2

To skip errors on `GRANT` statements on `SYS` or `HR` schemas, define a procedure `handle_error_ddl` and register it. In the following example, assume that `handle_error_ddl` is a free-standing procedure in the `SYS` schema.

1. Create the error-handler procedure:

```
CREATE OR REPLACE PROCEDURE sys.handle_error_ddl (
  old_stmt IN VARCHAR2,
  stmt_type IN VARCHAR2,
  schema IN VARCHAR2,
  name IN VARCHAR2,
  xidusn IN NUMBER,
  xidslt IN NUMBER,
  xidsqn IN NUMBER,
  error IN VARCHAR2,
```

```
new_error OUT VARCHAR2
) AS
BEGIN
-- Default to what we already have
new_error := error;
-- Ignore any GRANT errors on SYS or HR schemas
IF INSTR(UPPER(old_stmt), 'GRANT') > 0
THEN
IF schema IS NULL
OR (schema IS NOT NULL AND
(UPPER(schema) = 'SYS' OR UPPER(schema) = 'HR' )
THEN
new_error := NULL;
-- record the fact that we just skipped an error on 'SYS' or 'HR' schemas
-- code not shown here
END IF;
END IF;
END handle_error_ddl;
/
```

## 2. Register the error handler with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR ( -
      statement => 'NON_SCHEMA_DDL', -
      schema_name => NULL, -
      object_name => NULL, -
      proc_name => 'SYS.HANDLE_ERROR_DDL');
```

## SKIP\_TRANSACTION Procedure

This procedure provides a way to skip (ignore) applying transactions to the logical standby database. You can skip specific transactions by specifying transaction identification information.

### Syntax

```
DBMS_LOGSTDBY.SKIP_TRANSACTION (
    xidusn          IN NUMBER,
    xidslt          IN NUMBER,
    xidsqn          IN NUMBER);
```

### Parameters

**Table 86–18** SKIP\_TRANSACTION Procedure Parameters

Parameter	Description
XIDUSN NUMBER	Transaction ID undo segment number of the transaction being skipped
XIDSLT NUMBER	Transaction ID slot number of the transaction being skipped
XIDSQN NUMBER	Transaction ID sequence number of the transaction being skipped

### Usage Notes

If SQL Apply stops due to a particular transaction (for example, a DDL transaction), you can specify that transaction ID and then continue to apply. You can call this procedure multiple times for as many transactions as you want SQL Apply to ignore.

---

**CAUTION:** `SKIP_TRANSACTION` is an inherently dangerous operation. Do not invoke this procedure unless you have examined the transaction in question through the `V$LOGMNR_CONTENTS` view and have taken compensating actions at the logical standby database. `SKIP_TRANSACTION` is not the appropriate procedure to invoke to skip DML changes to a table.

To skip a DML failure, use a `SKIP` procedure, such as `SKIP('DML', 'MySchema', 'MyFailed Table')`. Using the `SKIP_TRANSACTION` procedure for DML transactions may skip changes for other tables, thus logically corrupting them.

---

- This procedure requires DBA privileges to execute.
- Use the `DBA_LOGSTDBY_SKIP_TRANSACTION` view to list the transactions that are going to be skipped by SQL Apply.
- Also, see the `ALTER DATABASE START LOGICAL STANDBY SKIP FAILED TRANSACTION` statement in *Oracle Database SQL Language Reference*.

## Exceptions

**Table 86–19** SKIP\_TRANSACTION Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

## Examples

To skip a DDL transaction with (XIDUSN, XIDSLT, XIDSQN) of (1.13.1726) you can register a rule as shown in the following example:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_TRANSACTION (-  
      XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```

## UNSKIP Procedure

Use the UNSKIP procedure to delete rules specified earlier with the SKIP procedure. The parameters specified in the UNSKIP procedure must match exactly for it to delete an already-specified rule.

### Syntax

```
DBMS_LOGSTDBY.UNSKIP (
    stmt                IN VARCHAR2,
    schema_name         IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2 DEFAULT NULL);
```

### Parameters

The parameter information for the UNSKIP procedure is the same as that described for the SKIP procedure. See [Table 86–13](#) on page 86-21 for complete parameter information.

### Exceptions

**Table 86–20 UNSKIP Procedure Exceptions**

Exception	Description
ORA-01031	need DBA privileges to execute this procedure
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

### Usage Notes

---

**CAUTION:** If DML changes for a table have been skipped and not compensated for, you must follow the call to the UNSKIP procedure with a call to the **INSTANTIATE\_TABLE** procedure to synchronize this table with those maintained by SQL Apply.

---

- This procedure requires DBA privileges to execute.
- Wildcards passed in the `schema_name` or the `object_name` parameter are not expanded. The wildcard character is matched at the character level. Thus, you can delete only one specified rule by invoking the UNSKIP procedure, and you will need a distinct UNSKIP procedure call to delete each rule that was previously specified.

For example, assume you have specified the following two rules to skip applying DML statements to the HR.EMPLOYEE and HR.EMPTEMP tables:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML', -
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMPLOYEE', -
    PROC_NAME => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML', -
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMPTEMP', -
    PROC_NAME => null);
```

In the following example, the wildcard in the `TABLE_NAME` parameter cannot be used to delete the rules that were specified:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP (STMT => 'DML', -  
    SCHEMA_NAME => 'HR', -  
    OBJECT_NAME => 'EMP%');
```

In fact, this `UNSKIP` procedure matches neither of the rules, because the wildcard character in the `TABLE_NAME` parameter is not expanded. Instead, the wildcard character will be used in an exact match to find the corresponding `SKIP` rule.

## UNSKIP\_ERROR Procedure

Use the UNSKIP\_ERROR procedure to delete rules specified earlier with the SKIP\_ERROR procedure. The parameters specified in the UNSKIP\_ERROR procedure must match exactly for the procedure to delete an already-specified rule.

### Syntax

```
DBMS_LOGSTDBY.UNSKIP_ERROR (
    stmt                IN VARCHAR2,
    schema_name         IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2 DEFAULT NULL);
```

### Parameters

The parameter information for the UNSKIP\_ERROR procedure is the same as that described for the SKIP\_ERROR procedure. See [Table 86–16](#) for complete parameter information.

### Exceptions

**Table 86–21 UNSKIP\_ERROR Procedure Exceptions**

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

### Usage Notes

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema\_name or the object\_name parameters are not expanded. Instead, the wildcard character is treated as any other character and an exact match is made. Thus, you can delete only one specified rule by invoking the UNSKIP\_ERROR procedure, and you need a distinct UNSKIP\_ERROR procedure call to delete each rule that you previously specified.

For example, assume you have specified the following two rules to handle the HR.EMPLOYEE and HR.EMPTEMP tables:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML', -
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMPLOYEE', -
    PROC_NAME => 'hr_employee_handler');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML', -
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMPTEMP', -
    PROC_NAME => 'hr_tempemp_handler');
```

In this case, the following UNSKIP procedure cannot be used to delete the rules that you have specified:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP_ERROR (STMT => 'DML', -
    SCHEMA_NAME => 'HR', -
    OBJECT_NAME => 'EMP%');
```

In fact, the UNSKIP procedure will match neither of the rules, because the wildcard character in the OBJECT\_NAME parameter will not be expanded.

### Example

To remove a handler that was previously registered with SQL Apply from getting called on encountering an error, you can issue the following statement:

```
DBMS_LOGSTDBY.UNSKIP_ERROR ( -  
    statement => 'NON_SCHEMA_DDL', -  
    schema_name => NULL, -  
    object_name => NULL);
```



## UNSKIP\_TRANSACTION Procedure

Use the UNSKIP\_TRANSACTION procedure to delete rules specified earlier with the SKIP\_TRANSACTION procedure. The parameters specified in the UNSKIP\_TRANSACTION procedure must match exactly for the procedure to delete an already-specified rule.

### Syntax

```
DBMS_LOGSTDBY.UNSKIP_TRANSACTION (
    xidusn_p      IN NUMBER,
    xidslt_p      IN NUMBER,
    xidsqn_p      IN NUMBER);
```

### Parameters

**Table 86–22 UNSKIP\_TRANSACTION Procedure Parameters**

Parameter	Description
XIDUSN	Transaction ID undo segment number of the transaction being skipped
XIDSLT	Transaction ID slot number of the transaction being skipped
XIDSQN	Transaction ID sequence number of the transaction being skipped

### Exceptions

**Table 86–23 UNSKIP\_TRANSACTION Procedure Exceptions**

Exception	Description
ORA-01031	need DBA privileges to execute this procedure
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

### Usage Notes

- This procedure requires DBA privileges to execute.
- Query the DBA\_LOGSTDBY\_SKIP\_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.

### Examples

To remove a rule that was originally specified to skip the application of a transaction with (XIDUSN, XIDSLT, XIDSQN) of (1.13.1726) issue the following statement:

```
SQL> DBMS_LOGSTDBY.UNSKIP_TRANSACTION (XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```



---

---

## DBMS\_METADATA

The DBMS\_METADATA package provides a way for you to retrieve metadata from the database dictionary as XML or creation DDL and to submit the XML to re-create the object.

**See Also:** *Oracle Database Utilities* for more information and for examples of using the Metadata API

This chapter contains the following topics:

- [Using DBMS\\_METADATA](#)
  - Overview
  - Security Model
  - Rules and Limits
- [Data Structures - Object and Table Types](#)
- [Subprogram Groupings](#)
  - Subprograms for Retrieving Multiple Objects From the Database
  - Subprograms for Submitting XML to the Database
- [Summary of All DBMS\\_METADATA Subprograms](#)

---

## Using DBMS\_METADATA

This section contains topics which relate to using the DBMS\_METADATA package.

- [Overview](#)
- [Security Model](#)
- [Rules and Limits](#)

## Overview

You can use the DBMS\_METADATA package to retrieve metadata and also to submit XML, as described in the following sections.

- [Retrieving Metadata](#)
- [Submitting XML](#)

### Retrieving Metadata

If you are retrieving metadata, you can specify:

- The kind of object to be retrieved. This can be either a particular object type (such as a table, index, or procedure) or a heterogeneous collection of object types that form a logical unit (such as a database export or schema export).
- Optional selection criteria, such as owner or name.
- Parse items (attributes of the returned objects to be parsed and returned separately).
- Optional transformations on the output, implemented by XSLT (Extensible Stylesheet Language Transformation) scripts. By default the output is represented in XML, but you can specify transformations (into SQL DDL, for example), which are implemented by XSLT stylesheets stored in the database or externally.

DBMS\_METADATA provides the following retrieval interfaces:

- For programmatic use: OPEN, SET\_FILTER, SET\_COUNT, GET\_QUERY, SET\_PARSE\_ITEM, ADD\_TRANSFORM, SET\_TRANSFORM\_PARAM, SET\_REMAP\_PARAM, FETCH\_XXX, and CLOSE retrieve multiple objects.
- For use in SQL queries and for browsing: GET\_XML, GET\_DDL and GET\_SXML return metadata for a single named object. The GET\_DEPENDENT\_XML, GET\_DEPENDENT\_DDL, GET\_GRANTED\_XML, and GET\_GRANTED\_DDL interfaces return metadata for one or more dependent or granted objects. These procedures do not support heterogeneous object types.

### Submitting XML

If you are submitting XML, you specify:

- The type of object
- Optional transform parameters to modify the object (for example, changing the object's owner)
- Parse items (attributes of the submitted objects to be parsed and submitted separately)
- Whether to execute the operation or simply return the generated DDL

DBMS\_METADATA provides a programmatic interface for submission of XML. It is comprised of the following procedures: OPENW, ADD\_TRANSFORM, SET\_TRANSFORM\_PARAM, SET\_REMAP\_PARAM, SET\_PARSE\_ITEM, CONVERT, PUT, and CLOSE.

## Security Model

The `DBMS_METADATA` package considers a privileged user to be one who is connected as user `SYS` or who has the `SELECT_CATALOG_ROLE` role. The object views of the Oracle metadata model implement security as follows:

- Nonprivileged users can see the metadata of only their own objects.
- Nonprivileged users can also retrieve public synonyms, system privileges granted to them, and object privileges granted to them or by them to others. This also includes privileges granted to `PUBLIC`.
- If callers request objects they are not privileged to retrieve, no exception is raised; the object is simply not retrieved.
- If nonprivileged users are granted some form of access to an object in someone else's schema, they will be able to retrieve the grant specification through the Metadata API, but not the object's actual metadata.
- In stored procedures, functions, and definers-rights packages, roles (such as `SELECT_CATALOG_ROLE`) are disabled. Therefore, such a PL/SQL program can only fetch metadata for objects in its own schema. If you want to write a PL/SQL program that fetches metadata for objects in a different schema (based on the invoker's possession of `SELECT_CATALOG_ROLE`), you must make the program invokers-rights.
- For all objects that have passwords (for example, db links, users, and roles), the following rules apply:
  - A user who has the `SELECT_CATALOG_ROLE` can see all metadata for an object except the passwords for that object.
  - The `SYS` user, users who have the `EXP_FULL_DATABASE` role, and users who own an object can see all metadata for that object, including passwords.

## Rules and Limits

In an Oracle Shared Server (OSS) environment, the `DBMS_METADATA` package must disable session migration and connection pooling. This results in any shared server process that is serving a session running the package to effectively become a default, dedicated server for the life of the session. You should ensure that sufficient shared servers are configured when the package is used and that the number of servers is not artificially limited by too small a value for the `MAX_SHARED_SERVERS` initialization parameter.

## Data Structures - Object and Table Types

The DBMS\_METADATA package defines, in the SYS schema, the following OBJECT and TABLE types.

```
CREATE TYPE sys.ku$_parsed_item AS OBJECT (  
    item          VARCHAR2(30),  
    value         VARCHAR2(4000),  
    object_row    NUMBER )  
/  
  
CREATE PUBLIC SYNONYM ku$_parsed_item FOR sys.ku$_parsed_item;  
  
CREATE TYPE sys.ku$_parsed_items IS TABLE OF sys.ku$_parsed_item  
/  
  
CREATE PUBLIC SYNONYM ku$_parsed_items FOR sys.ku$_parsed_items;  
  
CREATE TYPE sys.ku$_ddl AS OBJECT (  
    ddlText      CLOB,  
    parsedItem  sys.ku$_parsed_items )  
/  
  
CREATE PUBLIC SYNONYM ku$_ddl FOR sys.ku$_ddl;  
  
CREATE TYPE sys.ku$_ddls IS TABLE OF sys.ku$_ddl  
/  
  
CREATE PUBLIC SYNONYM ku$_ddls FOR sys.ku$_ddls;  
  
CREATE TYPE sys.ku$_multi_ddl AS OBJECT (  
    object_row   NUMBER,  
    ddls         sys.ku$_ddls )  
/  
  
CREATE OR REPLACE PUBLIC SYNONYM ku$_multi_ddl FOR sys.ku$_multi_ddl;  
  
CREATE TYPE sys.ku$_multi_ddls IS TABLE OF sys.ku$_multi_ddl;  
/  
  
CREATE OR REPLACE PUBLIC SYNONYM ku$_multi_ddls FOR  
    sys.ku$_multi_ddls;  
  
CREATE TYPE sys.ku$_ErrorLine IS OBJECT (  
    errorNumber  NUMBER,  
    errorText    VARCHAR2(2000) )  
/  
  
CREATE PUBLIC SYNONYM ku$_ErrorLine FOR sys.ku$_ErrorLine;  
  
CREATE TYPE sys.ku$_ErrorLines IS TABLE OF sys.ku$_ErrorLine  
/  
CREATE PUBLIC SYNONYM ku$ErrorLines FOR sys.ku$_ErrorLines;  
  
CREATE TYPE sys.ku$_SubmitResult AS OBJECT (  
    ddl          sys.ku$_ddl,  
    errorLines  sys.ku$_ErrorLines );  
/  

```



```
CREATE TYPE sys.ku$_SubmitResults IS TABLE OF sys.ku$_SubmitResult  
/  
  
CREATE PUBLIC SYNONYM ku$_SubmitResults FOR sys.ku$_SubmitResults;
```

---

## Subprogram Groupings

The `DBMS_METADATA` subprograms are used to retrieve objects from, and submit XML to, a database. Some subprograms are used for both activities, while others are used only for retrieval or only for submission.

- [Table 87-1](#) provides a summary, in alphabetical order, of `DBMS_METADATA` subprograms used to retrieve multiple objects from a database.
- [Table 87-2](#) provides a summary, in alphabetical order, of `DBMS_METADATA` subprograms used to submit XML metadata to a database.

## Subprograms for Retrieving Multiple Objects From the Database

Table 87–1 lists the subprograms used for retrieving multiple objects from the database.

**Table 87–1 DBMS\_METADATA Subprograms for Retrieving Multiple Objects**

Subprogram	Description
<a href="#">ADD_TRANSFORM Function</a> on page 87-12	Specifies a transform that <code>FETCH_XXX</code> applies to the XML representation of the retrieved objects
<a href="#">CLOSE Procedure</a> on page 87-16	Invalidates the handle returned by <code>OPEN</code> and cleans up the associated state
<a href="#">FETCH_XXX Functions and Procedures</a> on page 87-19	Returns metadata for objects meeting the criteria established by <code>OPEN</code> , <code>SET_FILTER</code> , <code>SET_COUNT</code> , <code>ADD_TRANSFORM</code> , and so on
<a href="#">GET_QUERY Function</a> on page 87-26	Returns the text of the queries that are used by <code>FETCH_XXX</code>
<a href="#">GET_XXX Functions</a> on page 87-22	Fetches the metadata for a specified object as XML, SXML, or DDL, using only a single call
<a href="#">OPEN Function</a> on page 87-27	Specifies the type of object to be retrieved, the version of its metadata, and the object model
<a href="#">SET_COUNT Procedure</a> on page 87-36	Specifies the maximum number of objects to be retrieved in a single <code>FETCH_XXX</code> call
<a href="#">SET_FILTER Procedure</a> on page 87-37	Specifies restrictions on the objects to be retrieved, for example, the object name or schema
<a href="#">SET_PARSE_ITEM Procedure</a> on page 87-47	Enables output parsing by specifying an object attribute to be parsed and returned
<a href="#">SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures</a> on page 87-50	Specifies parameters to the XSLT stylesheets identified by <code>transform_handle</code>

## Subprograms for Submitting XML to the Database

Table 87–2 lists the subprograms used for submitting XML to the database.

**Table 87–2 DBMS\_METADATA Subprograms for Submitting XML**

Subprogram	Description
<a href="#">ADD_TRANSFORM Function</a> on page 87-12	Specifies a transform for the XML documents
<a href="#">CLOSE Procedure</a> on page 87-16	Closes the context opened with <code>OPENW</code>
<a href="#">CONVERT Functions and Procedures</a> on page 87-17	Converts an XML document to DDL
<a href="#">OPENW Function</a> on page 87-33	Opens a write context
<a href="#">PUT Function</a> on page 87-34	Submits an XML document to the database
<a href="#">SET_PARSE_ITEM Procedure</a> on page 87-47	Specifies an object attribute to be parsed
<a href="#">SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures</a> on page 87-50	<code>SET_TRANSFORM_PARAM</code> specifies a parameter to a transform <code>SET_REMAP_PARAM</code> specifies a remapping for a transform

---

## Summary of All DBMS\_METADATA Subprograms

**Table 87-3 DBMS\_METADATA Package Subprograms**

Subprogram	Description
<a href="#">ADD_TRANSFORM Function</a> on page 87-12	Specifies a transform that <code>FETCH_xxx</code> applies to the XML representation of the retrieved objects
<a href="#">CLOSE Procedure</a> on page 87-16	Invalidates the handle returned by <code>OPEN</code> and cleans up the associated state
<a href="#">CONVERT Functions and Procedures</a> on page 87-17	Converts an XML document to DDL.
<a href="#">FETCH_xxx Functions and Procedures</a> on page 87-19	Returns metadata for objects meeting the criteria established by <code>OPEN</code> , <code>SET_FILTER</code> , <code>SET_COUNT</code> , <code>ADD_TRANSFORM</code> , and so on
<a href="#">GET_xxx Functions</a> on page 87-22	Fetches the metadata for a specified object as XML, SXML, or DDL, using only a single call
<a href="#">GET_QUERY Function</a> on page 87-26	Returns the text of the queries that are used by <code>FETCH_xxx</code>
<a href="#">OPEN Function</a> on page 87-27	Specifies the type of object to be retrieved, the version of its metadata, and the object model
<a href="#">OPENW Function</a> on page 87-33	Opens a write context
<a href="#">PUT Function</a> on page 87-34	Submits an XML document to the database
<a href="#">SET_COUNT Procedure</a> on page 87-36	Specifies the maximum number of objects to be retrieved in a single <code>FETCH_xxx</code> call
<a href="#">SET_FILTER Procedure</a> on page 87-37	Specifies restrictions on the objects to be retrieved, for example, the object name or schema
<a href="#">SET_PARSE_ITEM Procedure</a> on page 87-47	Enables output parsing by specifying an object attribute to be parsed and returned
<a href="#">SET_TRANSFORM_PARAM and SET_REMAP_PARAM Procedures</a> on page 87-50	Specifies parameters to the XSLT stylesheets identified by <code>transform_handle</code>

## ADD\_TRANSFORM Function

This function is used for both retrieval and submission:

- When this procedure is used to retrieve objects, it specifies a transform that `FETCH_XXX` applies to the XML representation of the retrieved objects.
- When used to submit objects, it specifies a transform that `CONVERT` or `PUT` applies to the XML representation of the submitted objects. It is possible to add more than one transform.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9
- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

```
DBMS_METADATA.ADD_TRANSFORM (
    handle      IN NUMBER,
    name        IN VARCHAR2,
    encoding    IN VARCHAR2 DEFAULT NULL,
    object_type IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

### Parameters

**Table 87–4** ADD\_TRANSFORM Function Parameters

Parameters	Description
handle	The handle returned from <code>OPEN</code> when this transform is used to retrieve objects. Or the handle returned from <code>OPENW</code> when this transform is used in the submission of XML metadata.

**Table 87-4 (Cont.) ADD\_TRANSFORM Function Parameters**

Parameters	Description
name	<p>The name of the transform. If name contains a period, colon, or forward slash, it is interpreted as the URL of a user-supplied XSLT script. See <i>Oracle XML DB Developer's Guide</i>.</p> <p>Otherwise, name designates a transform implemented by DBMS_METADATA. The following transforms are defined:</p> <ul style="list-style-type: none"> <li>■ ALTERDDL - The document is converted from ALTER_XML format to SQL DDL.</li> <li>■ ALTERXML - The document is converted from SXML difference format to ALTER_XML format. See the DBMS_METADATA_DIFF PL/SQL package for more information about SXML difference format.</li> <li>■ DDL - The document is transformed to DDL that creates the object. The output of this transform is <i>not</i> an XML document.</li> <li>■ MODIFY - The document is modified as directed by transform and remap parameters. The output of this transform is an XML document. If no transform or remap parameters are specified, the document is unchanged.</li> <li>■ MODIFYSXML - The SXML format document is modified as directed by transform and remap parameters. The output of this transform is an XML document. If no transform or remap parameters are specified, the document is unchanged.</li> <li>■ SXML - The document is converted to SXML format. This transform is only valid for a subset of object types. Valid type names are CLUSTER, CONTEXT, DB_LINK, FGA_POLICY, INDEX, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, QUEUE, QUEUE_TABLE, RLS_CONTEXT, RLS_GROUP, RLS_POLICY, ROLE, SEQUENCE, SYNONYM, TABLE, TABLESPACE, TRIGGER, TYPE, TYPE_SPEC, TYPE_BODY, USER, and VIEW.</li> <li>■ SXMLDDL - The SXML format document is transformed to DDL that creates the object.</li> </ul>
encoding	<p>The name of the Globalization Support character set in which the stylesheet pointed to by name is encoded. This is only valid if name is a URL. If left NULL and the URL is external to the database, UTF-8 encoding is assumed. If left NULL and the URL is internal to the database (that is, it begins with /oradb/), then the encoding is assumed to be the database character set.</p>

**Table 87–4 (Cont.) ADD\_TRANSFORM Function Parameters**

Parameters	Description
object_type	<p>The definition of this parameter depends upon whether you are retrieving objects or submitting XML metadata.</p> <ol style="list-style-type: none"> <li>When you use ADD_TRANSFORM to retrieve objects, the following definition of object_type applies: <p>Designates the object type to which the transform applies. (Note that this is an object type name, not a path name.) By default the transform applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, the following behavior can occur:</p> <ul style="list-style-type: none"> <li>if object_type is omitted, the transform applies to all object types within the heterogeneous collection</li> <li>if object_type is specified, the transform only applies to that specific object type within the collection</li> </ul> <p>If you omit this parameter you can add the DDL transform to all objects in a heterogeneous collection with a single call. If you supply this parameter, you can add a transform for a specific object type.</p> </li> <li>When you use ADD_TRANSFORM in the submission of XML metadata, this parameter is the object type to which the transform applies. By default, it is the object type of the OPENW handle. Because the OPENW handle cannot designate a heterogeneous object type, the caller would normally leave this parameter NULL in the ADD_TRANSFORM calls.</li> </ol>

## Return Values

The opaque handle that is returned is used as input to SET\_TRANSFORM\_PARAM and SET\_REMAP\_PARAM. Note that this handle is different from the handle returned by OPEN or OPENW; it refers to the transform, not the set of objects to be retrieved.

## Usage Notes

- With no transforms added, objects are returned by default as XML documents. You call ADD\_TRANSFORM to specify the XSLT stylesheets to be used to transform the returned XML documents.
- You can call ADD\_TRANSFORM more than once to apply multiple transforms to XML documents. Transforms are applied in the order in which they were specified, the output of the first transform being used as input to the second, and so on.
- The output of a DDL transform is *not* an XML document. Therefore, no transform should be added after the DDL transform.



- Each transform expects a certain format XML document as input. If the input document is unspecified, metadata XML format is assumed.

## Exceptions

- `INVALID_ARGVAL`. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INVALID_OPERATION`. `ADD_TRANSFORM` was called after the first call to `FETCH_XXX` for the `OPEN` context. After the first call to `FETCH_XXX` is made, no further calls to `ADD_TRANSFORM` for the current `OPEN` context are permitted.
- `INCONSISTENT_ARGS`. The arguments are inconsistent. Possible inconsistencies include the following:
  - `encoding` is specified even though `name` is not a URL
  - `object_type` is not part of the collection designated by `handle`

## CLOSE Procedure

This procedure is used for both retrieval and submission. This procedure invalidates the handle returned by `OPEN` (or `OPENW`) and cleans up the associated state.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9
- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

```
DBMS_METADATA.CLOSE (
    handle IN NUMBER);
```

### Parameters

**Table 87-5** *CLOSE Procedure Parameters*

Parameter	Description
handle	The handle returned from <code>OPEN</code> (or <code>OPENW</code> ).

### Usage Notes

---



---

**Note:** The following notes apply only to object retrieval

---



---

You can prematurely terminate the stream of objects established by `OPEN` or (`OPENW`).

- If a call to `FETCH_XXX` returns `NULL`, indicating no more objects, a call to `CLOSE` is made transparently. In this case, you can still call `CLOSE` on the handle and not get an exception. (The call to `CLOSE` is not required.)
- If you know that only one specific object will be returned, you should explicitly call `CLOSE` after the single `FETCH_XXX` call to free resources held by the handle.

### Exceptions

- `INVALID_ARGVAL`. The value for the `handle` parameter is `NULL` or invalid.

## CONVERT Functions and Procedures

The `CONVERT` functions and procedures transform input XML documents. The `CONVERT` functions return creation DDL. The `CONVERT` procedures return either XML or DDL, depending on the specified transforms.

**See Also:** For more information about related subprograms:

- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

The `CONVERT` functions are as follows:

```
DBMS_METADATA.CONVERT (
    handle IN NUMBER,
    document IN sys.XMLType)
RETURN sys.ku$_multi_ddls;
```

```
DBMS_METADATA.CONVERT (
    handle IN NUMBER,
    document IN CLOB)
RETURN sys.ku$_multi_ddls;
```

The `CONVERT` procedures are as follows:

```
DBMS_METADATA.CONVERT (
    handle IN NUMBER,
    document IN sys.XMLType,
    result IN OUT NOCOPY CLOB);
```

```
DBMS_METADATA.CONVERT (
    handle IN NUMBER,
    document IN CLOB,
    result IN OUT NOCOPY CLOB);
```

### Parameters

**Table 87–6** *CONVERT Subprogram Parameters*

Parameter	Description
handle	The handle returned from <code>OPENW</code> .
document	The XML document containing object metadata of the type of the <code>OPENW</code> handle.
result	The converted document.

### Return Values

Either XML or DDL, depending on the specified transforms.

### Usage Notes

You can think of `CONVERT` as the second half of `FETCH_xxx`, either `FETCH_DDL` (for the function variants) or `FETCH_CLOB` (for the procedure variants). There are two differences:

- `FETCH_XXX` gets its XML document from the database, but `CONVERT` gets its XML document from the caller
- `FETCH_DDL` returns its results in a `sys.ku$_ddls` nested table, but `CONVERT` returns a `sys.ku$_multi_ddls` nested table

The transforms specified with `ADD_TRANSFORM` are applied in turn, and the result is returned to the caller. For the function variants, the DDL transform must be specified. If parse items were specified, they are returned in the `parsedItems` column. Parse items are ignored by the procedure variants.

The encoding of the XML document is embedded in its CLOB or XMLType representation. The version of the metadata is embedded in the XML. The generated DDL is valid for the database version specified in `OPENW`.

## Exceptions

- `INVALID_ARGVAL`. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INCONSISTENT_OPERATION`. No transform was specified. The DDL transform was not specified (function variants only).
- `INCOMPATIBLE_DOCUMENT`. The version of the XML document is not compatible with this version of the software.

## FETCH\_xxx Functions and Procedures

These functions and procedures return metadata for objects meeting the criteria established by OPEN, SET\_FILTER, SET\_COUNT, ADD\_TRANSFORM, and so on. See "Usage Notes" on page 87-20 for the variants.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9

### Syntax

The FETCH functions are as follows:

```
DBMS_METADATA.FETCH_XML (
    handle IN NUMBER)
RETURN sys.XMLType;
```

**See Also:** *Oracle XML DB Developer's Guide* for a description of XMLType

```
DBMS_METADATA.FETCH_DDL (
    handle IN NUMBER)
RETURN sys.ku$_ddl;
```

```
DBMS_METADATA.FETCH_CLOB (
    handle          IN NUMBER,
    cache_lob       IN BOOLEAN DEFAULT TRUE,
    lob_duration    IN PLS_INTEGER DEFAULT DBMS_LOB.SESSION)
RETURN CLOB;
```

The FETCH procedures are as follows:

```
DBMS_METADATA.FETCH_CLOB (
    handle IN NUMBER,
    doc    IN OUT NOCOPY CLOB);
```

```
DBMS_METADATA.FETCH_XML_CLOB (
    handle IN NUMBER,
    doc    IN OUT NOCOPY CLOB,
    parsed_items OUT sys.ku$_parsed_items,
    object_type_path OUT VARCHAR2);
```

### Parameters

**Table 87-7** *FETCH\_xxx Function Parameters*

Parameters	Description
handle	The handle returned from OPEN.
cache_lob	TRUE=read LOB into buffer cache
lob_duration	The duration for the temporary LOB created by FETCH_CLOB, either DBMS_LOB.SESSION (the default) or DBMS_LOB.CALL.
doc	The metadata for the objects, or NULL if all objects have been returned.

**Table 87–7 (Cont.) FETCH\_XXX Function Parameters**

Parameters	Description
<code>parsed_items</code>	A nested table containing the items specified by <code>SET_PARSE_ITEM</code> . If <code>SET_PARSE_ITEM</code> was not called, a NULL is returned.
<code>object_type_path</code>	For heterogeneous object types, this is the full path name of the object type for the objects returned by the call to <code>FETCH_XXX</code> . If <code>handle</code> designates a homogeneous object type, a NULL is returned.

## Return Values

The metadata for the objects or NULL if all objects have been returned.

## Usage Notes

These functions and procedures return metadata for objects meeting the criteria established by the call to `OPEN` that returned the handle, and subsequent calls to `SET_FILTER`, `SET_COUNT`, `ADD_TRANSFORM`, and so on. Each call to `FETCH_XXX` returns the number of objects specified by `SET_COUNT` (or less, if fewer objects remain in the underlying cursor) until all objects have been returned. After the last object is returned, subsequent calls to `FETCH_XXX` return NULL and cause the stream created by `OPEN` to be transparently closed.

There are several different `FETCH_XXX` functions and procedures:

- The `FETCH_XML` function returns the XML metadata for an object as an `XMLType`. It assumes that if any transform has been specified, that transform will produce an XML document. In particular, it assumes that the DDL transform has not been specified.
- The `FETCH_DDL` function returns the DDL (to create the object) in a `sys.ku$_ddl`s nested table. It assumes that the DDL transform has been specified. Each row of the `sys.ku$_ddl`s nested table contains a single DDL statement in the `ddlText` column; if requested, parsed items for the DDL statement will be returned in the `parsedItems` column. Multiple DDL statements may be returned under the following circumstances:
  - When you call `SET_COUNT` to specify a count greater than 1
  - When an object is transformed into multiple DDL statements. For example, A `TYPE` object that has a DDL transform applied to it can be transformed into both `CREATE TYPE` and `CREATE TYPE BODY` statements. A `TABLE` object can be transformed into a `CREATE TABLE`, and one or more `ALTER TABLE` statements
- The `FETCH_CLOB` function simply returns the object, transformed or not, as a CLOB. By default, the CLOB is read into the buffer cache and has session duration, but these defaults can be overridden with the `cache_lob` and `lob_duration` parameters.
- The `FETCH_CLOB` procedure returns the objects by reference in an `IN OUT NOCOPY` parameter. This is faster than the function variant, which returns LOBs by value, a practice that involves an expensive LOB copy.
- The `FETCH_XML_CLOB` procedure returns the XML metadata for the objects as a CLOB in an `IN OUT NOCOPY` parameter. This helps to avoid LOB copies, which can consume a lot of resources. It also returns a nested table of parse items and the full path name of the object type of the returned objects.

- All LOBs returned by `FETCH_xxx` are temporary LOBs. You must free the LOB. If the LOB is supplied as an `IN OUT NOCOPY` parameter, you must also create the LOB.
- If `SET_PARSE_ITEM` was called, `FETCH_DDL` and `FETCH_XML_CLOB` return attributes of the object's metadata (or the DDL statement) in a `sys.ku$_parsed_items` nested table. For `FETCH_XML_CLOB`, the nested table is an `OUT` parameter. For `FETCH_DDL`, it is a column in the returned `sys.ku$_ddl`s nested table. Each row of the nested table corresponds to an item specified by `SET_PARSE_ITEM` and contains the following columns:
  - `item`—the name of the attribute as specified in the `name` parameter to `SET_PARSE_ITEM`.
  - `value`—the attribute value, or `NULL` if the attribute is not present in the DDL statement.
  - `object_row`—a positive integer indicating the object to which the parse item applies. If multiple objects are returned by `FETCH_xxx`, (because `SET_COUNT` specified a count greater than 1) then `object_row=1` for all items for the first object, 2 for the second, and so on.
- The rows of the `sys.ku$_parsed_items` nested table are ordered by ascending `object_row`, but otherwise the row order is undetermined. To find a particular parse item within an object row the caller must search the table for a match on `item`.
- In general there is no guarantee that a requested parse item will be returned. For example, the parse item may not apply to the object type or to the particular line of DDL, or the item's value may be `NULL`.
- If `SET_PARSE_ITEM` was not called, `NULL` is returned as the value of the parsed items nested table.
- It is expected that the same variant of `FETCH_xxx` will be called for all objects selected by `OPEN`. That is, programs will not intermix calls to `FETCH_XML`, `FETCH_DDL`, `FETCH_CLOB`, and so on using the same `OPEN` handle. The effect of calling different variants is undefined; it might do what you expect, but there are no guarantees.
- Every object fetched will be internally consistent with respect to on-going DDL (and the subsequent recursive DML) operations against the dictionary. In some cases, multiple queries may be issued, either because the object type is heterogeneous or for performance reasons (for example, one query for heap tables, one for index-organized tables). Consequently the `FETCH_xxx` calls may in fact be fetches from different underlying cursors (meaning that read consistency is not guaranteed).

## Exceptions

Most exceptions raised during execution of the query are propagated to the caller. Also, the following exceptions may be raised:

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INCONSISTENT_OPERATION`. Either `FETCH_XML` was called when the DDL transform had been specified, or `FETCH_DDL` was called when the DDL transform had *not* been specified.

## GET\_xxx Functions

The following GET\_xxx functions let you fetch metadata for objects with a single call:

- GET\_XML
- GET\_DDL
- GET\_SXML
- GET\_DEPENDENT\_XML
- GET\_DEPENDENT\_DDL
- GET\_GRANTED\_XML
- GET\_GRANTED\_DDL

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9

### Syntax

```
DBMS_METADATA.GET_XML (  
object_type      IN VARCHAR2,  
name             IN VARCHAR2,  
schema          IN VARCHAR2 DEFAULT NULL,  
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',  
model           IN VARCHAR2 DEFAULT 'ORACLE',  
transform       IN VARCHAR2 DEFAULT NULL)  
RETURN CLOB;
```

```
DBMS_METADATA.GET_DDL (  
object_type      IN VARCHAR2,  
name             IN VARCHAR2,  
schema          IN VARCHAR2 DEFAULT NULL,  
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',  
model           IN VARCHAR2 DEFAULT 'ORACLE',  
transform       IN VARCHAR2 DEFAULT 'DDL')  
RETURN CLOB;
```

```
DBMS_METADATA.GET_SXML (  
object_type      IN VARCHAR2,  
name             IN VARCHAR2 DEFAULT NULL,  
schema          IN VARCHAR2 DEFAULT NULL,  
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',  
model           IN VARCHAR2 DEFAULT 'ORACLE',  
transform       IN VARCHAR2 DEFAULT 'SXML')  
RETURN CLOB;
```

```
DBMS_METADATA.GET_DEPENDENT_XML (  
object_type      IN VARCHAR2,  
base_object_name IN VARCHAR2,  
base_object_schema IN VARCHAR2 DEFAULT NULL,  
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',  
model           IN VARCHAR2 DEFAULT 'ORACLE',  
transform       IN VARCHAR2 DEFAULT NULL,  
object_count    IN NUMBER   DEFAULT 10000)  
RETURN CLOB;
```



```

DBMS_METADATA.GET_DEPENDENT_DDL (
object_type      IN VARCHAR2,
base_object_name IN VARCHAR2,
base_object_schema IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model          IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT 'DDL',
object_count    IN NUMBER   DEFAULT 10000)
RETURN CLOB;

```

```

DBMS_METADATA.GET_GRANTED_XML (
object_type      IN VARCHAR2,
grantee         IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model          IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT NULL,
object_count    IN NUMBER   DEFAULT 10000)
RETURN CLOB;

```

```

DBMS_METADATA.GET_GRANTED_DDL (
object_type      IN VARCHAR2,
grantee         IN VARCHAR2 DEFAULT NULL,
version         IN VARCHAR2 DEFAULT 'COMPATIBLE',
model          IN VARCHAR2 DEFAULT 'ORACLE',
transform       IN VARCHAR2 DEFAULT 'DDL',
object_count    IN NUMBER   DEFAULT 10000)
RETURN CLOB;

```

## Parameters

**Table 87–8** *GET\_xxx Function Parameters*

Parameter	Description
object_type	The type of object to be retrieved. This parameter takes the same values as the OPEN object_type parameter, except that it cannot be a heterogeneous object type. The attributes of the object type must be appropriate to the function. That is, for GET_xxx it must be a named object.
name	The object name. It is used internally in a NAME filter. (If the name is longer than 30 characters, it will be used in a LONGNAME filter.) If this parameter is NULL, then no NAME or LONGNAME filter is specified. See Table 87–17 for a list of filters.
schema	The object schema. It is used internally in a SCHEMA filter. The default is the current user.
version	The version of metadata to be extracted. This parameter takes the same values as the OPEN version parameter.
model	The object model to use. This parameter takes the same values as the OPEN model parameter.
transform	The name of a transformation on the output. This parameter takes the same values as the ADD_TRANSFORM name parameter. For GET_XML this must not be DDL.
base_object_name	The base object name. It is used internally in a BASE_OBJECT_NAME filter.
base_object_schema	The base object schema. It is used internally in a BASE_OBJECT_SCHEMA filter. The default is the current user.

**Table 87–8 (Cont.) GET\_xxx Function Parameters**

Parameter	Description
grantee	The grantee. It is used internally in a GRANTEE filter. The default is the current user.
object_count	The maximum number of objects to return. See <a href="#">SET_COUNT Procedure</a> on page 87-36.

## Return Values

The metadata for the specified object as XML or DDL.

## Usage Notes

- These functions allow you to fetch metadata for objects with a single call. They encapsulate calls to OPEN, SET\_FILTER, and so on. The function you use depends on the characteristics of the object type and on whether you want XML, SXML, or DDL.
  - GET\_xxx is used to fetch named objects, especially schema objects (tables, views).
  - GET\_DEPENDENT\_xxx is used to fetch dependent objects (audits, object grants).
  - GET\_GRANTED\_xxx is used to fetch granted objects (system grants, role grants).
- For some object types you can use more than one function. For example, you can use GET\_xxx to fetch an index by name, or GET\_DEPENDENT\_xxx to fetch the same index by specifying the table on which it is defined.
- GET\_xxx only returns a single named object.
- For GET\_DEPENDENT\_xxx and GET\_GRANTED\_xxx, an arbitrary number of dependent or granted objects can match the input criteria. You can specify an object count when fetching these objects. (The default count of 10000 should be adequate in most cases.)
- If the DDL transform is specified, session-level transform parameters are inherited.
- If you invoke these functions from SQL\*Plus, you should set the PAGESIZE to 0 and set LONG to some large number to get complete, uninterrupted output.

## Exceptions

- INVALID\_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- OBJECT\_NOT\_FOUND. The specified object was not found in the database.

## Examples

### Example: Fetch the XML Representation of SCOTT.EMP

To generate complete, uninterrupted output, set the PAGESIZE to 0 and set LONG to some large number, as shown, before executing your query.

```
SET LONG 2000000
SET PAGESIZE 0
SELECT DBMS_METADATA.GET_XML('TABLE', 'EMP', 'SCOTT')
```

```
FROM DUAL;
```

**Example: Fetch the DDL for all Complete Tables in the Current Schema, Filter Out Nested Tables and Overflow Segments**

This example fetches the DDL for all "complete" tables in the current schema, filtering out nested tables and overflow segments. The example uses `SET_TRANSFORM_PARAM` (with the handle value = `DBMS_METADATA.SESSION_TRANSFORM` meaning "for the current session") to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the example resets the session-level parameters to their defaults.

To generate complete, uninterrupted output, set the `PAGESIZE` to 0 and set `LONG` to some large number, as shown, before executing your query.

```
SET LONG 2000000
SET PAGESIZE 0
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(DBMS_METADATA.SESSION_
TRANSFORM, 'STORAGE', false);
SELECT DBMS_METADATA.GET_DDL('TABLE', u.table_name)
       FROM USER_ALL_TABLES u
       WHERE u.nested='NO'
          AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(DBMS_METADATA.SESSION_
TRANSFORM, 'DEFAULT');
```

**Example: Fetch the DDL For All Object Grants On HR.EMPLOYEES**

```
SELECT DBMS_METADATA.GET_DEPENDENT_DDL('OBJECT_GRANT',
    'EMPLOYEES', 'HR') FROM DUAL;
```

**Example: Fetch the DDL For All System Grants Granted To SCOTT**

```
SELECT DBMS_METADATA.GET_GRANTED_DDL('SYSTEM_GRANT', 'SCOTT')
       FROM DUAL;
```

## GET\_QUERY Function

This function returns the text of the queries that are used by `FETCH_XXX`. This function assists in debugging.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9

### Syntax

```
DBMS_METADATA.GET_QUERY (  
    handle IN NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 87-9** *GET\_QUERY Function Parameters*

Parameter	Description
handle	The handle returned from <code>OPEN</code> . It cannot be the handle for a heterogeneous object type.

### Return Values

The text of the queries that will be used by `FETCH_XXX`.

### Exceptions

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for the `handle` parameter.

## OPEN Function

This function specifies the type of object to be retrieved, the version of its metadata, and the object model. The return value is an opaque context handle for the set of objects to be used in subsequent calls.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9

### Syntax

```
DBMS_METADATA.OPEN (
  object_type IN VARCHAR2,
  version     IN VARCHAR2 DEFAULT 'COMPATIBLE',
  model       IN VARCHAR2 DEFAULT 'ORACLE',
  network_link IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
```

### Parameters

**Table 87–10 Open Function Parameters**

Parameter	Description
object_type	<p>The type of object to be retrieved. <a href="#">Table 87–11</a> lists the valid type names and their meanings. These object types will be supported for the ORACLE model of metadata (see <a href="#">model</a> in this table).</p> <p>The Attributes column in <a href="#">Table 87–11</a> specifies some object type attributes:</p> <ul style="list-style-type: none"> <li>▪ Schema objects, such as tables, belong to schemas.</li> <li>▪ Named objects have unique names (if they are schema objects, the name is unique to the schema).</li> <li>▪ Dependent objects, such as indexes, are defined with reference to a base schema object.</li> <li>▪ Granted objects are granted or assigned to a user or role and therefore have a named grantee.</li> <li>▪ Heterogeneous object types denote a collection of related objects of different types. See <a href="#">Table 87–12</a> for a listing of object types returned for the heterogeneous object type.</li> </ul> <p>These attributes are relevant when choosing object selection criteria. See "<a href="#">SET_FILTER Procedure</a>" on page 87-37 for more information.</p>
version	<p>The version of metadata to be extracted. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows:</p> <p>COMPATIBLE (default)—the version of the metadata corresponds to the database compatibility level.</p> <p>LATEST—the version of the metadata corresponds to the database version.</p> <p>A specific database version. The value cannot be lower than 9.2.0.</p>

**Table 87–10 (Cont.) Open Function Parameters**

Parameter	Description
model	Specifies which view to use, because the API can support multiple views on the metadata. Only the ORACLE model is supported.
network_link	The name of a database link to the database whose metadata is to be retrieved. If NULL (the default), metadata is retrieved from the database on which the caller is running

Table 87–11 provides the name, meaning, attributes, and notes for the DBMS\_METADATA package object types. In the attributes column, S represents a schema object, N represents a named object, D represents a dependent object, G represents a granted object, and H represents a heterogeneous object.

**Table 87–11 DBMS\_METADATA: Object Types**

Type Name	Meaning	Attributes	Notes
AQ_QUEUE	queues	SND	Dependent on table
AQ_QUEUE_TABLE	additional metadata for queue tables	ND	Dependent on table
AQ_TRANSFORM	transforms	SN	None
ASSOCIATION	associate statistics	D	None
AUDIT	audits of SQL statements	DG	Modeled as dependent, granted object. The base object name is the statement audit option name (for example, ALTER SYSTEM). There is no base object schema. The grantee is the user or proxy whose statements are audited.
AUDIT_OBJ	audits of schema objects	D	None
CLUSTER	clusters	SN	None
COMMENT	comments	D	None
CONSTRAINT	constraints	SND	Does not include: <ul style="list-style-type: none"> <li>■ primary key constraint for IOT</li> <li>■ column NOT NULL constraints</li> <li>■ certain REF SCOPE and WITH ROWID constraints for tables with REF columns</li> </ul>
CONTEXT	application contexts	N	None
DATABASE_EXPORT	all metadata objects in a database	H	Corresponds to a full database export
DB_LINK	database links	SN	Modeled as schema objects because they have owners. For public links, the owner is PUBLIC. For private links, the creator is the owner.
DEFAULT_ROLE	default roles	G	Granted to a user by ALTER USER
DIMENSION	dimensions	SN	None
DIRECTORY	directories	N	None
FGA_POLICY	fine-grained audit policies	D	Not modeled as named object because policy names are not unique.

**Table 87-11 (Cont.) DBMS\_METADATA: Object Types**

Type Name	Meaning	Attributes	Notes
FUNCTION	stored functions	SN	None
INDEX_STATISTICS	precomputed statistics on indexes	D	The base object is the index's table.
INDEX	indexes	SND	None
INDEXTYPE	indextypes	SN	None
JAVA_SOURCE	Java sources	SN	None
JOB	jobs	S	None
LIBRARY	external procedure libraries	SN	None
MATERIALIZED_VIEW	materialized views	SN	None
MATERIALIZED_VIEW_LOG	materialized view logs	D	None
OBJECT_GRANT	object grants	DG	None
OPERATOR	operators	SN	None
PACKAGE	stored packages	SN	By default, both package specification and package body are retrieved. See " <a href="#">SET_FILTER Procedure</a> " on page 87-37.
PACKAGE_SPEC	package specifications	SN	None
PACKAGE_BODY	package bodies	SN	None
PROCEDURE	stored procedures	SN	None
PROFILE	profiles	N	None
PROXY	proxy authentications	G	Granted to a user by ALTER USER
REF_CONSTRAINT	referential constraint	SND	None
REFRESH_GROUP	refresh groups	SN	None
RESOURCE_COST	resource cost info		None
RLS_CONTEXT	driving contexts for enforcement of fine-grained access-control policies	D	Corresponds to the DBMS_RLS.ADD_POLICY_CONTENT procedure
RLS_GROUP	fine-grained access-control policy groups	D	Corresponds to the DBMS_RLS.CREATE_GROUP procedure
RLS_POLICY	fine-grained access-control policies	D	Corresponds to DBMS_RLS.ADD_GROUPED_POLICY. Not modeled as named objects because policy names are not unique.
RMGR_CONSUMER_GROUP	resource consumer groups	SN	Data Pump does not use these object types. Instead, it exports resource manager objects as procedural objects.
RMGR_INITIAL_CONSUMER_GROUP	assign initial consumer groups to users	G	None
RMGR_PLAN	resource plans	SN	None
RMGR_PLAN_DIRECTIVE	resource plan directives	D	Dependent on resource plan
ROLE	roles	N	None

**Table 87–11 (Cont.) DBMS\_METADATA: Object Types**

Type Name	Meaning	Attributes	Notes
ROLE_GRANT	role grants	G	None
ROLLBACK_SEGMENT	rollback segments	N	None
SCHEMA_EXPORT	all metadata objects in a schema	H	Corresponds to user-mode export.
SEQUENCE	sequences	SN	None
SYNONYM	synonyms	See notes	Private synonyms are schema objects. Public synonyms are not, but for the purposes of this API, their schema name is PUBLIC. The name of a synonym is considered to be the synonym itself. For example, in <code>CREATE PUBLIC SYNONYM FOO FOR BAR</code> , the resultant object is considered to have name FOO and schema PUBLIC.
SYSTEM_GRANT	system privilege grants	G	None
TABLE	tables	SN	None
TABLE_DATA	metadata describing row data for a table, nested table, or partition	SND	For partitions, the object name is the partition name.  For nested tables, the object name is the storage table name. The base object is the top-level table to which the table data belongs. For nested tables and partitioning, this is the top-level table ( <i>not</i> the parent table or partition). For nonpartitioned tables and non-nested tables this is the table itself.
TABLE_EXPORT	metadata for a table and its associated objects	H	Corresponds to table-mode export
TABLE_STATISTICS	precomputed statistics on tables	D	None
TABLESPACE	tablespaces	N	None
TABLESPACE_QUOTA	tablespace quotas	G	Granted with ALTER USER
TRANSPORTABLE_EXPORT	metadata for objects in a transportable tablespace set	H	Corresponds to transportable tablespace export
TRIGGER	triggers	SND	None
TRUSTED_DB_LINK	trusted links	N	None
TYPE	user-defined types	SN	By default, both type and type body are retrieved. See " <a href="#">SET_FILTER Procedure</a> " on page 87-37.
TYPE_SPEC	type specifications	SN	None
TYPE_BODY	type bodies	SN	None
USER	users	N	None
VIEW	views	SN	None
XMLSCHEMA	XML schema	SN	The object's name is its URL (which may be longer than 30 characters). Its schema is the user who registered it.



Table 87–12 lists the types of objects returned for the major heterogeneous object types. For SCHEMA\_EXPORT, certain object types are only returned if the INCLUDE\_USER filter is specified at TRUE. In the table, such object types are marked INCLUDE\_USER.

**Table 87–12 Object Types Returned for the Heterogeneous Object Type**

Object Type	DATABASE_EXPORT	SCHEMA_EXPORT	TABLE_EXPORT	TRANSPORTABLE_EXPORT
ASSOCIATION	Yes	No	No	No
AUDIT	Yes	No	No	No
AUDIT_OBJ	Yes	Yes	Yes	Yes
CLUSTER	Yes	Yes	No	Yes
COMMENT	Yes	Yes	Yes	Yes
CONSTRAINT	Yes	Yes	Yes	Yes
CONTEXT	Yes	No	No	No
DB_LINK	Yes	Yes	No	No
DEFAULT_ROLE	Yes	INCLUDE_USER	No	No
DIMENSION	Yes	Yes	No	No
DIRECTORY	Yes	No	No	No
FGA_POLICY	Yes	No	No	Yes
FUNCTION	Yes	Yes	No	No
INDEX_STATISTICS	Yes	Yes	Yes	Yes
INDEX	Yes	Yes	Yes	Yes
INDEXTYPE	Yes	Yes	No	No
JAVA_SOURCE	Yes	Yes	No	No
JOB	Yes	Yes	No	No
LIBRARY	Yes	Yes	No	No
MATERIALIZED_VIEW	Yes	Yes	No	No
MATERIALIZED_VIEW_LOG	Yes	Yes	No	No
OBJECT_GRANT	Yes	Yes	Yes	Yes
OPERATOR	Yes	Yes	No	No
PACKAGE	Yes	Yes	No	No
PACKAGE_SPEC	Yes	Yes	No	No
PACKAGE_BODY	Yes	Yes	No	No
PASSWORD_HISTORY	Yes	INCLUDE_USER	No	No
PASSWORD_VERIFY_FUNCTION	Yes	No	No	No
PROCEDURE	Yes	Yes	No	No
PROFILE	Yes	No	No	No
PROXY	Yes	No	No	No
REF_CONSTRAINT	Yes	Yes	Yes	Yes
REFRESH_GROUP	Yes	Yes	No	No
RESOURCE_COST	Yes	No	No	No

**Table 87–12 (Cont.) Object Types Returned for the Heterogeneous Object Type**

Object Type	DATABASE_EXPORT	SCHEMA_EXPORT	TABLE_EXPORT	TRANSPORTABLE_EXPORT
RLS_CONTEXT	Yes	No	No	Yes
RLS_GROUP	Yes	No	No	Yes
RLS_POLICY	Yes	Table data is retrieved according to policy	Table data is retrieved according to policy	Yes
ROLE	Yes	No	No	No
ROLE_GRANT	Yes	No	No	No
ROLLBACK_SEGMENT	Yes	No	No	No
SEQUENCE	Yes	Yes	No	No
SYNONYM	Yes	Yes	No	No
SYSTEM_GRANT	Yes	INCLUDE_USER	No	No
TABLE	Yes	Yes	Yes	Yes
TABLE_DATA	Yes	Yes	Yes	Yes
TABLE_STATISTICS	Yes	Yes	Yes	Yes
TABLESPACE	Yes	No	No	No
TABLESPACE_QUOTA	Yes	INCLUDE_USER	No	No
TRIGGER	Yes	Yes	Yes	Yes
TRUSTED_DB_LINK	Yes	No	No	No
TYPE	Yes	Yes	No	Yes, if the types are used by tables in the transportable set
TYPE_SPEC	Yes	Yes	No	Yes, if the types are used by tables in the transportable set
TYPE_BODY	Yes	Yes	No	Yes, if the types are used by tables in the transportable set
USER	Yes	INCLUDE_USER	No	No
VIEW	Yes	Yes	No	No
XMLSCHEMA	Yes	Yes	No	No

## Return Values

An opaque handle to the class of objects. This handle is used as input to `SET_FILTER`, `SET_COUNT`, `ADD_TRANSFORM`, `GET_QUERY`, `SET_PARSE_ITEM`, `FETCH_xxx`, and `CLOSE`.

## Exceptions

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INVALID_OBJECT_PARAM`. The `version` or `model` parameter was not valid for the `object_type`.

## OPENW Function

This function specifies the type of object to be submitted and the object model. The return value is an opaque context handle.

**See Also:** For more information about related subprograms:

- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

```
DBMS_METADATA.OPENW
(object_type IN VARCHAR2,
version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
model       IN VARCHAR2 DEFAULT 'ORACLE')
RETURN NUMBER;
```

### Parameters

**Table 87–13** OPENW Function Parameters

Parameter	Description
object_type	The type of object to be submitted. Valid types names and their meanings are listed in <a href="#">Table 87–11</a> . The type cannot be a heterogeneous object type.
version	The version of DDL to be generated by the CONVERT function. DDL clauses that are incompatible with the version will not be generated. The legal values for this parameter are as follows: <ul style="list-style-type: none"> <li>■ COMPATIBLE - This is the default. The version of the DDL corresponds to the database compatibility level. Database compatibility must be set to 9.2.0 or higher.</li> <li>■ LATEST - The version of the DDL corresponds to the database version.</li> <li>■ A specific database version. The value cannot be lower than 9.2.0.</li> </ul>
model	Specifies which view to use. Only the Oracle proprietary (ORACLE) view is supported by DBMS_METADATA.

### Return Values

An opaque handle to write context. This handle is used as input to the ADD\_TRANSFORM, CONVERT, PUT, and CLOSE procedures.

### Exceptions

- INVALID\_ARGVAL. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- INVALID\_OBJECT\_PARAM. The model parameter was not valid for the object\_type.

## PUT Function

This function submits an XML document containing object metadata to the database to create the object.

**See Also:** For more information about related subprograms:

- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

```
DBMS_METADATA.PUT (
    handle      IN          NUMBER,
    document    IN          sys.XMLType,
    flags       IN          NUMBER,
    results     IN OUT NOCOPY sys.ku$_SubmitResults)
RETURN BOOLEAN;
```

```
DBMS_METADATA.PUT (
    handle      IN          NUMBER,
    document    IN          CLOB,
    flags       IN          NUMBER,
    results     IN OUT NOCOPY sys.ku$_SubmitResults)
RETURN BOOLEAN;
```

### Parameters

**Table 87–14** PUT Function Parameters

Parameter	Description
handle	The handle returned from OPENW.
document	The XML document containing object metadata for the type of the OPENW handle.
flags	Reserved for future use
results	Detailed results of the operation.

### Return Values

TRUE if all SQL operations succeeded; FALSE if there were any errors.

### Usage Notes

The PUT function converts the XML document to DDL just as CONVERT does (applying the specified transforms in turn) and then submits each resultant DDL statement to the database. As with CONVERT, the DDL transform must be specified. The DDL statements and associated parse items are returned in the `sys.ku$_SubmitResults` nested table. With each DDL statement is a nested table of error lines containing any errors or exceptions raised by the statement.

The encoding of the XML document is embedded in its CLOB or XMLType representation. The version of the metadata is embedded in the XML. The generated DDL is valid for the database version specified in OPENW.

**Exceptions**

- `INVALID_ARGVAL`. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INCONSISTENT_OPERATION`. The DDL transform was not specified.
- `INCOMPATIBLE_DOCUMENT`. The version of the XML document is not compatible with this version of the software.

## SET\_COUNT Procedure

This procedure specifies the maximum number of objects to be retrieved in a single `FETCH_XXX` call. By default, each call to `FETCH_XXX` returns one object. You can use the `SET_COUNT` procedure to override this default. If `FETCH_XXX` is called from a client, specifying a count value greater than 1 can result in fewer server round trips and, therefore, improved performance.

For heterogeneous object types, a single `FETCH_XXX` operation only returns objects of a single object type.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9

### Syntax

```
DBMS_METADATA.SET_COUNT (
    handle          IN NUMBER,
    value           IN NUMBER,
    object_type_path IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 87–15** *SET\_COUNT Procedure Parameters*

Parameter	Description
handle	The handle returned from <code>OPEN</code> .
value	The maximum number of objects to retrieve.
object_type_path	A path name designating the object types to which the count value applies. By default, the count value applies to the object type of the <code>OPEN</code> handle. When the <code>OPEN</code> handle designates a heterogeneous object type, behavior can be either of the following: <ul style="list-style-type: none"> <li>▪ if <code>object_type_path</code> is omitted, the count applies to all object types within the heterogeneous collection</li> <li>▪ if <code>object_type_path</code> is specified, the count only applies to the specific node (or set of nodes) within the tree of object types forming the heterogeneous collection</li> </ul>

### Exceptions

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INVALID_OPERATION`. `SET_COUNT` was called after the first call to `FETCH_XXX` for the `OPEN` context. After the first call to `FETCH_XXX` is made, no further calls to `SET_COUNT` for the current `OPEN` context are permitted.
- `INCONSISTENT_ARGS`. `object_type` parameter is not consistent with `handle`.

## SET\_FILTER Procedure

This procedure specifies restrictions on the objects to be retrieved, for example, the object name or schema.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9

### Syntax

```
DBMS_METADATA.SET_FILTER (
  handle          IN NUMBER,
  name            IN VARCHAR2,
  value           IN VARCHAR2,
  object_type_path IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_METADATA.SET_FILTER (
  handle          IN NUMBER,
  name            IN VARCHAR2,
  value           IN BOOLEAN DEFAULT TRUE,
  object_type_path IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_METADATA.SET_FILTER (
  handle          IN NUMBER,
  name            IN VARCHAR2,
  value           IN NUMBER,
  object_type_path IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 87–16 SET\_FILTER Procedure Parameters**

Parameter	Description
handle	The handle returned from OPEN.
name	<p>The name of the filter. For each filter, <a href="#">Table 87–17</a> lists the <code>object_type</code> it applies to, its name, its datatype (text or Boolean) and its meaning or effect (including its default value, if any).</p> <p>The Datatype column of <a href="#">Table 87–17</a> also indicates whether a text filter is an expression filter. An expression filter is the right-hand side of a SQL comparison (that is, a SQL comparison operator (=, !=, and so on.) and the value compared against. The value must contain parentheses and quotation marks where appropriate. Note that in PL/SQL and SQL*Plus, two single quotes (<i>not</i> a double quote) are needed to represent an apostrophe. For example, an example of a NAME_EXPR filter in PL/SQL is as follows:</p> <pre>' IN ( 'DEPT' , 'EMP' )'</pre> <p>The filter value is combined with a particular object attribute to produce a WHERE condition in the query that fetches the objects. In the preceding example, the filter is combined with the attribute corresponding to an object name; objects named 'DEPT' and 'EMP' are selected.</p>
value	The value of the filter. Text, Boolean, and Numeric filters are supported.

**Table 87–16 (Cont.) SET\_FILTER Procedure Parameters**

Parameter	Description
object_type_path	A path name designating the object types to which the filter applies. By default, the filter applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, you can use this parameter to specify a filter for a specific node or set of nodes within the tree of object types that form the heterogeneous collection. See <a href="#">Table 87–18</a> for a listing of some of the values for this parameter.

[Table 87–17](#) describes the object type, name, datatype, and meaning of the filters available with the SET\_FILTER procedure.

**Table 87–17 SET\_FILTER: Filters**

Object Type	Name	Datatype	Meaning
Named objects	NAME	text	Objects with this exact name are selected.
Named objects	NAME_EXPR	text expression	The filter value is combined with the object attribute corresponding to the object name to produce a WHERE condition in the query that fetches the objects.  By default, all named objects of object_type are selected.
Named objects	EXCLUDE_NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the object name to specify objects that are to be excluded from the set of objects fetched.  By default, all named objects of the object type are selected.
Schema objects	SCHEMA	text	Objects in this schema are selected. If the object type is SYNONYM, specify PUBLIC to select public synonyms.
Schema objects	SCHEMA_EXPR	text expression	The filter value is combined with the attribute corresponding to the object's schema.  The default is determined as follows: - if BASE_OBJECT_SCHEMA is specified, then objects in that schema are selected; - otherwise, objects in the current schema are selected.
PACKAGE, TYPE	SPECIFICATION	Boolean	If TRUE, retrieve the package or type specification. Defaults to TRUE.
PACKAGE, TYPE	BODY	Boolean	If TRUE, retrieve the package or type body. Defaults to TRUE.
TABLE, CLUSTER, INDEX, TABLE_EXPORT, TRANSPORTABLE_EXPORT	TABLESPACE	text	Objects in this tablespace (or having a partition in this tablespace) are selected.



**Table 87-17 (Cont.) SET\_FILTER: Filters**

Object Type	Name	Datatype	Meaning
TABLE, CLUSTER, INDEX, TABLE_DATA, TABLE_EXPORT, TRANSPORTABLE_EXPORT	TABLESPACE_EXPR	text expression	The filter value is combined with the attribute corresponding to the object's tablespace (or in the case of a partitioned table or index, the partition's tablespaces). By default, objects in all tablespaces are selected.
TABLE, objects dependent on tables	PRIMARY	Boolean	If TRUE, retrieve primary tables (that is, tables for which the secondary object bit in obj\$ is clear. Defaults to TRUE.
TABLE, objects dependent on tables	SECONDARY	Boolean	If TRUE, retrieve secondary tables (that is, tables for which the secondary object bit in obj\$ is set). Defaults to TRUE.
Dependent Objects	BASE_OBJECT_NAME	text	Objects are selected that are defined or granted on objects with this name. Specify SCHEMA for triggers on schemas. Specify DATABASE for database triggers. Column-level comments cannot be selected by column name; the base object name must be the name of the table, view, or materialized view containing the column.
Dependent Objects	BASE_OBJECT_SCHEMA	text	Objects are selected that are defined or granted on objects in this schema. If BASE_OBJECT_NAME is specified with a value other than SCHEMA or DATABASE, this defaults to the current schema.
Dependent Objects	BASE_OBJECT_NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the name of the base object. Not valid for schema and database triggers.
Dependent Objects	EXCLUDE_BASE_OBJECT_NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the name of the base object to specify objects that are to be excluded from the set of objects fetched. Not valid for schema and database triggers.
Dependent Objects	BASE_OBJECT_SCHEMA_EXPR	text expression	The filter value is combined with the attribute corresponding to the schema of the base object.
Dependent Objects	BASE_OBJECT_TYPE	text	The object type of the base object.
Dependent Objects	BASE_OBJECT_TYPE_EXPR	text expression	The filter value is combined with the attribute corresponding to the object type of the base object. By default no filtering is done on object type.
Dependent Objects	BASE_OBJECT_TABLESPACE	text	The tablespace of the base object.
Dependent Objects	BASE_OBJECT_TABLESPACE_EXPR	text expression	The filter value is combined with the attribute corresponding to the tablespaces of the base object. By default, no filtering is done on the tablespace.
INDEX, TRIGGER	SYSTEM_GENERATED	Boolean	If TRUE, select indexes or triggers even if they are system-generated. If FALSE, omit system-generated indexes or triggers. Defaults to TRUE.
Granted Objects	GRANTEE	text	Objects are selected that are granted to this user or role. Specify PUBLIC for grants to PUBLIC.
Granted Objects	PRIVNAME	text	The name of the privilege or role to be granted. For TABLESPACE_QUOTA, only UNLIMITED can be specified.

**Table 87–17 (Cont.) SET\_FILTER: Filters**

Object Type	Name	Datatype	Meaning
Granted Objects	PRIVNAME_EXPR	text expression	The filter value is combined with the attribute corresponding to the privilege or role name. By default, all privileges/roles are returned.
Granted Objects	GRANTEE_EXPR	text expression	The filter value is combined with the attribute corresponding to the grantee name.
Granted Objects	EXCLUDE_ GRANTEE_EXPR	text expression	The filter value is combined with the attribute corresponding to the grantee name to specify objects that are to be excluded from the set of objects fetched.
OBJECT_GRANT	GRANTOR	text	Object grants are selected that are granted by this user.
SYNONYM, JAVA_ SOURCE, XMLSCHEMA	LONGNAME	text	A name longer than 30 characters. Objects with this exact name are selected. If the object name is 30 characters or less, the NAME filter must be used.
SYNONYM, JAVA_ SOURCE, XMLSCHEMA	LONGNAME_EXPR	text	The filter value is combined with the attribute corresponding to the object's long name. By default, no filtering is done on the long name of an object.
All objects	CUSTOM_FILTER	text	<p>The text of a WHERE condition. The condition is appended to the query that fetches the objects. By default, no custom filter is used.</p> <p>The other filters are intended to meet the needs of the majority of users. Use CUSTOM_FILTER when no defined filters exists for your purpose. Of necessity such a filter depends on the detailed structure of the UDTs and views used in the query. Because filters may change from version to version, upward compatibility is not guaranteed.</p>
All objects	EDITION	text	The edition filter is accepted for any object type, but affects only objects that support editions. The filter is only accepted for local objects (that is, the network_link parameter is not specified in the OPEN call). The edition name must be a valid edition name. If an edition is not specified, the edition of the active session is used.
SCHEMA_EXPORT	SCHEMA	text	The schema whose objects are selected.
SCHEMA_EXPORT	SCHEMA_EXPR	text expression	<p>The filter value is either:</p> <ul style="list-style-type: none"> <li>combined with the attribute corresponding to a schema name to produce a WHERE condition in the query that fetches schema objects,</li> <li>combined with the attribute corresponding to a base schema name to produce a WHERE condition in the query that fetches dependent objects.</li> </ul> <p>By default the current user's objects are selected.</p>
SCHEMA_EXPORT	INCLUDE_USER	Boolean	<p>If TRUE, retrieve objects containing privileged information about the user. For example, USER, PASSWORD_HISTORY, TABLESPACE_QUOTA.</p> <p>Defaults to FALSE.</p>
TABLE_EXPORT	SCHEMA	text	Objects (tables and their dependent objects) in this schema are selected.

**Table 87–17 (Cont.) SET\_FILTER: Filters**

Object Type	Name	Datatype	Meaning
TABLE_EXPORT	SCHEMA_EXPR	text expression	The filter value is either:  combined with the attribute corresponding to a schema name to produce a WHERE condition in the query that fetches the tables,  combined with the attribute corresponding to a base schema name to produce a WHERE condition in the query that fetches the tables' dependent objects.  By default the current user's objects are selected.
TABLE_EXPORT	NAME	text	The table with this exact name is selected along with its dependent objects.
TABLE_EXPORT	NAME_EXPR	text expression	The filter value is combined with the attribute corresponding to a table name in the queries that fetch tables and their dependent objects.  By default all tables in the selected schemas are selected, along with their dependent objects.
Heterogeneous objects	BEGIN_WITH	text	The fully qualified path name of the first object type in the heterogeneous collection to be retrieved. Objects normally fetched prior to this object type will not be retrieved.
Heterogeneous objects	BEGIN_AFTER	text	The fully qualified path name of an object type after which the heterogeneous retrieval should begin. Objects of this type will not be retrieved, nor will objects normally fetched prior to this object type.
Heterogeneous objects	END_BEFORE	text	The fully qualified path name of an object type where the heterogeneous retrieval should end. Objects of this type will not be retrieved, nor will objects normally fetched after this object type.
Heterogeneous objects	END_WITH	text	The fully qualified path name of the last object type in the heterogeneous collection to be retrieved. Objects normally fetched after this object type will not be retrieved.
Heterogeneous objects	INCLUDE_PATH_EXPR, EXCLUDE_PATH_EXPR	text expression	For these two filters, the filter value is combined with the attribute corresponding to an object type path name to produce a WHERE condition in the query that fetches the object types belonging to the heterogeneous collection. Objects of types satisfying this condition are included (INCLUDE_PATH_EXPR) or excluded (EXCLUDE_PATH_EXPR) from the set of object types fetched. Path names in the filter value do not have to be fully qualified. See <a href="#">Table 87–18</a> for valid path names that can be used with these filters.  BEGIN_WITH, BEGIN_AFTER, END_BEFORE, END_WITH, INCLUDE_PATH_EXPR, and EXCLUDE_PATH_EXPR all restrict the set of object types in the heterogeneous collection. By default, objects of all object types in the heterogeneous collection are retrieved.

## Usage Notes

- Each call to SET\_FILTER causes a WHERE condition to be added to the underlying query that fetches the set of objects. The WHERE conditions are ANDed together, so you can use multiple SET\_FILTER calls to refine the set of objects to be returned.

For example to specify that you want the object named EMP in schema SCOTT, do the following:

```
SET_FILTER(handle, 'SCHEMA', 'SCOTT');
SET_FILTER(handle, 'NAME', 'EMP');
```

- You can use the same text expression filter multiple times with different values. All the filter conditions will be applied to the query. For example, to get objects with names between Felix and Oscar, do the following:

```
SET_FILTER(handle, 'NAME_EXPR', '>=' 'FELIX' ');
SET_FILTER(handle, 'NAME_EXPR', '<=' 'OSCAR' ');
```

- With SET\_FILTER, you can specify the schema of objects to be retrieved, but security considerations may override this specification. If the caller is SYS or has SELECT\_CATALOG\_ROLE, then any object can be retrieved; otherwise, only the following can be retrieved:
  - Schema objects owned by the current user
  - Public synonyms
  - System privileges granted to the current user or to PUBLIC
  - Grants on objects for which the current user is owner, grantor, or grantee (either explicitly or as PUBLIC).
  - SCHEMA\_EXPORT where the name is the current user
  - TABLE\_EXPORT where SCHEMA is the current user

If you request objects that you are not privileged to retrieve, no exception is raised; the object is not retrieved, as if it did not exist.

In stored procedures, functions, and definers-rights packages, roles (such as SELECT\_CATALOG\_ROLE) are disabled. Therefore, such a PL/SQL program can only fetch metadata for objects in its own schema. If you want to write a PL/SQL program that fetches metadata for objects in a different schema (based on the invoker's possession of SELECT\_CATALOG\_ROLE), you must make the program invokers-rights.

- For heterogeneous object types, the BEGIN\_WITH and BEGIN\_AFTER filters allow restart on an object type boundary. Appropriate filter values are returned by the FETCH\_XML\_CLOB procedure.

Filters on heterogeneous objects provide default values for filters on object types within the collection. You can override this default for a particular object type by specifying the appropriate filter for the specific object type path. For example, for SCHEMA\_EXPORT the NAME filter specifies the schema to be fetched including all the tables in the schema, but you can further restrict this set of tables by supplying a NAME\_EXPR filter explicitly for the TABLE object type path. [Table 87-18](#) lists valid object type path names for the major heterogeneous object types along with an explanation of the scope of each path name. (The same information is available in the following catalog views: DATABASE\_EXPORT\_OBJECTS, SCHEMA\_EXPORT\_OBJECTS, and TABLE\_EXPORT\_OBJECTS.) See [Table 87-17](#) for filters defined for each path name. These path names are valid in the INCLUDE\_PATH\_EXPR and EXCLUDE\_PATH\_EXPR filters. Path names marked with an asterisk (\*) are *only* valid in those filters; they cannot be used as values of the SET\_FILTER object\_type\_path parameter.

**Table 87–18 Object Type Path Names for Heterogeneous Object Types**

<b>Heterogeneous Type</b>	<b>Path Name (*=valid only in xxx_PATH_EXPR)</b>	<b>Scope</b>
TABLE_EXPORT	AUDIT_OBJ	Object audits on the selected tables
TABLE_EXPORT	COMMENT	Table and column comments for the selected tables
TABLE_EXPORT	CONSTRAINT	Constraints (including referential constraints) on the selected tables
TABLE_EXPORT	*GRANT	Object grants on the selected tables
TABLE_EXPORT	INDEX	Indexes (including domain indexes) on the selected tables
TABLE_EXPORT	OBJECT_GRANT	Object grants on the selected tables
TABLE_EXPORT	REF_CONSTRAINT	Referential (foreign key) constraints on the selected tables
TABLE_EXPORT	STATISTICS	Statistics on the selected tables
TABLE_EXPORT	TABLE_DATA	Row data for the selected tables
TABLE_EXPORT	TRIGGER	Triggers on the selected tables
SCHEMA_EXPORT	ASSOCIATION	Statistics type associations for objects in the selected schemas
SCHEMA_EXPORT	AUDIT_OBJ	Audits on all objects in the selected schemas
SCHEMA_EXPORT	CLUSTER	Clusters in the selected schemas and their indexes
SCHEMA_EXPORT	COMMENT	Comments on all objects in the selected schemas
SCHEMA_EXPORT	CONSTRAINT	Constraints (including referential constraints) on all objects in the selected schemas
SCHEMA_EXPORT	DB_LINK	Private database links in the selected schemas
SCHEMA_EXPORT	DEFAULT_ROLE	Default roles granted to users associated with the selected schemas
SCHEMA_EXPORT	DIMENSION	Dimensions in the selected schemas
SCHEMA_EXPORT	FUNCTION	Functions in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	*GRANT	Grants on objects in the selected schemas
SCHEMA_EXPORT	INDEX	Indexes (including domain indexes) on tables and clusters in the selected schemas
SCHEMA_EXPORT	INDEXTYPE	Indextypes in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	JAVA_SOURCE	Java sources in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	JOB	Jobs in the selected schemas
SCHEMA_EXPORT	LIBRARY	External procedure libraries in the selected schemas
SCHEMA_EXPORT	MATERIALIZED_VIEW	Materialized views in the selected schemas
SCHEMA_EXPORT	MATERIALIZED_VIEW_LOG	Materialized view logs on tables in the selected schemas
SCHEMA_EXPORT	OBJECT_GRANT	Grants on objects in the selected schemas
SCHEMA_EXPORT	OPERATOR	Operators in the selected schemas and their dependent grants and audits

**Table 87–18 (Cont.) Object Type Path Names for Heterogeneous Object Types**

<b>Heterogeneous Type</b>	<b>Path Name (*=valid only in xxx_PATH_EXPR)</b>	<b>Scope</b>
SCHEMA_EXPORT	PACKAGE	Packages (both specification and body) in the selected schemas, and their dependent grants and audits
SCHEMA_EXPORT	PACKAGE_BODY	Package bodies in the selected schemas
SCHEMA_EXPORT	PACKAGE_SPEC	Package specifications in the selected schemas
SCHEMA_EXPORT	PASSWORD_HISTORY	The password history for users associated with the selected schemas
SCHEMA_EXPORT	PROCEDURE	Procedures in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	REF_CONSTRAINT	Referential (foreign key) constraints on tables in the selected schemas
SCHEMA_EXPORT	REFRESH_GROUP	Refresh groups in the selected schemas
SCHEMA_EXPORT	SEQUENCE	Sequences in the selected schemas and their dependent grants and audits
SCHEMA_EXPORT	STATISTICS	Statistics on tables and indexes in the selected schemas
SCHEMA_EXPORT	SYNONYM	Private synonyms in the selected schemas
SCHEMA_EXPORT	TABLE	Tables in the selected schemas and their dependent objects (indexes, constraints, triggers, grants, audits, comments, table data, and so on)
SCHEMA_EXPORT	TABLE_DATA	Row data for tables in the selected schemas
SCHEMA_EXPORT	TABLESPACE_QUOTA	Tablespace quota granted to users associated with the selected schemas
SCHEMA_EXPORT	TRIGGER	Triggers on tables in the selected schemas
SCHEMA_EXPORT	TYPE	Types (both specification and body) in the selected schemas, and their dependent grants and audits
SCHEMA_EXPORT	TYPE_BODY	Type bodies in the selected schemas
SCHEMA_EXPORT	TYPE_SPEC	Type specifications in the selected schemas
SCHEMA_EXPORT	USER	User definitions for users associated with the selected schemas
SCHEMA_EXPORT	VIEW	Views in the selected schemas and their dependent objects (grants, constraints, comments, audits)
DATABASE_EXPORT	ASSOCIATION	Statistics type associations for objects in the database
DATABASE_EXPORT	AUDIT	Audits of SQL statements
DATABASE_EXPORT	AUDIT_OBJ	Audits on all objects in the database
DATABASE_EXPORT	CLUSTER	Clusters and their indexes
DATABASE_EXPORT	COMMENT	Comments on all objects
DATABASE_EXPORT	CONSTRAINT	Constraints (including referential constraints)
DATABASE_EXPORT	CONTEXT	Application contexts
DATABASE_EXPORT	DB_LINK	Private and public database links
DATABASE_EXPORT	DEFAULT_ROLE	Default roles granted to users in the database
DATABASE_EXPORT	DIMENSION	Dimensions in the database

**Table 87–18 (Cont.) Object Type Path Names for Heterogeneous Object Types**

<b>Heterogeneous Type</b>	<b>Path Name (*=valid only in xxx_PATH_EXPR)</b>	<b>Scope</b>
DATABASE_EXPORT	DIRECTORY	Directory objects in the database
DATABASE_EXPORT	FGA_POLICY	Fine-grained audit policies
DATABASE_EXPORT	FUNCTION	Functions
DATABASE_EXPORT	* GRANT	Object and system grants
DATABASE_EXPORT	INDEX	Indexes (including domain indexes) on tables and clusters
DATABASE_EXPORT	INDEXTYPE	Indextypes and their dependent grants and audits
DATABASE_EXPORT	JAVA_SOURCE	Java sources and their dependent grants and audits
DATABASE_EXPORT	JOB	Jobs
DATABASE_EXPORT	LIBRARY	External procedure libraries
DATABASE_EXPORT	MATERIALIZED_VIEW	Materialized views
DATABASE_EXPORT	MATERIALIZED_VIEW_LOG	Materialized view logs
DATABASE_EXPORT	OBJECT_GRANT	All object grants in the database
DATABASE_EXPORT	OPERATOR	Operators and their dependent grants and audits
DATABASE_EXPORT	PACKAGE	Packages (both specification and body) and their dependent grants and audits
DATABASE_EXPORT	PACKAGE_BODY	Package bodies
DATABASE_EXPORT	PACKAGE_SPEC	Package specifications
DATABASE_EXPORT	PASSWORD_HISTORY	Password histories for database users
DATABASE_EXPORT	*PASSWORD_VERIFY_FUNCTION	The password complexity verification function
DATABASE_EXPORT	PROCEDURE	Procedures and their dependent grants and objects
DATABASE_EXPORT	PROFILE	Profiles
DATABASE_EXPORT	PROXY	Proxy authentications
DATABASE_EXPORT	REF_CONSTRAINT	Referential (foreign key) constraints on tables in the database
DATABASE_EXPORT	REFRESH_GROUP	Refresh groups
DATABASE_EXPORT	*RESOURCE_COST	Resource cost information
DATABASE_EXPORT	RLS_CONTEXT	Fine-grained access-control driving contexts
DATABASE_EXPORT	RLS_GROUP	Fine-grained access-control policy groups
DATABASE_EXPORT	RLS_POLICY	Fine-grained access-control policies
DATABASE_EXPORT	ROLE	Roles
DATABASE_EXPORT	ROLE_GRANT	Role grants to users in the database
DATABASE_EXPORT	ROLLBACK_SEGMENT	Rollback segments

**Table 87–18 (Cont.) Object Type Path Names for Heterogeneous Object Types**

Heterogeneous Type	Path Name (*=valid only in xxx_PATH_EXPR)	Scope
DATABASE_EXPORT	*SCHEMA (named object)	Database schemas including for each schema all related and dependent objects: user definitions and their attributes (default roles, role grants, tablespace quotas, and so on), objects in the schema (tables, view, packages, types, and so on), and their dependent objects (grants, audits, indexes, constraints, and so on). The NAME and NAME_EXPR filters can be used with this object type path name to designate the database schemas to be fetched.
DATABASE_EXPORT	SEQUENCE	Sequences
DATABASE_EXPORT	STATISTICS	Statistics on tables and indexes
DATABASE_EXPORT	SYNONYM	Public and private synonyms
DATABASE_EXPORT	SYSTEM_GRANT	System privilege grants
DATABASE_EXPORT	TABLE	Tables and their dependent objects (indexes, constraints, triggers, grants, audits, comments, table data, and so on)
DATABASE_EXPORT	TABLE_DATA	Row data for all tables
DATABASE_EXPORT	TABLESPACE	Tablespace definitions
DATABASE_EXPORT	TABLESPACE_QUOTA	Tablespace quota granted to users in the database
DATABASE_EXPORT	TRIGGER	Triggers on the database, on schemas, and on schema objects
DATABASE_EXPORT	TRUSTED_DB_LINK	Trusted links
DATABASE_EXPORT	TYPE	Types (both specification and body) and their dependent grants and audits
DATABASE_EXPORT	TYPE_BODY	Type bodies
DATABASE_EXPORT	TYPE_SPEC	Type specifications
DATABASE_EXPORT	USER	User definitions
DATABASE_EXPORT	VIEW	Views

## Exceptions

- **INVALID\_ARGVAL.** A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- **INVALID\_OPERATION.** SET\_FILTER was called after the first call to FETCH\_xxx for the OPEN context. After the first call to FETCH\_xxx is made, no further calls to SET\_FILTER are permitted.
- **INCONSISTENT\_ARGS.** The arguments are inconsistent. Possible inconsistencies include the following:
  - filter name not valid for the object type associated with the OPEN context
  - filter name not valid for the object\_type\_path
  - object\_type\_path not part of the collection designated by handle
  - filter value is the wrong datatype



## SET\_PARSE\_ITEM Procedure

This procedure is used for both retrieval and submission. This procedure enables output parsing and specifies an object attribute to be parsed and returned.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9
- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

The following syntax applies when SET\_PARSE\_ITEM is used for object retrieval:

```
DBMS_METADATA.SET_PARSE_ITEM (
  handle      IN NUMBER,
  name        IN VARCHAR2,
  object_type IN VARCHAR2 DEFAULT NULL);
```

The following syntax applies when SET\_PARSE\_ITEM is used for XML submission:

```
DBMS_METADATA.SET_PARSE_ITEM (
  handle      IN NUMBER,
  name        IN VARCHAR2);
```

### Parameters

**Table 87-19 SET\_PARSE\_ITEM Procedure Parameters**

Parameter	Description
handle	The handle returned from OPEN (or OPENW).
name	The name of the object attribute to be parsed and returned. See <a href="#">Table 87-20</a> for the attribute object type, name, and meaning.
object_type	Designates the object type to which the parse item applies (this is an object type name, not a path name). By default, the parse item applies to the object type of the OPEN handle. When the OPEN handle designates a heterogeneous object type, behavior can be either of the following: <ul style="list-style-type: none"> <li>▪ if object_type is omitted, the parse item applies to all object types within the heterogeneous collection</li> <li>▪ if object_type is specified, the parse item only applies to that specific object type within the collection</li> </ul> This parameter only applies when SET_PARSE_ITEM is used for object retrieval.

[Table 87-20](#) describes the object type, name, and meaning of the items available in the SET\_PARSE\_ITEM procedure.

**Table 87–20 SET\_PARSE\_ITEM: Parse Items**

Object Type	Name	Meaning
All objects	VERB	If <code>FETCH_XML_CLOB</code> is called, no value is returned.  If <code>FETCH_DDL</code> is called, then for every row in the <code>sys.ku\$_ddl</code> s nested table returned by <code>FETCH_DDL</code> the verb in the corresponding <code>ddlText</code> is returned. If the <code>ddlText</code> is a SQL DDL statement, then the SQL verb (for example, <code>CREATE</code> , <code>GRANT</code> , <code>AUDIT</code> ) is returned. If the <code>ddlText</code> is a procedure call (for example, <code>DBMS_AQADM.CREATE_QUEUE_TABLE()</code> ) then the <code>package.procedure-name</code> is returned.
All objects	OBJECT_TYPE	If <code>FETCH_XML_CLOB</code> is called, an object type name from <a href="#">Table 87–11</a> is returned.  If <code>FETCH_DDL</code> is called and the <code>ddlText</code> is a SQL DDL statement whose verb is <code>CREATE</code> or <code>ALTER</code> , the object type as used in the DDL statement is returned (for example, <code>TABLE</code> , <code>PACKAGE_BODY</code> , and so on). Otherwise, an object type name from <a href="#">Table 87–11</a> is returned.
Schema objects	SCHEMA	The object schema is returned. If the object is not a schema object, no value is returned.
Named objects	NAME	The object name is returned. If the object is not a named object, no value is returned.
TABLE, TABLE_DATA, INDEX	TABLESPACE	The name of the object's tablespace or, if the object is a partitioned table, the default tablespace is returned. For a <code>TABLE_DATA</code> object, this is always the tablespace where the rows are stored.
TRIGGER	ENABLE	If the trigger is enabled, <code>ENABLE</code> is returned. If the trigger is disabled, <code>DISABLE</code> is returned.
OBJECT_ GRANT, TABLESPACE_ QUOTA	GRANTOR	The grantor is returned.
Dependent objects (including domain index secondary tables)	BASE_OBJECT_NAME	The name of the base object is returned. If the object is not a dependent object, no value is returned.
Dependent objects (including domain index secondary tables)	BASE_OBJECT_SCHEMA	The schema of the base object is returned. If the object is not a dependent object, no value is returned.
Dependent objects (including domain index secondary tables)	BASE_OBJECT_TYPE	The object type of the base object is returned. If the object is not a dependent object, no value is returned.
Granted objects	GRANTEE	The grantee is returned. If the object is not a granted object, no value is returned.

## Usage Notes

These notes apply when using `SET_PARSE_ITEM` to retrieve objects.

By default, the `FETCH_xxx` routines return an object's metadata as XML or creation DDL. By calling `SET_PARSE_ITEM` you can request that individual attributes of the object be returned as well.

You can call `SET_PARSE_ITEM` multiple times to ask for multiple items to be parsed and returned. Parsed items are returned in the `sys.ku$_parsed_items` nested table.

For `TABLE_DATA` objects, the following parse item return values are of interest:

If Object Is	NAME, SCHEMA	BASE_OBJECT_NAME, BASE_OBJECT_SCHEMA
nonpartitioned table	table name, schema	table name, schema
table partition	partition name, schema	table name, schema
nested table	storage table name, schema	name and schema of top-level table ( <i>not</i> the parent nested table)

Tables are not usually thought of as dependent objects. However, secondary tables for domain indexes are dependent on the domain indexes. Consequently, the `BASE_OBJECT_NAME`, `BASE_OBJECT_SCHEMA` and `BASE_OBJECT_TYPE` parse items for secondary `TABLE` objects return the name, schema, and type of the domain index.

**See Also:**

- ["FETCH\\_xxx Functions and Procedures"](#) on page 87-19
- *Oracle Database Utilities* for information about using the Metadata API

By default, the `CONVERT` and `PUT` procedures simply transform an object's XML metadata to DDL. By calling `SET_PARSE_ITEM` you can request that individual attributes of the object be returned as well.

## Exceptions

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INVALID_OPERATION`. `SET_PARSE_ITEM` was called after the first call to `FETCH_xxx` for the `OPEN` context. After the first call to `FETCH_xxx` is made, no further calls to `SET_PARSE_ITEM` are permitted.
- `INCONSISTENT_ARGS`. The attribute name is not valid for the object type associated with the `OPEN` context.

## SET\_TRANSFORM\_PARAM and SET\_REMAP\_PARAM Procedures

These procedures are used for both retrieval and submission. SET\_TRANSFORM\_PARAM and SET\_REMAP\_PARAM specify parameters to the XSLT stylesheet identified by transform\_handle. Use them to modify or customize the output of the transform.

**See Also:** For more information about related subprograms:

- [Subprograms for Retrieving Multiple Objects From the Database](#) on page 87-9
- [Subprograms for Submitting XML to the Database](#) on page 87-10

### Syntax

```
DBMS_METADATA.SET_TRANSFORM_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    value               IN VARCHAR2,
    object_type         IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_METADATA.SET_TRANSFORM_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    value               IN BOOLEAN DEFAULT TRUE,
    object_type         IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_METADATA.SET_TRANSFORM_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    value               IN NUMBER,
    object_type         IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_METADATA.SET_REMAP_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    old_value           IN VARCHAR2,
    new_value           IN VARCHAR2,
    object_type         IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 87-21](#) describes the parameters for the SET\_TRANSFORM\_PARAM and SET\_REMAP\_PARAM procedures.

**Table 87-21** SET\_TRANSFORM\_PARAM and SET\_REMAP\_PARAM Parameters

Parameters	Description
transform_handle	<p>Either (1) the handle returned from ADD_TRANSFORM, or (2) the enumerated constant SESSION_TRANSFORM that designates the DDL transform for the whole session.</p> <p>Note that the handle returned by OPEN is not a valid transform handle.</p> <p>For SET_REMAP_PARAM, the transform handle must designate the MODIFY transform.</p>

**Table 87–21 (Cont.) SET\_TRANSFORM\_PARAM and SET\_REMAP\_PARAM Parameters**

Parameters	Description
name	The name of the parameter.  Table 87–22 lists the transform parameters defined for the DDL transform, specifying the <code>object_type</code> it applies to, its datatype, and its meaning or effect. This includes its default value, if any, and whether the parameter is additive.  Table 87–23 describes the parameters for the MODIFY transform in the <code>SET_TRANSFORM_PARAM</code> procedure.  Table 87–24 describes the parameters for the MODIFY transform in the <code>SET_REMAP_PARAM</code> procedure.
value	The value of the transform. This parameter is valid only for <code>SET_TRANSFORM_PARAM</code> .
old_value	The old value for the remapping. This parameter is valid only for <code>SET_REMAP_PARAM</code> .
new_value	The new value for the remapping. This parameter is valid only for <code>SET_REMAP_PARAM</code> .
object_type	Designates the object type to which the transform or remap parameter applies. By default, it applies to the same object type as the transform. In cases where the transform applies to all object types within a heterogeneous collection, the following apply: <ul style="list-style-type: none"> <li>▪ If <code>object_type</code> is omitted, the parameter applies to all applicable object types within the heterogeneous collection.</li> <li>▪ If <code>object_type</code> is specified, the parameter only applies to that object type.</li> </ul> <p>This allows a caller who has added a transform to a heterogeneous collection to specify different transform parameters for different object types within the collection.</p>

Table 87–22 describes the object type, name, datatype, and meaning of the parameters for the DDL transform in the `SET_TRANSFORM_PARAM` procedure.

**Table 87–22 SET\_TRANSFORM\_PARAM: Transform Parameters for the DDL Transform**

Object Type	Name	Datatype	Meaning
All objects	PRETTY	BOOLEAN	If TRUE, format the output with indentation and line feeds. Defaults to TRUE.
All objects	SQLTERMINATOR	BOOLEAN	If TRUE, append a SQL terminator (; or /) to each DDL statement. Defaults to FALSE.
TABLE	SEGMENT_ATTRIBUTES	BOOLEAN	If TRUE, include segment attributes clauses in the DDL. If FALSE, omit them. Defaults to TRUE.
TABLE	STORAGE	BOOLEAN	If TRUE, include storage clauses in the DDL. If FALSE, omit them. Defaults to TRUE. (Ignored if <code>SEGMENT_ATTRIBUTES</code> is FALSE.)
TABLE	TABLESPACE	BOOLEAN	If TRUE, include tablespace clauses in the DDL. If FALSE, omit them. (Ignored if <code>SEGMENT_ATTRIBUTES</code> is FALSE.) Defaults to TRUE.
TABLE	CONSTRAINTS	BOOLEAN	If TRUE, include all non-referential table constraints in the DDL. If FALSE, omit them. Defaults to TRUE.

**Table 87–22 (Cont.) SET\_TRANSFORM\_PARAM: Transform Parameters for the DDL Transform**

Object Type	Name	Datatype	Meaning
TABLE	REF_CONSTRAINTS	BOOLEAN	If TRUE, include all referential constraints (foreign keys) in the DDL. If FALSE, omit them. Defaults to TRUE.
TABLE	CONSTRAINTS_AS_ALTER	BOOLEAN	If TRUE, include table constraints as separate ALTER TABLE (and, if necessary, CREATE INDEX) statements. If FALSE, specify table constraints as part of the CREATE TABLE statement. Defaults to FALSE. Requires that CONSTRAINTS be TRUE.
TABLE	OID	BOOLEAN	If TRUE, include the OID clause for object tables in the DDL. If FALSE, omit it. Defaults to FALSE.
TABLE	SIZE_BYTE_KEYWORD	BOOLEAN	If TRUE, include the BYTE keyword as part of the size specification of CHAR and VARCHAR2 columns that use byte semantics. If FALSE, omit the keyword. Defaults to FALSE.
TABLE, INDEX	PARTITIONING	BOOLEAN	If TRUE, include partitioning clauses in the DDL. If FALSE, omit them. Defaults to TRUE.
INDEX, CONSTRAINT, ROLLBACK_ SEGMENT, CLUSTER, TABLESPACE	SEGMENT_ATTRIBUTES	BOOLEAN	If TRUE, include segment attributes clauses (physical attributes, storage attributes, tablespace, logging) in the DDL. If FALSE, omit them. Defaults to TRUE.
INDEX, CONSTRAINT, ROLLBACK_ SEGMENT, CLUSTER	STORAGE	BOOLEAN	If TRUE, include storage clauses in the DDL. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
INDEX, CONSTRAINT, ROLLBACK_ SEGMENT, CLUSTER	TABLESPACE	BOOLEAN	If TRUE, include tablespace clauses in the DDL. If FALSE, omit them. (Ignored if SEGMENT_ATTRIBUTES is FALSE.) Defaults to TRUE.
TYPE	SPECIFICATION	BOOLEAN	If TRUE, include the type specification in the DDL. If FALSE, omit it. Defaults to TRUE.
TYPE	BODY	BOOLEAN	If TRUE, include the type body in the DDL. If FALSE, omit it. Defaults to TRUE.
TYPE	OID	BOOLEAN	If TRUE, include the OID clause in the DDL. If FALSE, omit it. Defaults to FALSE.
PACKAGE	SPECIFICATION	BOOLEAN	If TRUE, include the package specification in the DDL. If FALSE, omit it. Defaults to TRUE.
PACKAGE	BODY	BOOLEAN	If TRUE, include the package body in the DDL. If FALSE, omit it. Defaults to TRUE.
VIEW	FORCE	BOOLEAN	If TRUE, use the FORCE keyword in the CREATE VIEW statement. If FALSE, do not use the FORCE keyword in the CREATE VIEW statement. Defaults to TRUE.
OUTLINE	INSERT	BOOLEAN	If TRUE, include the INSERT statements into the OL\$ dictionary tables that will create the outline and its hints. If FALSE, omit a CREATE OUTLINE statement. Defaults to FALSE.

Note: This object type is being deprecated.

**Table 87–22 (Cont.) SET\_TRANSFORM\_PARAM: Transform Parameters for the DDL Transform**

Object Type	Name	Datatype	Meaning
All objects	DEFAULT	BOOLEAN	Calling SET_TRANSFORM_PARAM with this parameter set to TRUE has the effect of resetting all parameters for the transform to their default values. Setting this FALSE has no effect. There is no default.
All objects	INHERIT	BOOLEAN	If TRUE, inherits session-level parameters. Defaults to FALSE. If an application calls ADD_TRANSFORM to add the DDL transform, then by default the only transform parameters that apply are those explicitly set for that transform handle. This has no effect if the transform handle is the session transform handle.
ROLE	REVOKE_FROM	Text	The name of a user from whom the role must be revoked. If this is a non-null string and if the CREATE ROLE statement grants you the role, a REVOKE statement is included in the DDL after the CREATE ROLE statement.  Note: When you issue a CREATE ROLE statement, Oracle may grant you the role. You can use this transform parameter to undo the grant.  Defaults to null string.
TABLESPACE	REUSE	BOOLEAN	If TRUE, include the REUSE parameter for datafiles in a tablespace to indicate that existing files can be reused. If FALSE, omit the REUSE parameter.  Defaults to FALSE.
CLUSTER, INDEX, ROLLBACK_ SEGMENT, TABLE, TABLESPACE	PCTSPACE	NUMBER	A number representing the percentage by which space allocation for the object type is to be modified. The value is the number of one-hundredths of the current allocation. For example, 100 means 100%.  If the object type is TABLESPACE, the following size values are affected: - in file specifications, the value of SIZE - MINIMUM EXTENT - EXTENT MANAGEMENT LOCAL UNIFORM SIZE  For other object types, INITIAL and NEXT are affected.

Table 87–23 describes the object type, name, datatype, and meaning of the parameters for the MODIFY transform in the SET\_TRANSFORM\_PARAM procedure.

**Table 87–23 SET\_TRANSFORM\_PARAM: Transform Parameters for the MODIFY Transform**

Object Type	Name	Datatype	Meaning
All objects	OBJECT_ROW	NUMBER	A number designating the object row for an object. The object in the document that corresponds to this number will be copied to the output document.  This parameter is additive.  By default, all objects are copied to the output document.

Table 87–24 describes the object type, name, datatype, and meaning of the parameters for the MODIFY transform in the SET\_REMAP\_PARAM procedure.

**Table 87–24 SET\_REMAP\_PARAM: Transform Parameters for the MODIFY Transform**

Object Type	Name	Datatype	Meaning
LIBRARY, TABLESPACE, DIRECTORY	REMAP_DATAFILE	Text	<p>Objects in the document will have their filespecs renamed as follows: any filespec matching <code>old_value</code> will be changed to <code>new_value</code>. Filespecs should <i>not</i> be enclosed in quotes.</p> <p>This parameter is additive.</p> <p>By default, filespecs are not renamed.</p>
Schema Objects, Dependent Objects, Granted Objects, USER	REMAP_SCHEMA	Text	<p>Any schema object in the document whose name matches <code>old_value</code> will have its schema name changed to <code>new_value</code>.</p> <p>Any dependent object whose base object schema name matches <code>old_value</code> will have its base object schema name changed to <code>new_value</code>.</p> <p>Any granted object whose grantee name matches <code>old_value</code> will have its grantee name changed to <code>new_value</code>.</p> <p>Any user whose name matches <code>old_value</code> will have its name changed to <code>new_value</code>.</p> <p>This parameter is additive.</p> <p>By default, schemas are not remapped.</p>
TABLE, CLUSTER, CONSTRAINT, INDEX, ROLLBACK_ SEGMENT, MATERIALIZED_ VIEW, MATERIALIZED_ VIEW_LOG, TABLESPACE_ QUOTA	REMAP_TABLESPACE	Text	<p>Objects in the document will have their tablespaces renamed as follows: any tablespace name matching <code>old_value</code> will be changed to <code>new_value</code>.</p> <p>This parameter is additive.</p> <p>By default, tablespaces are not remapped.</p>
Named objects and all objects dependent on named objects	REMAP_NAME	Text	<p>Any named object in the document whose name matches <code>old_value</code> will have its name changed to <code>new_value</code>.</p> <p>Any dependent object whose base object name matches <code>old_value</code> will have its base schema name changed to <code>new_value</code>.</p> <p>This parameter is additive.</p> <p>By default, names are not remapped.</p> <p>(Use <code>REMAP_TABLESPACE</code> to remap the name of a <code>TABLESPACE</code> object.)</p>

## Exceptions

- `INVALID_ARGVAL`. A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `INVALID_OPERATION`. Either `SET_TRANSFORM_PARAM` or `SET_REMAP_PARAM` was called after the first call to `FETCH_XXX` for the `OPEN` context. After the first



call to `FETCH_XXX` is made, no further calls to `SET_TRANSFORM_PARAM` or `SET_REMAP_PARAM` are permitted.

- `INCONSISTENT_ARGS`. The arguments are inconsistent. This can mean the following:
  - The transform parameter name is not valid for the object type associated with the `OPEN` context or for the transform associated with the transform handle.
  - The transform applies to all object types in a heterogeneous collection, but `object_type` is not part of the collection.

## Usage Notes

XSLT allows parameters to be passed to stylesheets. You call `SET_TRANSFORM_PARAM` or `SET_REMAP_PARAM` to specify the value of a parameter to be passed to the stylesheet identified by `transform_handle`.

Normally, if you call `SET_TRANSFORM_PARAMETER` multiple times for the same parameter name, each call overrides the prior call. For example, the following sequence simply sets the `STORAGE` transform parameter to `TRUE`.

```
SET_TRANSFORM_PARAM(tr_handle, 'STORAGE', false);
SET_TRANSFORM_PARAM(tr_handle, 'STORAGE', true);
```

However, some transform parameters are additive which means that all specified parameter values are applied to the document, not just the last one. For example, the `OBJECT_ROW` parameter to the `MODIFY` transform is additive. If you specify the following, then both specified rows are copied to the output document.

```
SET_TRANSFORM_PARAM(tr_handle, 'OBJECT_ROW', 5);
SET_TRANSFORM_PARAM(tr_handle, 'OBJECT_ROW', 8);
```

The `REMAP_TABLESPACE` parameter is also additive. If you specify the following, then tablespaces `TBS1` and `TBS3` are changed to `TBS2` and `TBS4`, respectively.

```
SET_REMAP_PARAM(tr_handle, 'REMAP_TABLESPACE', 'TBS1', 'TBS2');
SET_REMAP_PARAM(tr_handle, 'REMAP_TABLESPACE', 'TBS3', 'TBS4');
```

The order in which the transformations are performed is undefined. For example, if you specify the following, the result is undefined.

```
SET_REMAP_PARAM(tr_handle, 'REMAP_TABLESPACE', 'TBS1', 'TBS2');
SET_REMAP_PARAM(tr_handle, 'REMAP_TABLESPACE', 'TBS2', 'TBS3');
```

---

**Note:** The number of remap parameters that can be specified for a `MODIFY` transform is limited to ten. That is, you can specify up to ten `REMAP_DATAFILE` parameters, up to ten `REMAP_SCHEMA` parameters and so on. Additional instances are ignored. To work around this, you can perform another `DBMS_METADATA.ADD_TRANSFORM` and specify additional remap parameters.

---

The `GET_DDL`, `GET_DEPENDENT_DDL`, and `GET_GRANTED_DDL` functions allow the casual browser to extract the creation DDL for an object. So that you can specify transform parameters, this package defines an enumerated constant `SESSION_TRANSFORM` as the handle of the DDL transform at the session level. You can call `SET_TRANSFORM_PARAM` using `DBMS_METADATA.SESSION_TRANSFORM` as the transform handle to set transform parameters for the whole session. `GET_DDL`, `GET_`

DEPENDENT\_DDL, and GET GRANTED\_DDL inherit these parameters when they invoke the DDL transform.

---

---

**Note:** The enumerated constant must be prefixed with the package name `DBMS_METADATA.SESSION_TRANSFORM`.

---

---

---

---

## DBMS\_METADATA\_DIFF

The DBMS\_METADATA\_DIFF package contains the interfaces for comparing two metadata documents in SXML format.

**See Also:** *Oracle Database Utilities* for more information and for examples of using the Metadata API

This chapter contains the following topics:

- [Using DBMS\\_METADATA\\_DIFF](#)
  - [Overview](#)
  - [Security Model](#)
- [Browsing APIs for Fetching and Comparing Objects](#)
- [Summary of DBMS\\_METADATA\\_DIFF Subprograms](#)

---

## Using DBMS\_METADATA\_DIFF

This section contains topics which relate to using the DBMS\_METADATA\_DIFF package.

- [Overview](#)
- [Security Model](#)

## Overview

You can use the interfaces contained in the `DBMS_METADATA_DIFF` package to compare two metadata documents in SXML format. The result of the comparison is an SXML difference document. This document can be converted to other formats using the `DBMS_METADATA` submit interface and the `CONVERT` API.

## Security Model

The browsing interface of the `DBMS_METADATA_DIFF` package actually uses the `DBMS_METADATA` package to fetch the metadata to be compared. Therefore, the security model used for `DBMS_METADATA` also applies to `DBMS_METADATA_DIFF`. (Note, however, that `DBMS_METADATA_DIFF` does not support all object types.)

**See Also:** [DBMS\\_METADATA](#) for information about the `DBMS_METADATA` security model

## Browsing APIs for Fetching and Comparing Objects

These functions allow you to compare the metadata for two objects with a single call.

### Syntax

```
DBMS_METADATA_DIFF.COMPARE_SXML (
  object_type  IN VARCHAR2,
  name1        IN VARCHAR2,
  name2        IN VARCHAR2,
  schema1      IN VARCHAR2 DEFAULT NULL,
  schema2      IN VARCHAR2 DEFAULT NULL,
  network_link1 IN VARCHAR2 DEFAULT NULL,
  network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;
```

```
DBMS_METADATA_DIFF.COMPARE_ALTER (
  object_type  IN VARCHAR2,
  name1        IN VARCHAR2,
  name2        IN VARCHAR2,
  schema1      IN VARCHAR2 DEFAULT NULL,
  schema2      IN VARCHAR2 DEFAULT NULL,
  network_link1 IN VARCHAR2 DEFAULT NULL,
  network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;
```

```
DBMS_METADATA_DIFF.COMPARE_ALTER_XML (
  object_type  IN VARCHAR2,
  name1        IN VARCHAR2,
  name2        IN VARCHAR2,
  schema1      IN VARCHAR2 DEFAULT NULL,
  schema2      IN VARCHAR2 DEFAULT NULL,
  network_link1 IN VARCHAR2 DEFAULT NULL,
  network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;
```

### Parameters

**Table 88–1 COMPARE\_xxx Function Parameters**

Parameters	Description
object_type	The type of object to be compared. Valid type names are CLUSTER, CONTEXT, DB_LINK, FGA_POLICY, INDEX, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, QUEUE, QUEUE_TABLE, RLS_CONTEXT, RLS_GROUP, RLS_POLICY, ROLE, SEQUENCE, SYNONYM, TABLE, TABLESPACE, TRIGGER, TYPE, TYPE_SPEC, TYPE_BODY, USER, and VIEW.
name1	The name of the first object in the comparison.
name2	The name of the second object in the comparison.
schema1	The schema of the first object in the comparison. The default is the current user.
schema2	The schema of the second object in the comparison. The default is the value of schema1.

**Table 88–1 (Cont.) COMPARE\_xxx Function Parameters**

Parameters	Description
<code>network_link1</code>	The name of a database link to the database on which the first object resides. If NULL (the default), then the object is assumed to be in the database on which the caller is running.
<code>network_link2</code>	The name of a database link to the database on which the second object resides. The default is the value of <code>network_link1</code> .

## Return Values

`DBMS_METADATA_DIFF.COMPARE_xxx` returns the differences between two objects.

## Exceptions

- `INVALID_ARGVAL`  
A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- `OBJECT_NOT_FOUND`  
The specified object was not found in the database.

## Usage Notes

These functions encapsulate calls to both `DBMS_METADATA` and `DBMS_METADATA_DIFF` functions and procedures to fetch the metadata for each of the two objects and compare them.

Which function you use depends on the comparison format you want:

- `COMPARE_SXML` returns an SXML difference document.
- `COMPARE_ALTER` returns a set of ALTER statements for making the first object like the second object.
- `COMPARE_ALTER_XML` returns an ALTER\_XML document.



---

## Summary of DBMS\_METADATA\_DIFF Subprograms

The DBMS\_METADATA\_DIFF subprograms are used to:

- Specify the type of objects to be compared
- Specify the SXML documents to be compared
- Show the differences between the compared documents
- Clean up after the comparison

Table 88–2 provides a summary of DBMS\_METADATA\_DIFF subprograms.

**Table 88–2 DBMS\_METADATA\_DIFF Package Subprograms**

Subprogram	Description
<a href="#">OPENC Function</a> on page 88-8	Specifies the type of objects to be compared
<a href="#">ADD_DOCUMENT Procedure</a> on page 88-9	Specifies an SXML document to be compared
<a href="#">FETCH_CLOB Functions and Procedures</a> on page 88-10	Returns a CLOB showing the differences between the two documents specified by <code>ADD_DOCUMENT</code>
<a href="#">CLOSE Procedure</a> on page 88-11	Invalidates the handle returned by <code>OPENC</code> and cleans up associated state

## OPENC Function

This function specifies the type of objects to be compared. The return value is an opaque context handle.

### Syntax

```
DBMS_METADATA_DIFF.OPENC (
object_type IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 88–3** *OPENC Function Parameters*

Parameters	Description
object_type	The type of object to be compared. Valid type names are CLUSTER, CONTEXT, DB_LINK, FGA_POLICY, INDEX, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG, QUEUE, QUEUE_TABLE, RLS_CONTEXT, RLS_GROUP, RLS_POLICY, ROLE, SEQUENCE, SYNONYM, TABLE, TABLESPACE, TRIGGER, TYPE, TYPE_SPEC, TYPE_BODY, USER, and VIEW.

### Return Values

The opaque handle that is returned is used as input to ADD\_DOCUMENT, FETCH\_XXX and CLOSE.

### Exceptions

- INVALID\_ARGVAL  
A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.

## ADD\_DOCUMENT Procedure

This procedure specifies an SXML document to be compared.

### Syntax

```
DBMS_METADATA_DIFF.ADD_DOCUMENT(  
handle IN NUMBER, document IN sys.XMLType);
```

```
DBMS_METADATA_DIFF.ADD_DOCUMENT(  
handle IN NUMBER, document IN CLOB);
```

### Parameters

**Table 88–4** *CLOSE Procedure Parameters*

Parameter	Description
handle	The handle returned from OPENC
document	A document to be compared. The document must be of the type specified in OPENC.

### Usage Notes

Because the comparison interface allows you to compare exactly two SXML documents, a program must call ADD\_DOCUMENT exactly twice for each OPENC handle. In the comparison result, the document specified by the first call is document 1, and the document specified by the second call is document 2.

### Exceptions

- **INVALID\_ARGVAL**  
A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.

## FETCH\_CLOB Functions and Procedures

The `FETCH_CLOB` functions and procedures return a CLOB showing the differences between the two documents specified by `ADD_DOCUMENT`.

### Syntax

```
DBMS_METADATA_DIFF.FETCH_CLOB(
  handle IN NUMBER)
RETURN CLOB;
```

```
DBMS_METADATA_DIFF.FETCH_CLOB(
  handle IN NUMBER,
  doc    IN OUT NOCOPY CLOB);
```

```
DBMS_METADATA_DIFF.FETCH_CLOB(
  handle IN NUMBER,
  doc    IN OUT NOCOPY CLOB
  diffs  OUT BOOLEAN);
```

### Parameters

**Table 88–5** *CONVERT Subprogram Parameters*

Parameter	Description
<code>handle</code>	The handle returned from <code>OPENC</code> .
<code>doc</code>	A CLOB containing the differences between documents 1 and 2.
<code>diffs</code>	<code>TRUE</code> if the documents are different or <code>FALSE</code> if they are identical.

### Return Values

The differences between documents 1 and 2.

### Exceptions

- `INVALID_ARGVAL`  
A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.

## CLOSE Procedure

This procedure invalidates the handle returned by `OPENC` and cleans up associated state.

### Syntax

```
DBMS_METADATA_DIFF.CLOSE(  
handle IN NUMBER);
```

### Parameters

**Table 88–6** *FETCH\_xxx Function Parameters*

Parameters	Description
handle	The handle returned from <code>OPENC</code>

### Exceptions

- `INVALID_ARGVAL`  
A `NULL` or invalid value was supplied for an input parameter. The error message text identifies the parameter.



---

---

## DBMS\_MGD\_ID\_UTL

The DBMS\_MGD\_ID\_UTL package contains various functions and procedures that comprise the following utility subprograms:

- A logging utility that sets and gets Java and PL/SQL logging levels.
- A proxy utility consisting of two procedures used to set and unset the host and port of the proxy server.
- A metadata utility consisting of functions and procedures used for managing metadata.

**See Also:** *Oracle Database Advanced Application Developer's Guide* for more information.

This chapter describes each of these utility subprograms and contains the following topics:

- [Using DBMS\\_MGD\\_ID\\_UTL](#)
  - [Security Model](#)
  - [Constants](#)
  - [Exceptions](#)
- [Summary of DBMS\\_MGD\\_ID\\_UTL Subprograms](#)

The examples in this chapter assume that the user has run the following set of commands before running the contents of each script:

```
SQL> connect / as sysdba;
Connected.
SQL> create user mgduser identified by password;
SQL> grant connect, resource to mgduser;
SQL> connect mgduser
Enter password: mgduserpassword
Connected.
SQL> set serveroutput on;
```

## Using DBMS\_MGD\_ID\_UTL

- [Security Model](#)
- [Constants](#)
- [Exceptions](#)



## Security Model

You must run the `catmgd.sql` script to load the `DBMS_MGD_ID_UTL` package and Identity Code Package schema objects in the `MGDSYS` schema.

`DBMS_MGD_ID_UTL` is a `MGDSYS`-owned package. Any `DBMS_MGD_ID_UTL` subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

A user must be granted connect and resource roles to use the `DBMS_MGD_ID_UTL` package and its subprograms.

`EXECUTE` privilege is granted to `PUBLIC` for these ADTs: `MGD_ID`, `MGD_ID_COMPONENT`, `MGD_ID_COMPONENT_VARRAY`, and for this package `DBMS_MGD_ID_UTL`.

`SELECT` privilege is granted to `PUBLIC` for these read-only views: `MGD_ID_CATEGORY` and `MGD_ID_SCHEME` and for these metadata views: `USER_MGD_ID_CATEGORY` and `USER_MGD_ID_SCHEME`, and for table `MGD_ID_XML_VALIDATOR`, and for sequence `MGD$SEQUENCE_CATEGORY`.

`INSERT`, `UPDATE` and `DELETE` privilege is granted to `PUBLIC` for these metadata views: `USER_MGD_ID_CATEGORY` and `USER_MGD_ID_SCHEME`.

Public synonyms, by the same name, are created for these ADTs: `MGD_ID`, `MGD_ID_COMPONENT`, `MGD_ID_COMPONENT_VARRAY` and for this package `DBMS_MGD_ID_UTL`, as well as for these read-only views: `MGD_ID_CATEGORY` and `MGD_ID_SCHEME` and for these metadata views: `USER_MGD_ID_CATEGORY` and `USER_MGD_ID_SCHEME`, and for table `MGD_ID_XML_VALIDATOR`.

## Constants

DBMS\_MGD\_ID\_UTL uses the constants shown in [Table 89-1](#).

**Table 89-1 DBMS\_MGD\_ID\_UTL Constants**

<b>Name</b>	<b>Value</b>
<b>Installed Category IDs and Names</b>	
EPC_ENCODING_CATEGORY_ID	1
EPC_ENCODING_CATEGORY_NAME	EPC
<b>Logging Levels</b>	
LOGGING_LEVEL_OFF	0
LOGGING_LEVEL_SEVERE	1
LOGGING_LEVEL_WARNING	2
LOGGING_LEVEL_INFO	3
LOGGING_LEVEL_FINE	4
LOGGING_LEVEL_FINER	5
LOGGING_LEVEL_FINEST	6
LOGGING_LEVEL_ALL	7

## Exceptions

Table 89–2 lists the DBMS\_MGD\_ID\_UTL exceptions.

**Table 89–2 Exceptions Raised by DBMS\_MGD\_ID\_UTL Package**

<b>Name</b>	<b>Error Code</b>	<b>Description</b>
TDTJavaException	-55200	During the tag data translation, a Java exception was raised.
TDTCategoryNotFound	-55201	The specified category was not found.
TDTSchemeNotFound	-55202	During the tag data translation, the specified scheme was not found.
TDTLevelNotFound	-55203	During the tag data translation, the specified level was not found.
TDTOptionNotFound	-55204	During the tag data translation, the specified option was not found.
TDTFieldValidationException	-55205	During the tag data translation, the validation operation failed on a field.
TDTUndefinedField	-55206	During the tag data translation, an undefined field was detected.
TDTRuleEvaluationFailed	-55207	During the tag data translation, the rule evaluation operation failed.
TDTTooManyMatchingLevels	-55208	During the tag data translation, too many matching levels were found.

---

## Summary of DBMS\_MGD\_ID\_UTL Subprograms

Table 89–3 describes the utility subprograms in the DBMS\_MGD\_ID\_UTL package.

All the values and names passed to the procedures defined in the DBMS\_MGD\_ID\_UTL package are case insensitive unless otherwise mentioned. To preserve the case, enclose the values with double quotation marks.

**Table 89–3 DBMS\_MGD\_ID\_UTL Package Subprograms**

Subprogram	Description
ADD_SCHEME Procedure	Adds a tag data translation scheme to an existing category
CREATE_CATEGORY Function	Creates a new category or a new version of a category
EPC_TO_ORACLE_SCHEME Function	Converts the EPCglobal tag data translation (TDT) XML to Oracle tag data translation XML
GET_CATEGORY_ID Function	Returns the category ID given the category name and the category version
GET_COMPONENTS Function	Returns all relevant separated component names separated by semicolon (;) for the specified scheme
GET_ENCODINGS Function	Returns a list of semicolon (;) separated encodings (formats) for the specified scheme
GET_JAVA_LOGGING_LEVEL Function	Returns an integer representing the current Java trace logging level
GET_PLSQL_LOGGING_LEVEL Function	Returns an integer representing the current PL/SQL trace logging level
GET_SCHEME_NAMES Function	Returns a list of semicolon (;) separated scheme names for the specified category
GET_TDT_XML Function	Returns the Oracle tag data translation XML for the specified scheme
GET_VALIDATOR Function	Returns the Oracle Database tag data translation schema
REFRESH_CATEGORY Function	Refreshes the metadata information on the Java stack for the specified category
REMOVE_CATEGORY Procedure	Removes a category including all the related TDT XML if the value of <code>category_version</code> parameter is NULL
REMOVE_PROXY Procedure	Unsets the host and port of the proxy server
REMOVE_SCHEME Procedure	Removes a tag data translation scheme from a category
SET_JAVA_LOGGING_LEVEL Procedure	Sets the Java logging level
SET_PLSQL_LOGGING_LEVEL Procedure	Sets the PL/SQL tracing logging level
SET_PROXY Procedure	Sets the host and port of the proxy server for Internet access
VALIDATE_SCHEME Function	Validates the input tag data translation XML against the Oracle tag data translation schema

## ADD\_SCHEME Procedure

This procedure adds a tag data translation scheme to an existing category.

### Syntax

```
DBMS_MGD_ID_UTL.ADD_SCHEME (
  category_id IN VARCHAR2,
  tdt_xml     IN CLOB);
```

### Parameters

**Table 89–4 ADD\_SCHEME Procedure Parameters**

Parameter	Description
category_id	Category ID
tdt_xml	Tag data translation XML

### Examples

This example performs the following actions:

1. Creates a category.
2. Adds a contractor scheme and an employee scheme to the MGD\_SAMPLE\_CATEGORY category.
3. Validates the MGD\_SAMPLE\_CATEGORY scheme.
4. Tests the tag translation of the contractor scheme and the employee scheme.
5. Removes the contractor scheme.
6. Tests the tag translation of the contractor scheme and this returns the expected exception for the removed contractor scheme.
7. Tests the tag translation of the employee scheme and this returns the expected values.
8. Removes the MGD\_SAMPLE\_CATEGORY category.

```
--contents of add_scheme2.sql
SET LINESIZE 160
-----
---CREATE CATEGORY, ADD_SCHEME, REMOVE_SCHEME, REMOVE_CATEGORY-----
-----
DECLARE
  amt          NUMBER;
  buf          VARCHAR2(32767);
  pos          NUMBER;
  tdt_xml      CLOB;
  validate_tdtxml VARCHAR2(1042);
  category_id  VARCHAR2(256);
BEGIN
  -- remove the testing category if already existed
  DBMS_MGD_ID_UTL.remove_category('MGD_SAMPLE_CATEGORY', '1.0');
  -- Step 1. Create the testing category 'MGD_SAMPLE_CATEGORY', version 1.0.
  category_id := DBMS_MGD_ID_UTL.CREATE_CATEGORY('MGD_SAMPLE_CATEGORY', '1.0', 'Oracle',
'http://www.oracle.com/mgd/sample');
  -- Step 2. Add contractor scheme to the category.
  DBMS_LOB.CREATETEMPORARY(tdt_xml, true);
```

```

DBMS_LOB.OPEN(tdt_xml, DBMS_LOB.LOB_READWRITE);

buf := '<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema"
      xmlns="oracle.mgd.idcode">
<scheme name="CONTRACTOR_TAG" optionKey="1" xmlns="">
<level type="URI" prefixMatch="example.contractor.">
  <option optionKey="1" pattern="example.contractor.([0-9]*).([0-9]*)"
        grammar="'example.contractor.'" contractorID '.' divisionID">
    <field seq="1" characterSet="[0-9]*" name="contractorID"/>
    <field seq="2" characterSet="[0-9]*" name="divisionID"/>
  </option>
</level>
<level type="BINARY" prefixMatch="11">
  <option optionKey="1" pattern="11([01]{7})([01]{6})"
        grammar="'11'" contractorID divisionID ">
    <field seq="1" characterSet="[01]*" name="contractorID"/>
    <field seq="2" characterSet="[01]*" name="divisionID"/>
  </option>
</level>
</scheme>
</TagDataTranslation>';

amt := length(buf);
pos := 1;
DBMS_LOB.WRITE(tdt_xml, amt, pos, buf);
DBMS_LOB.CLOSE(tdt_xml);

DBMS_MGD_ID_UTL.ADD_SCHEME(category_id, tdt_xml);

-- Add the employee scheme to the category.
DBMS_LOB.CREATETEMPORARY(tdt_xml, true);
DBMS_LOB.OPEN(tdt_xml, DBMS_LOB.LOB_READWRITE);

buf := '<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema"
      xmlns="oracle.mgd.idcode">
<scheme name="EMPLOYEE_TAG" optionKey="1" xmlns="">
<level type="URI" prefixMatch="example.employee.">
  <option optionKey="1" pattern="example.employee.([0-9]*).([0-9]*)"
        grammar="'example.employee.'" employeeID '.' divisionID">
    <field seq="1" characterSet="[0-9]*" name="employeeID"/>
    <field seq="2" characterSet="[0-9]*" name="divisionID"/>
  </option>
</level>
<level type="BINARY" prefixMatch="01">
  <option optionKey="1" pattern="01([01]{7})([01]{6})"
        grammar="'01'" employeeID divisionID ">
    <field seq="1" characterSet="[01]*" name="employeeID"/>
    <field seq="2" characterSet="[01]*" name="divisionID"/>
  </option>
</level>
</scheme>
</TagDataTranslation>';

amt := length(buf);
pos := 1;
DBMS_LOB.WRITE(tdt_xml, amt, pos, buf);

```

```

DBMS_LOB.CLOSE(tdt_xml);
DBMS_MGD_ID_UTL.ADD_SCHEME(category_id, tdt_xml);

-- Step 3. Validate the scheme.
dbms_output.put_line('Validate the MGD_SAMPLE_CATEGORY Scheme');
validate_tdtxml := DBMS_MGD_ID_UTL.validate_scheme(tdt_xml);
dbms_output.put_line(validate_tdtxml);
dbms_output.put_line('Length of scheme xml is: '||DBMS_LOB.GETLENGTH(tdt_xml));

-- Step 4. Test tag translation of contractor scheme.
dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    'example.contractor.123.45',
    NULL, 'BINARY'));

dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    '111111011101101',
    NULL, 'URI'));

-- Test tag translation of employee scheme.
dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    'example.employee.123.45',
    NULL, 'BINARY'));

dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    '011111011101101',
    NULL, 'URI'));

DBMS_MGD_ID_UTL.REMOVE_SCHEME(category_id, 'CONTRACTOR_TAG');

-- Step 6. Test tag translation of contractor scheme. Doesn't work any more.
BEGIN
  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
      'example.contractor.123.45',
      NULL, 'BINARY'));

  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
      '111111011101101',
      NULL, 'URI'));
EXCEPTION
  WHEN others THEN
    dbms_output.put_line('Contractor tag translation failed: '||SQLERRM);
END;

-- Step 7. Test tag translation of employee scheme. Still works.
BEGIN
  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
      'example.employee.123.45',
      NULL, 'BINARY'));

  dbms_output.put_line(
    mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
      '011111011101101',
      NULL, 'URI'));
EXCEPTION

```

```
    WHEN others THEN
        dbms_output.put_line('Employee tag translation failed: '||SQLERRM);
END;

-- Step 8. Remove the testing category, which also removes all the associated schemes
DBMS_MGD_ID_UTL.remove_category('MGD_SAMPLE_CATEGORY', '1.0');
END;
/
SHOW ERRORS;

SQL> @add_scheme3.sql
.
.
.
Validate the MGD_SAMPLE_CATEGORY Scheme
EMPLOYEE_TAG;URI,BINARY;divisionID,employeeID
Length of scheme xml is: 933
111111011101101
example.contractor.123.45
011111011101101
example.employee.123.45
Contractor tag translation failed: ORA-55203: Tag data translation level not found
ORA-06512: at "MGDSYS.DBMS_MGD_ID_UTL", line 54
ORA-06512: at "MGDSYS.MGD_ID", line 242
ORA-29532: Java call terminated by uncaught Java
exception: oracle.mgd.idcode.exceptions.TDTLevelNotFound: Matching level not
found for any configured scheme
011111011101101
example.employee.123.45
.
.
.
```



## CREATE\_CATEGORY Function

This function creates a new category or a new version of a category.

### Syntax

```
DBMS_MGD_ID_UTL.CREATE_CATEGORY (
  category_name  IN  VARCHAR2,
  category_version IN VARCHAR2,
  agency         IN  VARCHAR2,
  URI           IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 89–5 CREATE\_CATEGORY Function Parameters**

Parameter	Description
category_name	Name of category
category_version	Category version
agency	Organization that owns the category. For example, EPCglobal owns the category EPC.
URI	URI that provides additional information about the category

### Usage Notes

The return value is the category ID.

### Examples

See the [ADD\\_SCHEME Procedure](#) for an example of creating the MGD\_SAMPLE\_CATEGORY category.

## EPC\_TO\_ORACLE\_SCHEME Function

This function converts the EPCglobal tag data translation (TDT) XML to Oracle Database tag data translation XML.

### Syntax

```
DBMS_MGD_ID_UTL.EPC_TO_ORACLE_SCHEME (
    xml_scheme IN CLOB)
RETURN CLOB;
```

### Parameters

**Table 89–6 EPC\_TO\_ORACLE\_SCHEME Function Parameters**

Parameter	Description
xml_scheme	Name of EPC tag scheme to be converted

### Usage Notes

The return value is the contents of the CLOB containing the Oracle Database tag data translation XML.

### Examples

The following example converts standard EPCglobal Tag Data Translation (TDT) files into Oracle Database TDT files:

```
--Contents of MGD_ID_DOC2.sql
-----
-- EPC_TO_ORACLE_SCHEME --
-----
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');

BEGIN
  DBMS_JAVA.set_output(1000000);
  DBMS_OUTPUT.ENABLE(1000000);
  DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_SEVERE);
END;
/

DECLARE
  epcScheme          CLOB;
  oracleScheme       CLOB;
  amt                 NUMBER;
  buf                 VARCHAR2(32767);
  pos                 NUMBER;
  seq                 BINARY_INTEGER;
  validate_epcscheme VARCHAR2(256);
  validate_oraclescheme VARCHAR2(256);
BEGIN

  DBMS_LOB.CREATETEMPORARY(epcScheme, true);
  DBMS_LOB.OPEN(epcScheme, DBMS_LOB.LOB_READWRITE);

  buf := '<?xml version="1.0" encoding="UTF-8"?>
<epcTagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
                    epcTDSVersion="1.1r1.27"
```

```

        xmlns:xsi="http://www.w3.org/2001/XMLSchema"
        xsi:noNamespaceSchemaLocation="EpcTagDataTranslation.xsd">
<scheme name="GID-96" optionKey="1" tagLength="96">
  <level type="BINARY" prefixMatch="00110101"
    requiredFormattingParameters="taglength">
    <option optionKey="1" pattern="00110101([01]{28})([01]{24})([01]{36})"
      grammar="'00110101' generalmanager objectclass serial">
      <field seq="1" decimalMinimum="0" decimalMaximum="268435455"
        characterSet="[01]*" bitLength="28" name="generalmanager"/>
      <field seq="2" decimalMinimum="0" decimalMaximum="16777215"
        characterSet="[01]*" bitLength="24" name="objectclass"/>
      <field seq="3" decimalMinimum="0" decimalMaximum="68719476735"
        characterSet="[01]*" bitLength="36" name="serial"/>
    </option>
  </level>
  <level type="TAG_ENCODING" prefixMatch="urn:epc:tag:gid-96"
    requiredFormattingParameters="taglength">
    <option optionKey="1"
      pattern="urn:epc:tag:gid-96:([0-9]*)\.[(0-9]*)\.[(0-9)*]"
      grammar="'urn:epc:tag:gid-96:' generalmanager '.' objectclass '.' serial">
      <field seq="1" decimalMinimum="0" decimalMaximum="268435455"
        characterSet="[0-9]*" name="generalmanager"/>
      <field seq="2" decimalMinimum="0" decimalMaximum="16777215"
        characterSet="[0-9]*" name="objectclass"/>
      <field seq="3" decimalMinimum="0" decimalMaximum="68719476735"
        characterSet="[0-9]*" name="serial"/>
    </option>
  </level>
  <level type="PURE_IDENTITY" prefixMatch="urn:epc:id:gid">
    <option optionKey="1"
      pattern="urn:epc:id:gid:([0-9]*)\.[(0-9]*)\.[(0-9)*]"
      grammar="'urn:epc:id:gid:' generalmanager '.' objectclass '.' serial">
      <field seq="1" decimalMinimum="0" decimalMaximum="268435455"
        characterSet="[0-9]*" name="generalmanager"/>
      <field seq="2" decimalMinimum="0" decimalMaximum="16777215"
        characterSet="[0-9]*" name="objectclass"/>
      <field seq="3" decimalMinimum="0" decimalMaximum="68719476735"
        characterSet="[0-9]*" name="serial"/>
    </option>
  </level>
  <level type="LEGACY" prefixMatch="generalmanager=">
    <option optionKey="1"
      pattern="generalmanager=([0-9]*);objectclass=([0-9]*);serial=([0-9]*)"
      grammar="'generalmanager='generalmanager';objectclass='objectclass ';serial='
serial">
      <field seq="1" decimalMinimum="0" decimalMaximum="268435455"
        characterSet="[0-9]*" name="generalmanager"/>
      <field seq="2" decimalMinimum="0" decimalMaximum="16777215"
        characterSet="[0-9]*" name="objectclass"/>
      <field seq="3" decimalMinimum="0" decimalMaximum="68719476735"
        characterSet="[0-9]*" name="serial"/>
    </option>
  </level>
</scheme>
</epcTagDataTranslation>';
amt := length(buf);
pos := 1;
DBMS_LOB.WRITE(epcScheme, amt, pos, buf);
DBMS_LOB.CLOSE(epcScheme);
oracleScheme := DBMS_MGD_ID_UTL.epc_to_oracle_scheme(epcScheme);

```

```

dbms_output.put_line('Length of oracle scheme xml is: '||DBMS_LOB.GETLENGTH(oracleScheme));
dbms_output.put_line(DBMS_LOB.SUBSTR(oracleScheme, DBMS_LOB.GETLENGTH(oracleScheme), 1));
dbms_output.put_line(' ');
dbms_output.put_line('Validate the Oracle Scheme');
validate_oraclescheme := DBMS_MGD_ID_UTL.validate_scheme(oracleScheme);
dbms_output.put_line('Validation result: '||validate_oraclescheme);
END;
/
SHOW ERRORS;

```

```
SQL> @mgd_id_doc2.sql
```

```
PL/SQL procedure successfully completed.
```

```
Length of oracle scheme xml is: 2475
```

```

<?xml version = '1.0' encoding = 'UTF-8'?>
<TagDataTranslation version="0.04"
date="2005-04-18T16:05:00Z" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="oracle.mgd.idcode"><scheme name="GID-96" optionKey="1" xmlns=""><level
type="BINARY" prefixMatch="00110101" requiredFormattingParameters=""><option
optionKey="1" pattern="00110101([01]{28})([01]{24})([01]{36})"
grammar="'00110101' generalmanager objectclass serial"><field seq="1"
decimalMinimum="0" decimalMaximum="268435455" characterSet="[01]*"
bitLength="28" name="generalmanager"/><field seq="2" decimalMinimum="0"
decimalMaximum="16777215" characterSet="[01]*" bitLength="24"
name="objectclass"/><field seq="3" decimalMinimum="0"
decimalMaximum="68719476735" characterSet="[01]*" bitLength="36"
name="serial"/></option></level><level type="TAG_ENCODING"
prefixMatch="urn:epc:tag:gid-96" requiredFormattingParameters=""><option
optionKey="1" pattern="urn:epc:tag:gid-96:([0-9]*)\.([0-9]*)\.([0-9]*"
grammar="'urn:epc:tag:gid-96:' generalmanager '.' objectclass '.' serial"><field
seq="1" decimalMinimum="0" decimalMaximum="268435455" characterSet="[0-9]*"
name="generalmanager"/><field seq="2" decimalMinimum="0"
decimalMaximum="16777215" characterSet="[0-9]*" name="objectclass"/><field
seq="3" decimalMinimum="0" decimalMaximum="68719476735" characterSet="[0-9]*"
name="serial"/></option></level><level type="PURE_IDENTITY"
prefixMatch="urn:epc:id:gid"><option optionKey="1"
pattern="urn:epc:id:gid:([0-9]*)\.([0-9]*)\.([0-9]*" grammar="'urn:epc:id:gid:'
generalmanager '.' objectclass '.' serial"><field seq="1" decimalMinimum="0"
decimalMaximum="268435455" characterSet="[0-9]*" name="generalmanager"/><field
seq="2" decimalMinimum="0" decimalMaximum="16777215" characterSet="[0-9]*"
name="objectclass"/><field seq="3" decimalMinimum="0"
decimalMaximum="68719476735" characterSet="[0-9]*"
name="serial"/></option></level><level type="LEGACY"
prefixMatch="generalmanager="><option optionKey="1"
pattern="generalmanager=([0-9]*);objectclass=([0-9]*);serial=([0-9]*"
grammar="'generalmanager='generalmanager';objectclass='objectclass';serial='
serial"><field seq="1" decimalMinimum="0" decimalMaximum="268435455"
characterSet="[0-9]*" name="generalmanager"/><field seq="2" decimalMinimum="0"
decimalMaximum="16777215" characterSet="[0-9]*" name="objectclass"/><field
seq="3" decimalMinimum="0" decimalMaximum="68719476735" characterSet="[0-9]*"
name="serial"/></option></level></scheme></TagDataTranslation>
Validate the Oracle Scheme
Validation result:
GID-96;LEGACY,TAG_ENCODING,PURE_IDENTITY,BINARY;objectclass,generalmanager,serial,

```

```
PL/SQL procedure successfully completed.
```

```

.
.
.

```

## GET\_CATEGORY\_ID Function

This function returns the category ID for a given category name and category version.

### Syntax

```
DBMS_MGD_ID_UTL.GET_CATEGORY_ID (
    category name      IN  VARCHAR2,
    category_version   IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 89–7 GET\_CATEGORY\_ID Function Parameters**

Parameter	Description
category_name	Name of category
category_version	Category version

### Usage Notes

- If the value of `category_version` is `NULL`, then the ID of the latest version of the specified category is returned.
- The return value is the category ID for the specified category name.

### Examples

The following example returns a category ID given a category name and its version:

```
-- Contents of get_category1.sql file
SELECT DBMS_MGD_ID_UTL.get_category_id('EPC', NULL) FROM DUAL;
```

```
SQL> @get_category1.sql
```

```
.
.
.
```

```
DBMS_MGD_ID_UTL.GET_CATEGORY_ID('EPC',NULL)
```

```
-----
```

```
1
.
.
.
```

## GET\_COMPONENTS Function

This function returns all relevant separated component names separated by semicolon (;) for the specified scheme.

### Syntax

```
DBMS_MGD_ID_UTL.GET_COMPONENTS (
    category_id IN VARCHAR2,
    scheme_name IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 89–8** GET\_COMPONENTS Function Parameters

Parameter	Description
category_id	Category ID
scheme_name	Name of scheme

### Usage Notes

The return value contains the component names separated by a semicolon (;) for the specified scheme.

### Examples

The following example gets the components:

```
--Contents of get_components.sql
DECLARE
    id            mgd_id;
    getcomps     VARCHAR2(1000);
    getencodings VARCHAR2(1000);
    getschemenames VARCHAR2(1000);
BEGIN
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
    DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
    getcomps := DBMS_MGD_ID_UTL.get_components(1, 'SGTIN-64');
    dbms_output.put_line('Component names are: ' || getcomps);
    getencodings := DBMS_MGD_ID_UTL.get_encodings(1, 'SGTIN-64');
    dbms_output.put_line('Encodings are: ' || getencodings);
    getschemenames := DBMS_MGD_ID_UTL.get_scheme_names(1);
    dbms_output.put_line('Scheme names are: ' || getschemenames);
END;
/
SHOW ERRORS;

SQL> @get_components.sql
.
.
.
Component names are:
filter,gtin,companyprefixlength,companyprefix,companyprefixindex,itemref,serial
Encodings are: ONS_HOSTNAME,LEGACY,TAG_ENCODING,PURE_IDENTITY,BINARY
Scheme names are:
GIAI-64,GIAI-96,GID-96,GRAI-64,GRAI-96,SGLN-64,SGLN-96,SGTIN-64,SGTIN-96,SSCC-64
,SSCC-96,USDOD-64,USDOD-96
```

PL/SQL procedure successfully completed.

.  
. .  
.

## GET\_ENCODINGS Function

This function returns a list of semicolon (;) separated encodings (formats) for the specified scheme.

### Syntax

```
DBMS_MGD_ID_UTL.GET_ENCODINGS (  
    category_id IN VARCHAR2,  
    scheme_name IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 89–9** GET\_ENCODINGS Function Parameters

Parameter	Description
category_id	Category ID
scheme_name	Name of scheme

### Usage Notes

The return value contains the encodings separated by a semicolon (;) for the specified scheme.

### Examples

See the [GET\\_COMPONENTS Function](#) for an example.



## GET\_JAVA\_LOGGING\_LEVEL Function

This function returns an integer representing the current trace logging level.

### Syntax

```
DBMS_MGD_ID_UTL.GET_JAVA_LOGGING_LEVEL  
RETURN INTEGER;
```

### Parameters

None.

### Usage Notes

The return value is the integer value denoting the current Java logging level.

### Examples

The following example gets the Java logging level.

```
--Contents of getjavalogginglevel.sql  
DECLARE  
    loglevel    NUMBER;  
BEGIN  
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);  
    loglevel := DBMS_MGD_ID_UTL.get_java_logging_level();  
    dbms_output.put_line('Java logging level = ' || loglevel);  
END;  
/  
SHOW ERRORS;  
  
SQL> @getjavalogginglevel.sql  
.  
.  
.  
Java logging level = 0  
PL/SQL procedure successfully completed.  
.  
.  
.
```

## GET\_PLSQL\_LOGGING\_LEVEL Function

This function returns an integer representing the current PL/SQL trace logging level.

### Syntax

```
DBMS_MGD_ID_UTL.GET_PLSQL_LOGGING_LEVEL
RETURN INTEGER;

PRAGMA restrict_references(get_plsql_logging_level, WNDS);
```

### Parameters

None.

### Usage Notes

The return value is the integer value denoting the current PL/SQL logging level.

### Examples

The following example gets the PL/SQL logging level.

```
--Contents of getplsqllogginglevel.sql
DECLARE
    loglevel    NUMBER;
BEGIN
    DBMS_MGD_ID_UTL.set_plsql_logging_level(0);
    loglevel := DBMS_MGD_ID_UTL.get_plsql_logging_level();
    dbms_output.put_line('PL/SQL logging level = ' || loglevel);
END;
/
SHOW ERRORS;

SQL> @getplsqllogginglevel.sql
.
.
.
PL/SQL logging level = 0
PL/SQL procedure successfully completed.
.
.
.
```

## GET\_SCHEME\_NAMES Function

This function returns a list of semicolon (;) separated scheme names for the specified category.

### Syntax

```
DBMS_MGD_ID_UTL.GET_SCHEME_NAMES (  
    category_id IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 89–10** *GET\_SCHEME\_NAMES Function Parameters*

Parameter	Description
category_id	Category ID

### Usage Notes

The return value contains the scheme names for the specified category ID.

### Examples

See the [GET\\_COMPONENTS Function](#) for an example.

## GET\_TDT\_XML Function

This function returns the Oracle Database tag data translation XML for the specified scheme.

### Syntax

```
DBMS_MGD_ID_UTL.GET_TDT_XML (
    category_id IN VARCHAR2,
    scheme_name IN VARCHAR2)
RETURN CLOB;
```

### Parameters

**Table 89-11** GET\_TDT\_XML Function Parameters

Parameter	Description
category_id	Category ID
scheme_name	Name of scheme

### Usage Notes

The return value contains the Oracle Database tag data translation XML for the specified scheme.

### Examples

The following example gets the Oracle Database TDT XML for the specified scheme:

```
--Contents of get_tdtxml.sql
DECLARE
    gettdtxml      CLOB;

BEGIN
    gettdtxml := DBMS_MGD_ID_UTL.get_tdt_xml(1, 'SGTIN-64');
    dbms_output.put_line('Length of tdt XML is '||DBMS_LOB.GETLENGTH(gettdtxml));
    dbms_output.put_line(DBMS_LOB.SUBSTR(gettdtxml, DBMS_LOB.GETLENGTH(gettdtxml), 1));
END;
/
SHOW ERRORS;

SQL> @get_tdtxml.sql
.
.
.
Length of tdt XML is 22884
<?xml version = '1.0' encoding = "UTF-8"?>
<TagDataTranslation version="0.04"
date="2005-04-18T16:05:00Z" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="oracle.mgd.idcode"><scheme name="SGTIN-64"
optionKey="companyprefixlength" xmlns="">
    <level type="BINARY"
prefixMatch="10" requiredFormattingParameters="filter">
        <option
optionKey="12" pattern="10([01]{3})([01]{14})([01]{20})([01]{25})" grammar="'10'
filter companyprefixindex itemref serial">
            <field seq="1"
decimalMinimum="0" decimalMaximum="7" characterSet="[01]*" bitLength="3"
```

```
length="1" padChar="0" padDir="LEFT" name="filter"/>
  <field seq="2"
decimalMinimum="0" decimalMaximum="16383" characterSet="[01]*" bitLength="14"
name="companyprefixindex"/>
  <field seq="3" decimalMinimum="0"
decimalMaximum="9" characterSet="[01]*" bitLength="20" length="1" padChar="0"
padDir="LEFT" name="itemref"/>
  <field seq="4" decimalMinimum="0"
decimalMaximum="33554431" characterSet="[01]*" bitLength="25" name="serial"/>
.
.
.
  <field seq="1" decimalMinimum="0" decimalMaximum="9999999" characterSet="[0-9]*"
length="7" padChar="0" padDir="LEFT" name="itemref"/>
  <field seq="2" decimalMinimum="0" decimalMaximum="999999" characterSet="[0-9]*" length="6"
padChar="0" padDir="LEFT" name="companyprefix"/>
  </option>
</level>

</scheme></TagDataTranslation>
PL/SQL procedure successfully completed.
.
.
.
```

## GET\_VALIDATOR Function

This function returns the Oracle Database tag data translation schema.

### Syntax

```
DBMS_MGD_ID_UTL.GET_VALIDATOR
RETURN CLOB;
```

### Usage Notes

The return value contains the Oracle Database tag data translation schema.

### Examples

This example returns the Oracle Database TDT schema.

```
--Contents of get_validator.sql
DECLARE
  getvalidator          CLOB;
BEGIN
  getvalidator := DBMS_MGD_ID_UTL.get_validator;
  dbms_output.put_line('Length of validated oracle scheme xml is '||DBMS_
LOB.GETLENGTH(getvalidator));
  dbms_output.put_line(DBMS_LOB.SUBSTR(getvalidator, DBMS_LOB.GETLENGTH(getvalidator), 1));
END;
/
SHOW ERRORS;

SQL> @get_validator.sql
.
.
.
Length of validated oracle scheme xml is 5780
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="oracle.mgd.idcode"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:tdt="oracle.mgd.idcode" elementFormDefault="unqualified"

attributeFormDefault="unqualified" version="1.0">
  <xsd:annotation>

<xsd:documentation>
  <![CDATA[
<epcglobal:copyright>Copyright ?2004
Epcglobal Inc., All
Rights
Reserved.</epcglobal:copyright>
<epcglobal:disclaimer>EPCglobal Inc., its
members, officers, directors,
employees, or agents shall not be liable for any
injury, loss, damages,
financial or otherwise, arising from, related to, or
caused by the use of this
document. The use of said document shall constitute
your express consent to
the foregoing
exculpation.</epcglobal:disclaimer>
```

```

<epcglobal:specification>Tag Data
Translation (TDT) version
1.0</epcglobal:specification>
]]>

</xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType
name="LevelTypeList">
    <xsd:restriction base="xsd:string">

</xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="TagLengthList"

<xsd:restriction base="xsd:string">
  </xsd:restriction>
  </xsd:simpleType>

<xsd:simpleType name="SchemeNameList">
  <xsd:restriction base="xsd:string">

</xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType
name="InputFormatList">
    <xsd:restriction base="xsd:string">

<xsd:enumeration value="BINARY"/>
  <xsd:enumeration value="STRING"/>

</xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ModeList">

<xsd:restriction base="xsd:string">
  <xsd:enumeration value="EXTRACT"/>

<xsd:enumeration value="FORMAT"/>
  </xsd:restriction>
  </xsd:simpleType>

<xsd:simpleType name="CompactionMethodList">
  <xsd:restriction
base="xsd:string">
    <xsd:enumeration value="32-bit"/>
<xsd:enumeration value="16-bit"/>
    <xsd:enumeration value="8-bit"/>

<xsd:enumeration value="7-bit"/>
  <xsd:enumeration value="6-bit"/>

<xsd:enumeration value="5-bit"/>
  </xsd:restriction>
  </xsd:simpleType>

<xsd:simpleType name="PadDirectionList">
  <xsd:restriction
base="xsd:string">
    <xsd:enumeration value="LEFT"/>

```

```

        <xsd:enumeration
value="RIGHT"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType
name="Field">
    <xsd:attribute name="seq" type="xsd:integer" use="required"/>

<xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute
name="bitLength" type="xsd:integer"/>
    <xsd:attribute name="characterSet"
type="xsd:string" use="required"/>
    <xsd:attribute name="compaction"
type="tdt:CompactionMethodList"/>
    <xsd:attribute name="compression"
type="xsd:string"/>
    <xsd:attribute name="padChar" type="xsd:string"/>

<xsd:attribute name="padDir" type="tdt:PadDirectionList"/>
    <xsd:attribute
name="decimalMinimum" type="xsd:long"/>
    <xsd:attribute name="decimalMaximum"
type="xsd:long"/>
    <xsd:attribute name="length" type="xsd:integer"/>

</xsd:complexType>
    <xsd:complexType name="Option">
        <xsd:sequence>

<xsd:element name="field" type="tdt:Field" maxOccurs="unbounded"/>

</xsd:sequence>
    <xsd:attribute name="optionKey" type="xsd:string"
use="required"/>
    <xsd:attribute name="pattern" type="xsd:string"/>

<xsd:attribute name="grammar" type="xsd:string" use="required"/>

</xsd:complexType>
    <xsd:complexType name="Rule">
        <xsd:attribute
name="type" type="tdt:ModeList" use="required"/>
        <xsd:attribute
name="inputFormat" type="tdt:InputFormatList"
use="required"/>
        <xsd:attribute name="seq" type="xsd:integer"
use="required"/>
        <xsd:attribute name="newFieldName" type="xsd:string"
use="required"/>
        <xsd:attribute name="characterSet" type="xsd:string"
use="required"/>
        <xsd:attribute name="padChar" type="xsd:string"/>

<xsd:attribute name="padDir" type="tdt:PadDirectionList"/>
    <xsd:attribute
name="decimalMinimum" type="xsd:long"/>
    <xsd:attribute name="decimalMaximum"
type="xsd:long"/>

```



```

    <xsd:attribute name="length" type="xsd:string"/>

<xsd:attribute name="function" type="xsd:string" use="required"/>

<xsd:attribute name="tableURI" type="xsd:string"/>
  <xsd:attribute
name="tableParams" type="xsd:string"/>
  <xsd:attribute name="tableXPath"
type="xsd:string"/>
  <xsd:attribute name="tableSQL" type="xsd:string"/>

</xsd:complexType>
  <xsd:complexType name="Level">
    <xsd:sequence>
<xsd:element name="option" type="tdt:Option" minOccurs="1"
maxOccurs="unbounded"/>
      <xsd:element name="rule" type="tdt:Rule"
minOccurs="0"
          maxOccurs="unbounded"/>
    </xsd:sequence>
<xsd:attribute name="type" type="tdt:LevelTypeList" use="required"/>
<xsd:attribute name="prefixMatch" type="xsd:string" use="optional"/>
<xsd:attribute name="requiredParsingParameters" type="xsd:string"/>
<xsd:attribute name="requiredFormattingParameters" type="xsd:string"/>

</xsd:complexType>
  <xsd:complexType name="Scheme">
    <xsd:sequence>
<xsd:element name="level" type="tdt:Level" minOccurs="1" maxOccurs="5"/>

</xsd:sequence>
    <xsd:attribute name="name" type="tdt:SchemeNameList"
use="required"/>
    <xsd:attribute name="optionKey" type="xsd:string"
use="required"/>
    <xsd:attribute name="tagLength" type="tdt:TagLengthList"
use="optional"/>
  </xsd:complexType>
  <xsd:complexType
name="TagDataTranslation">
    <xsd:sequence>
      <xsd:element name="scheme"
type="tdt:Scheme" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute
name="version" type="xsd:string" use="required"/>
    <xsd:attribute name="date"
type="xsd:dateTime" use="required"/>
  </xsd:complexType>
  <xsd:element
name="TagDataTranslation" type="tdt:TagDataTranslation"/>
</xsd:schema>

```

PL/SQL procedure successfully completed.

.  
.  
.

## REFRESH\_CATEGORY Function

This function refreshes the metadata information on the Java stack for the specified category. This function must be called before using MGD\_ID functions.

### Syntax

```
DBMS_MGD_ID_UTL.REFRESH_CATEGORY (
    category_id IN VARCHAR2);
```

### Parameters

**Table 89–12 REFRESH\_CATEGORY Function Parameters**

Parameter	Description
category_id	Category ID

### Examples

The following example refreshes the metadata information for the EPC category ID.

```
--Contents of tostring3.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
DECLARE
    id          MGD_ID;
BEGIN
    DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
    DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
    dbms_output.put_line('..Testing to_string');
    DBMS_OUTPUT.PUT_LINE('test to_string');
    id := mgd_id('EPC', NULL, 'urn:epc:id:gid:0037000.30241.1041970',
'scheme=GID-96');
    DBMS_OUTPUT.PUT_LINE('mgd_id object as a string');
    DBMS_OUTPUT.PUT_LINE(id.to_string);
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
```

```
SQL> @tostring3.sql
..Testing to_string
test to_string
mgd_id object as a string
category_id =1;schemes = GID-96;objectclass = 30241;generalmanager =
0037000;scheme = GID-96;1 = 1;serial = 1041970
```

PL/SQL procedure successfully completed.

## REMOVE\_CATEGORY Procedure

This procedure removes a category including all the related TDT XML. This procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Removes a category based on the specified category ID.

```
DBMS_MGD_ID_UTL.REMOVE_CATEGORY (
    category_id      IN VARCHAR2);
```

Removes a category based on the specified category name and category version.

```
DBMS_MGD_ID_UTL.REMOVE_CATEGORY (
    category_name    IN VARCHAR2,
    category_version IN VARCHAR2);
```

### Parameters

**Table 89–13 REMOVE\_CATEGORY Procedure Parameters**

Parameter	Description
category_id	Category ID
category_name	Name of category
category_version	Category version

### Usage Notes

If the value of `category_version` is `NULL`, all versions for the specified category will be removed.

### Examples

See the [ADD\\_SCHEME Procedure](#) for an example of removing a category.

## REMOVE\_PROXY Procedure

This procedure unsets the host and port of the proxy server.

### Syntax

```
DBMS_MGD_ID_UTL.REMOVE_PROXY;
```

### Examples

See the [REFRESH\\_CATEGORY Function](#) for an example.

## REMOVE\_SCHEME Procedure

This procedure removes a tag data translation scheme from a category.

### Syntax

```
DBMS_MGD_ID_UTL.REMOVE_SCHEME (  
  category_id IN VARCHAR2,  
  scheme_name IN VARCHAR2);
```

### Parameters

**Table 89–14 REMOVE\_SCHEME Procedure Parameters**

Parameter	Description
category_id	Category ID
scheme_name	Name of scheme

### Examples

See the [ADD\\_SCHEME Procedure](#) for an example of removing a scheme.

## SET\_JAVA\_LOGGING\_LEVEL Procedure

This procedure sets the Java trace logging level.

### Syntax

```
DBMS_MGD_ID_UTL.SET_JAVA_LOGGING_LEVEL (
    logginglevel IN INTEGER);
```

### Parameters

**Table 89–15 SET\_JAVA\_LOGGING\_LEVEL Procedure Parameters**

Parameter	Description
logginglevel	Logging level. The Java logging level can be one of the following values in descending order: <ul style="list-style-type: none"> <li>▪ LOGGING_LEVEL_OFF                    CONSTANT INTEGER := 0</li> <li>▪ LOGGING_LEVEL_SEVERE                CONSTANT INTEGER := 1</li> <li>▪ LOGGING_LEVEL_WARNING               CONSTANT INTEGER := 2</li> <li>▪ LOGGING_LEVEL_INFO                  CONSTANT INTEGER := 3</li> <li>▪ LOGGING_LEVEL_FINE                  CONSTANT INTEGER := 4</li> <li>▪ LOGGING_LEVEL_FINER                 CONSTANT INTEGER := 5</li> <li>▪ LOGGING_LEVEL_FINEST                CONSTANT INTEGER := 6</li> <li>▪ LOGGING_LEVEL_ALL                    CONSTANT INTEGER := 7</li> </ul>

### Examples

See the [GET\\_JAVA\\_LOGGING\\_LEVEL Function](#) for an example.

## SET\_PLSQL\_LOGGING\_LEVEL Procedure

This procedure sets the PL/SQL trace logging level.

### Syntax

```
DBMS_MGD_ID_UTL.SET_PLSQL_LOGGING_LEVEL (
    level IN INTEGER);

PRAGMA restrict_references(set_plsql_logging_level, WNDS);
```

### Parameters

**Table 89–16** *SET\_PLSQL\_LOGGING\_LEVEL Procedure Parameters*

Parameter	Description
level	Logging level. The PL/SQL logging level can be one of the following values in descending order: <ul style="list-style-type: none"> <li>▪ LOGGING_LEVEL_OFF                    CONSTANT INTEGER := 0</li> <li>▪ LOGGING_LEVEL_SEVERE                CONSTANT INTEGER := 1</li> <li>▪ LOGGING_LEVEL_WARNING               CONSTANT INTEGER := 2</li> <li>▪ LOGGING_LEVEL_INFO                  CONSTANT INTEGER := 3</li> <li>▪ LOGGING_LEVEL_FINE                  CONSTANT INTEGER := 4</li> <li>▪ LOGGING_LEVEL_FINER                 CONSTANT INTEGER := 5</li> <li>▪ LOGGING_LEVEL_FINEST                CONSTANT INTEGER := 6</li> <li>▪ LOGGING_LEVEL_ALL                    CONSTANT INTEGER := 7</li> </ul>

### Examples

See the [GET\\_PLSQL\\_LOGGING\\_LEVEL Function](#) for an example.

## SET\_PROXY Procedure

This procedure sets the host and port of the proxy server for Internet access. This procedure must be called if the database server accesses the Internet using a proxy server. Internet access is necessary because some rules need to look up the Object Naming Service (ONS) table to get the company prefix index.

You do not need to call this procedure if you are only using schemes that do not contain any rules requiring Internet access.

### Syntax

```
DBMS_MGD_ID_UTL.SET_PROXY (  
    prox_host IN VARCHAR2,  
    proxy_port IN VARCHAR2);
```

### Parameters

**Table 89–17** SET\_PROXY Procedure Parameters

Parameter	Description
proxy_host	Name of host
proxy_port	Host port number

### Examples

See the [REFRESH\\_CATEGORY Function](#) for an example.



## VALIDATE\_SCHEME Function

This function validates the input tag data translation XML against the Oracle Database tag data translation schema.

### Syntax

```
DBMS_MGD_ID_UTL.VALIDATE_SCHEME (  
    xml_scheme IN CLOB)  
RETURN VARCHAR2;
```

### Parameters

**Table 89–18** VALIDATE\_SCHEME Function Parameters

Parameter	Description
xml_scheme	Scheme to be validated.

### Usage Notes

The return value contains the components names for the specified scheme.

### Examples

See the [ADD\\_SCHEME Procedure](#) or the [EPC\\_TO\\_ORACLE\\_SCHEME Function](#) for an example.



---

---

## DBMS\_MGWADM

DBMS\_MGWADM defines the Messaging Gateway administrative interface. The package and object types are owned by SYS.

---

---

**Note:** You must run the `catmgw.sql` script to load the Messaging Gateway packages and types into the database.

---

---

**See Also:** *Oracle Streams Advanced Queuing User's Guide* contains information on loading database objects and using DBMS\_MGWADM

This chapter contains the following topics:

- [Using DBMS\\_MGWADM](#)
  - Constants
  - Deprecated Subprograms
- [Data Structures](#)
- [Summary of DBMS\\_MGWADM Subprograms](#)

---

## Using DBMS\_MGWADM

- [Deprecated Subprograms](#)
- [Constants](#)

## Deprecated Subprograms

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 11g Release 1 having been superseded by improved technology:

- [ADD\\_SUBSCRIBER Procedure](#) - use instead [CREATE\\_JOB Procedure](#)
- [ALTER\\_PROPAGATION\\_SCHEDULE Procedure](#) - use instead [ALTER\\_JOB Procedure](#)
- [ALTER\\_SUBSCRIBER Procedure](#) - use instead [ALTER\\_JOB Procedure](#)
- [DB\\_CONNECT\\_INFO Procedure](#) - use instead [ALTER\\_AGENT Procedures](#)
- [DISABLE\\_PROPAGATION\\_SCHEDULE Procedure](#) - use instead [DISABLE\\_JOB Procedure](#)
- [ENABLE\\_PROPAGATION\\_SCHEDULE Procedure](#) - use instead [ENABLE\\_JOB Procedure](#)
- [REMOVE\\_SUBSCRIBER Procedure](#) - use instead [REMOVE\\_JOB Procedure](#)
- [RESET\\_SUBSCRIBER Procedure](#) - use instead [RESET\\_JOB Procedure](#)
- [SCHEDULE\\_PROPAGATION Procedure](#) - use instead [CREATE\\_JOB Procedure](#)
- [UNSCHEDULE\\_PROPAGATION Procedure](#) - use instead [REMOVE\\_JOB Procedure](#)

## Constants

- [DBMS\\_MGWADM Constants—Cleanup Actions](#) on page 90-4
- [DBMS\\_MGWADM Constants—Force Values](#) on page 90-4
- [DBMS\\_MGWADM Constants—Logging Levels](#) on page 90-4
- [DBMS\\_MGWADM Constants—Named Property Constants](#) on page 90-5
- [DBMS\\_MGWADM Constants—Other Constants](#) on page 90-5
- [DBMS\\_MGWADM Constants—Propagation Types](#) on page 90-6
- [DBMS\\_MGWADM Constants—Queue Domain Types](#) on page 90-6
- [DBMS\\_MGWADM Constants—Shutdown Modes](#) on page 90-6
- [DBMS\\_MGWADM Constants—WebSphere MQ Interface Types](#) on page 90-6
- [DBMS\\_MGWADM Constants—target\\_type Argument of SET\\_OPTION and REMOVE\\_OPTION Procedures](#) on page 90-6
- [DBMS\\_MGWADM Constants—conntype Argument of CREATE\\_AGENT and ALTER\\_AGENT Procedures](#) on page 90-6

**Table 90–1 DBMS\_MGWADM Constants—Cleanup Actions**

Name	Type	Description
CLEAN_STARTUP_STATE	CONSTANT BINARY_INTEGER	Sets the Messaging Gateway agent to a known state so that it can be started
CLEAN_LOG_QUEUES	CONSTANT BINARY_INTEGER	Messaging Gateway agent will clean log queues for all configured messaging system links
RESET_SUB_MISSING_LOG_REC	CONSTANT BINARY_INTEGER	Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing log record
RESET_SUB_MISSING_MESSAGE	CONSTANT BINARY_INTEGER	Messaging Gateway agent recovers a Messaging Gateway subscriber that has failed due to a missing persistent source message

**Table 90–2 DBMS\_MGWADM Constants—Force Values**

Name	Type	Description
FORCE	CONSTANT BINARY_INTEGER	Represents a forced action
NO_FORCE	CONSTANT BINARY_INTEGER	Represents a normal, nonforced action

**Table 90–3 DBMS\_MGWADM Constants—Logging Levels**

Name	Type	Description
BASIC_LOGGING	CONSTANT BINARY_INTEGER	The standard (the least) information written to the log file

**Table 90–3 (Cont.) DBMS\_MGWADM Constants—Logging Levels**

Name	Type	Description
TRACE_DEBUG_LOGGING	CONSTANT BINARY_INTEGER	The greatest information written to the log file
TRACE_HIGH_LOGGING	CONSTANT BINARY_INTEGER	The third level of detail of logging information written to the log file
TRACE_LITE_LOGGING	CONSTANT BINARY_INTEGER	The second level detail of logging information written to the log file

**Table 90–4 DBMS\_MGWADM Constants—Named Property Constants**

Name	Type	Description
MGWPROP_PREFIX	CONSTANT VARCHAR2	A constant (MGWPROP\$_) for the reserved property name prefix
MGWPROP_REMOVE	CONSTANT VARCHAR2	A constant (MGWPROP\$_REMOVE) for the reserved property name used to remove an existing property
MGWPROP_REMOVE_ALL	CONSTANT VARCHAR2	A constant (MGWPROP\$_REMOVE_ALL) for the reserved property name used to remove all properties

**Table 90–5 DBMS\_MGWADM Constants—Other Constants**

Name	Type	Description
JMS_CONNECTION	CONSTANT BINARY_INTEGER	Used to indicate that JMS connections will be used to access JMS destinations in a domain-independent manner that supports a unified messaging model
JMS_QUEUE_CONNECTION	CONSTANT BINARY_INTEGER	Used to indicate that JMS queue connections will be used to access JMS destinations
JMS_TOPIC_CONNECTION	CONSTANT BINARY_INTEGER	Used to indicate that JMS topic connections will be used to access JMS destinations
NO_CHANGE	CONSTANT VARCHAR2	Indicates that an existing value should be preserved (not changed). This is used for certain APIs where the desire is to change one or more parameters but leave others unchanged.
DEFAULT_AGENT	CONSTANT VARCHAR2	Name of the Messaging Gateway default agent

**Table 90–6 DBMS\_MGWADM Constants—Propagation Types**

Name	Type	Description
INBOUND_PROPAGATION	CONSTANT BINARY_INTEGER	Represents the propagation type for non-Oracle to Oracle Streams AQ propagation. The propagation source is a queue in a foreign (non-Oracle) messaging system and the destination is a local Oracle Streams AQ queue.
OUTBOUND_PROPAGATION	CONSTANT BINARY_INTEGER	Represents the propagation type for Oracle Streams AQ to non-Oracle propagation. The propagation source is a local Oracle Streams AQ queue and the destination is a queue in a foreign (non-Oracle) messaging system.

**Table 90–7 DBMS\_MGWADM Constants—Queue Domain Types**

Name	Type	Description
DOMAIN_QUEUE	CONSTANT BINARY_INTEGER	Represents a queue destination. A JMS queue (point-to-point model) is classified as a queue.
DOMAIN_TOPIC	CONSTANT BINARY_INTEGER	Represents a topic destination. A JMS topic (publish-subscribe model) is classified as a topic.

**Table 90–8 DBMS\_MGWADM Constants—Shutdown Modes**

Name	Type	Description
SHUTDOWN_IMMEDIATE	CONSTANT BINARY_INTEGER	Represents the immediate shutdown mode
SHUTDOWN_NORMAL	CONSTANT BINARY_INTEGER	Represents the normal shutdown mode

**Table 90–9 DBMS\_MGWADM Constants—WebSphere MQ Interface Types**

Name	Type	Description
MQSERIES_BASE_JAVA_INTERFACE	CONSTANT BINARY_INTEGER	Represents the Base Java interface for the WebSphere MQ messaging system

**Table 90–10 DBMS\_MGWADM Constants—target\_type Argument of SET\_OPTION and REMOVE\_OPTION Procedures**

Name	Type	Description
AGENT_JAVA_PROP	CONSTANT PLS_INTEGER	Used for an agent option used to set a Java System property
MSGLINK_OPTION	CONSTANT PLS_INTEGER	Used for a messaging system link option
JOB_OPTION	CONSTANT PLS_INTEGER	Used for a propagation job option

**Table 90–11 DBMS\_MGWADM Constants—conntype Argument of CREATE\_AGENT and ALTER\_AGENT Procedures**

Name	Type	Description
JDBC_OCI	CONSTANT VARCHAR2	Used to specify the JDBC OCI driver
JDBC_THIN	CONSTANT VARCHAR2	Used to specify the JDBC Thin driver



## Data Structures

---

The DBMS\_MGWADM package defines the following OBJECT types.

### Object Types

- [SYS.MGW\\_MQSERIES\\_PROPERTIES](#) Object Type
- [SYS.MGW\\_PROPERTIES](#) Object Type
- [SYS.MGW\\_PROPERTY](#) Object Type
- [SYS.MGW\\_TIBRV\\_PROPERTIES](#) Object Type

## SYS.MGW\_MQSERIES\_PROPERTIES Object Type

This type specifies basic properties for a WebSphere MQ messaging system link.

### Syntax

```

TYPE SYS.MGW_MQSERIES_PROPERTIES IS OBJECT (
    queue_manager      VARCHAR2(64),
    hostname           VARCHAR2(64),
    port              INTEGER,
    channel            VARCHAR2(64),
    interface_type     INTEGER,
    max_connections   INTEGER,
    username           VARCHAR2(64),
    password           VARCHAR2(64),
    inbound_log_queue  VARCHAR2(64),
    outbound_log_queue VARCHAR2(64),

    -- Methods
    STATIC FUNCTION construct
    RETURN SYS.MGW_MQSERIES_PROPERTIES,

    STATIC FUNCTION alter_construct
    RETURN SYS.MGW_MQSERIES_PROPERTIES );

```

### Attributes

**Table 90–12** SYS.MGW\_MQSERIES\_PROPERTIES Attributes

Attribute	Description
queue_manager	The name of the WebSphere MQ queue manager
hostname	The host on which the WebSphere MQ messaging system resides. If hostname is NULL, then a WebSphere MQ bindings connection is used. If not NULL, then a client connection is used and requires that a port and channel be specified.
port	The port number. This is used only for client connections; that is, when hostname is not NULL.
channel	The channel used when establishing a connection to the queue manager. This is used only for client connections; that is, when hostname is not NULL.
interface_type	The type of messaging interface to use. Values: <ul style="list-style-type: none"> <li>■ DBMS_MGWADM.MQSERIES_BASE_JAVA_INTERFACE if the WebSphere MQ Base Java interface should be used.</li> <li>■ DBMS_MGWADM.JMS_CONNECTION if the link is to be used to access JMS destinations in a unified, domain-independent manner.</li> <li>■ DBMS_MGWADM.JMS_QUEUE_CONNECTION if the link is to be used for accessing JMS queues</li> <li>■ DBMS_MGWADM.JMS_TOPIC_CONNECTION if the link is to be used for accessing JMS topics.</li> </ul>
max_connections	The maximum number of messaging connections to the WebSphere MQ messaging system
username	The username used for authentication to the WebSphere MQ messaging system

**Table 90–12 (Cont.) SYS.MGW\_MQSERIES\_PROPERTIES Attributes**

Attribute	Description
password	The password used for authentication to the WebSphere MQ messaging system
inbound_log_queue	<p>The name of the WebSphere MQ queue used for propagation recovery purposes when this messaging link is used for inbound propagation; that is, when queues associated with this link serve as a propagation source:</p> <ul style="list-style-type: none"> <li>■ For MQSERIES_BASE_JAVA_INTERFACE, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</li> <li>■ For the JMS_CONNECTION interface and the JMS_QUEUE_CONNECTION interface, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</li> <li>■ For JMS_TOPIC_CONNECTION interface, this specifies the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by subscribers of that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.</li> </ul>
outbound_log_queue	<p>The name of the WebSphere MQ queue used for propagation recovery purposes when this messaging link is used for outbound propagation; that is, when queues associated with this link serve as a propagation destination:</p> <ul style="list-style-type: none"> <li>■ For MQSERIES_BASE_JAVA_INTERFACE, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</li> <li>■ For the JMS_CONNECTION interface and the JMS_QUEUE_CONNECTION interface, this is the name of a physical WebSphere MQ queue created using WebSphere MQ administration tools.</li> <li>■ For JMS_TOPIC_CONNECTION interface, this specifies the name of a WebSphere MQ JMS topic. The physical WebSphere MQ queue used by subscribers of that topic must be created using WebSphere MQ administration tools. By default, the physical queue used is SYSTEM.JMS.D.SUBSCRIBER.QUEUE.</li> </ul>

## Methods

**Table 90–13 SYS.MGW\_MQSERIES\_PROPERTIES Methods**

Method	Description
construct	Constructs a new SYS.MGW_MQSERIES_PROPERTIES instance. All attributes are assigned a value of NULL
alter_construct	Constructs a new SYS.MGW_MQSERIES_PROPERTIES instance for altering the properties of an existing messaging link. All attributes having a VARCHAR2 data type are assigned a value of DBMS_MGWADM.NO_CHANGE. Attributes of other data types are assigned a value of NULL.

## SYS.MGW\_PROPERTIES Object Type

This type specifies an array of properties.

### Syntax

```
TYPE SYS.MGW_PROPERTIES AS VARRAY (2000) OF SYS.MGW_PROPERTY;
```

### Attributes

**Table 90–14 SYS.MGW\_PROPERTIES Attributes**

Attribute	Description
name	Property name
value	Property value

### Usage Notes

Unless noted otherwise, Messaging Gateway uses named properties as follows:

- Names with the MGWPROP\$\_ prefix are reserved. They are used for special purposes and are invalid when used as a normal property name.
- A property name can exist only once in a property list; that is, a list can contain only one value for a given name. The name is case-insensitive.
- In general, a property list is order-independent, and the property names may appear in any order. An alter property list is an exception.
- You can use a new property list to alter an existing property list. Each new property modifies the original list in one of the following ways: adds a new property, modifies a property, removes a property, or removes all properties.

The alter list is processed in order, from the first element to the last element. Thus the order in which the elements appear in the alter list is meaningful, especially when the alter list is used to remove properties from an existing list.

The property name and value are used to determine how that element affects the original list. The following rules apply:

- Add or modify property

```
MGW_PROPERTY.NAME = property_name
MGW_PROPERTY.VALUE = property_value
```

If a property of the given name already exists, then the current value is replaced with the new value; otherwise the new property is added to the end of the list.

- Remove property

```
MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE'
MGW_PROPERTY.VALUE = name_of_property_to_remove
```

No action is taken if the property name does not exist in the original list.

- Remove all properties

```
MGW_PROPERTY.NAME = 'MGWPROP$_REMOVE_ALL'
MGW_PROPERTY.VALUE = not used
```

**See Also:** "The DBMS\_MGWADM package defines constants to represent the reserved property names on [Table 90-4](#), "DBMS\_MGWADM Constants—Named Property Constants"

## SYS.MGW\_PROPERTY Object Type

This type specifies a named property which is used to specify optional properties for messaging links, foreign queues, and subscribers.

### Syntax

```
TYPE SYS.MGW_PROPERTY IS OBJECT(
  name  VARCHAR2(500),
  value VARCHAR2(4000),

  -- Methods
  STATIC FUNCTION construct  --- (1)
  RETURN SYS.MGW_PROPERTY,

  STATIC FUNCTION construct(  --- (2)
    p_name  IN VARCHAR2,
    p_value IN VARCHAR2)
  RETURN SYS.MGW_PROPERTY );
```

### Attributes

**Table 90–15 SYS.MGW\_PROPERTY Attributes**

Attribute	Description
name	Property name
value	Property value

### Methods

**Table 90–16 SYS.MGW\_PROPERTY Methods**

Method	Description
construct --- (1)	Constructs a new MGW_PROPERTY instance. All attributes are assigned a value of NULL
construct --- (2)	Constructs a new MGW_PROPERTY instance initialized using the given parameters

## SYS.MGW\_TIBRV\_PROPERTIES Object Type

A type that specifies basic properties for a TIB/Rendezvous messaging system link. The Messaging Gateway agent creates a TIB/Rendezvous transport of type `TibrvRvdTransport` for each Messaging Gateway link.

### Syntax

```
TYPE SYS.MGW_TIBRV_PROPERTIES IS OBJECT(
  service  VARCHAR2(128),
  daemon   VARCHAR2(128),
  network  VARCHAR2(256),
  cm_name  VARCHAR2(256),
  cm_ledger VARCHAR2(256),

  -- Methods
  STATIC FUNCTION construct
  RETURN SYS.MGW_TIBRV_PROPERTIES,

  STATIC FUNCTION alter_construct
  RETURN SYS.MGW_TIBRV_PROPERTIES );
```

### Attributes

**Table 90–17 SYS.MGW\_TIBRV\_PROPERTIES Attributes**

Attribute	Description
<code>service</code>	The service parameter for the rvd transport
<code>daemon</code>	The daemon parameter for the rvd transport
<code>network</code>	The network parameter for the rvd transport
<code>cm_name</code>	The CM correspondent name. Reserved for future use.
<code>cm_ledger</code>	The CM ledger file name. Reserved for future use.

### Methods

**Table 90–18 SYS.MGW\_TIBRV\_PROPERTIES Methods**

Method	Description
<code>construct</code>	Constructs a new <code>SYS.MGW_TIBRV_PROPERTIES</code> instance. All attributes will be assigned a value of <code>NULL</code> .
<code>alter_construct</code>	Constructs a new <code>SYS.MGW_TIBRV_PROPERTIES</code> instance. This function is useful for altering the properties of an existing messaging link. All attributes having a <code>VARCHAR2</code> data type will be assigned a value of <code>DBMS_MGWADM.NO_CHANGE</code> . Attributes of other data types will be assigned a value of <code>NULL</code> .

---

## Summary of DBMS\_MGWADM Subprograms

**Table 90–19 DBMS\_MGWADM Package Subprograms**

Subprogram	Description
<a href="#">ADD_SUBSCRIBER Procedure</a> on page 90-16	Adds a subscriber used to consume messages from a source queue for propagation to a destination
<a href="#">ALTER_AGENT Procedures</a> on page 90-20	Alters Messaging Gateway agent parameters
<a href="#">ALTER_JOB Procedure</a> on page 90-22	Alters the properties of a propagation job
<a href="#">ALTER_MSGSYSTEM_LINK Procedure for TIB/Rendezvous</a> on page 90-24	Alters the properties of a TIB/Rendezvous messaging system link
<a href="#">ALTER_MSGSYSTEM_LINK Procedure for WebSphere MQ</a> on page 90-25	Alters the properties of a WebSphere MQ messaging system link
<a href="#">ALTER_PROPAGATION_SCHEDULE Procedure</a> on page 90-26	Alters a propagation schedule
<a href="#">ALTER_SUBSCRIBER Procedure</a> on page 90-27	Alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination
<a href="#">CLEANUP_GATEWAY Procedures</a> on page 90-29	Cleans up Messaging Gateway
<a href="#">CREATE_AGENT Procedure</a> on page 90-32	Creates a Messaging Gateway agent that will be used to process propagation jobs
<a href="#">CREATE_JOB Procedure</a> on page 90-34	Creates a job used to propagate message from a source to a destination
<a href="#">CREATE_MSGSYSTEM_LINK Procedures for TIB/Rendezvous</a> on page 90-38	Creates a messaging system link to a TIB/Rendezvous messaging system
<a href="#">CREATE_MSGSYSTEM_LINK Procedures for WebSphere MQ</a> on page 90-39	Creates a messaging system link to a WebSphere MQ messaging system
<a href="#">DB_CONNECT_INFO Procedure</a> on page 90-40	Configures connection information used by the Messaging Gateway agent for connections to Oracle Database
<a href="#">DISABLE_JOB Procedure</a> on page 90-41	Disables a propagation job
<a href="#">DISABLE_PROPAGATION_SCHEDULE Procedure</a> on page 90-42	Disables a propagation schedule
<a href="#">ENABLE_JOB Procedure</a> on page 90-43	Enables a propagation job
<a href="#">ENABLE_PROPAGATION_SCHEDULE Procedure</a> on page 90-44	Enables a propagation schedule
<a href="#">REGISTER_FOREIGN_QUEUE Procedure</a> on page 90-45	Registers a non-Oracle queue entity in Messaging Gateway



**Table 90–19 (Cont.) DBMS\_MGWADM Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">REMOVE_AGENT Procedure</a> on page 90-46	Removes a Messaging Gateway agent
<a href="#">REMOVE_JOB Procedure</a> on page 90-47	Removes a propagation job
<a href="#">REMOVE_MSGSYSTEM_LINK Procedure</a> on page 90-48	Removes a messaging system link for a non-Oracle messaging system
<a href="#">REMOVE_OPTION Procedure</a> on page 90-49	Removes a Messaging Gateway configuration option
<a href="#">REMOVE_SUBSCRIBER Procedure</a> on page 90-51	Removes a subscriber used to consume messages from a source queue for propagation to a destination
<a href="#">RESET_JOB Procedure</a> on page 90-52	Resets the propagation error state for a propagation job
<a href="#">RESET_SUBSCRIBER Procedure</a> on page 90-53	Resets the propagation error state for a subscriber
<a href="#">SCHEDULE_PROPAGATION Procedure</a> on page 90-54	Schedules message propagation from a source to a destination
<a href="#">SET_LOG_LEVEL Procedures</a> on page 90-56	Dynamically alters the Messaging Gateway agent logging level
<a href="#">SET_OPTION Procedure</a> on page 90-57	Sets a Messaging Gateway configuration option
<a href="#">SHUTDOWN Procedures</a> on page 90-59	Shuts down the Messaging Gateway agent
<a href="#">STARTUP Procedures</a> on page 90-60	Starts the Messaging Gateway agent
<a href="#">UNREGISTER_FOREIGN_QUEUE Procedure</a> on page 90-61	Removes a non-Oracle queue entity in Messaging Gateway
<a href="#">UNSCHEDULE_PROPAGATION Procedure</a> on page 90-62	Removes a propagation schedule

## ADD\_SUBSCRIBER Procedure

This procedure adds a subscriber used to consume messages from a source queue for propagation to a destination.

---



---

**Note:** This subprogram has been deprecated as a result of improved technology (see [CREATE\\_JOB Procedure](#) on page 90-34), and is retained only for reasons of backward compatibility.

---



---

### Syntax

```
DBMS_MGWADM.ADD_SUBSCRIBER(
  subscriber_id      IN VARCHAR2,
  propagation_type  IN BINARY_INTEGER,
  queue_name        IN VARCHAR2,
  destination       IN VARCHAR2,
  rule              IN VARCHAR2 DEFAULT NULL,
  transformation     IN VARCHAR2 DEFAULT NULL,
  exception_queue   IN VARCHAR2 DEFAULT NULL,
  options           IN SYS.MGW_PROPERTIES DEFAULT NULL);
```

### Parameters

**Table 90–20 ADD\_SUBSCRIBER Procedure Parameters**

Parameter	Description
subscriber_id	Specifies a user-defined name that identifies this subscriber
propagation_type	Specifies the type of message propagation. <code>DBMS_MGWADM.OUTBOUND_PROPAGATION</code> is for Oracle Streams AQ to non-Oracle propagation. <code>DBMS_MGWADM.INBOUND_PROPAGATION</code> is for non-Oracle to Oracle Streams AQ propagation
queue_name	Specifies the source queue to which this subscriber is being added. The syntax and interpretation of this parameter depend on the value specified for <code>propagation_type</code> .
destination	Specifies the destination queue to which messages consumed by this subscriber are propagated. The syntax and interpretation of this parameter depend on the value specified for <code>propagation_type</code> .
rule	Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. This is <code>NULL</code> if no rule is needed. The syntax and interpretation of this parameter depend on the value specified for <code>propagation_type</code> .
transformation	Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for <code>propagation_type</code> .  If <code>NULL</code> , then the Oracle Streams AQ payload type must be supported by Messaging Gateway.

**Table 90–20 (Cont.) ADD\_SUBSCRIBER Procedure Parameters**

Parameter	Description
exception_queue	Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If NULL, then an exception queue is not used and propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the value specified for propagation_type.  The source queue and exception queue cannot be the same queue.
options	Optional subscriber properties. NULL if there are none. Typically these are lesser used configuration properties supported by the messaging system.

## Usage Notes

**See Also:** "Handling Arbitrary Payload Types Using Message Transformations", in *Oracle Streams Advanced Queuing User's Guide* for more information regarding message conversion and transformation

If the non-Oracle messaging link being accessed for the subscriber uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be NULL.

**See Also:** For additional information regarding subscriber options

- "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide*
- "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide*

### OUTBOUND\_PROPAGATION Subscribers

The parameters for a subscriber used for outbound propagation are interpreted as follows:

- queue\_name specifies the local Oracle Streams AQ queue that is the propagation source. This must have a syntax of schema.queue.
- destination specifies the foreign queue to which messages are propagated. This must have a syntax of registered\_queue@message\_link.
- rule specifies an optional Oracle Streams AQ subscriber rule if the native Oracle Streams AQ interface is used, or a JMS selector if the Oracle JMS interface is used. If NULL, then no rule or selector is used.
- transformation specifies the transformation used to convert the Oracle Streams AQ payload to an ADT defined by Messaging Gateway.

Messaging Gateway propagation dequeues messages from the Oracle Streams AQ queue using the transformation to convert the Oracle Streams AQ payload to a

known ADT defined by Messaging Gateway. The message is then enqueued in the foreign messaging system based on the Messaging Gateway ADT.

- `exception_queue` specifies the name of a local Oracle Streams AQ queue to which messages are moved if an exception occurs. This must have a syntax of `schema.queue`.

If the native Oracle Streams AQ interface is used, then a subscriber will be added to the Oracle Streams AQ queue when this procedure is called, whether or not Messaging Gateway is running. The local subscriber will be of the form `sys.aq$_agent('MGW_subscriber_id', NULL, NULL)`.

If the Oracle JMS interface is used, then the Messaging Gateway agent will create a JMS durable subscriber with the name of `MGW_subscriber_id`. If the agent is not running when this procedure is called, then the durable subscriber will be created the next time the agent starts.

The exception queue has the following caveats:

- The user is responsible for creating the Oracle Streams AQ queue to be used as the exception queue.
- The payload type of the source and exception queue must match.
- The exception queue must be created as a queue type of `DBMS_AQADM.NORMAL_QUEUE` rather than `DBMS_AQADM.EXCEPTION_QUEUE`. Enqueue restrictions prevent Messaging Gateway propagation from using an Oracle Streams AQ queue of type `EXCEPTION_QUEUE` as a Messaging Gateway exception queue.

### **INBOUND\_PROPAGATION Subscribers**

The parameters for a subscriber used for inbound propagation are interpreted as follows:

- `queue_name` specifies the foreign queue that is the propagation source. This must have a syntax of `registered_queue@message_link`.
- `destination` specifies the local Oracle Streams AQ queue to which messages are propagated. This must have a syntax of `schema.queue`.
- `rule` specifies an optional subscriber rule that is valid for the foreign messaging system. This is `NULL` if no rule is needed.
- `transformation` specifies the transformation used to convert an ADT defined by Messaging Gateway to the Oracle Streams AQ payload type.

Messaging Gateway propagation dequeues messages from the foreign messaging system and converts the message body to a known ADT defined by Messaging Gateway. The transformation is used to convert the Messaging Gateway ADT to an Oracle Streams AQ payload type when the message is enqueued to the Oracle Streams AQ queue.

- `exception_queue` specifies the name of a foreign queue to which messages are moved if an exception occurs. This must have a syntax of `registered_queue@message_link`.

Whether or not a subscriber is needed depends on the requirements of the non-Oracle messaging system. If a durable subscriber is necessary, then it will be created by the Messaging Gateway agent. If the agent is not running at the time this procedure is called, then the creation of the subscriber on the non-Oracle messaging system will occur when the agent next starts.

The exception queue has the following caveats:

- The exception queue must be a registered non-Oracle queue.
- The source and exception queues must use the same messaging system link.

## ALTER\_AGENT Procedures

This procedure configures Messaging Gateway agent parameters.

### Syntax

```
DBMS_MGWADM.ALTER_AGENT (
    max_connections IN BINARY_INTEGER DEFAULT NULL,
    max_memory      IN BINARY_INTEGER DEFAULT NULL,
    max_threads     IN BINARY_INTEGER DEFAULT NULL,
    service         IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

```
DBMS_MGWADM.ALTER_AGENT (
    agent_name      IN VARCHAR2,
    username        IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    password        IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    database        IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    conntype        IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    max_memory      IN PLS_INTEGER DEFAULT NULL,
    max_threads     IN PLS_INTEGER DEFAULT NULL,
    service         IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    initfile        IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
    comment         IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

### Parameters

**Table 90–21 ALTER\_AGENT Procedure Parameters**

Parameter	Description
max_connections	The maximum number of messaging connections to Oracle Database used by the Messaging Gateway agent. If it is NULL, then the current value is unchanged.  <b>Caution: This parameter has been deprecated.</b>
max_memory	The maximum heap size, in MB, used by the Messaging Gateway agent. If it is NULL, then the current value is unchanged.
max_threads	The number of messaging threads that the Messaging Gateway agent creates. If it is NULL, then the current value is unchanged.
service	Specifies the database service that the Oracle Scheduler job class used by this agent will have affinity to. In an Oracle RAC environment, this means that the Messaging Gateway agent will run on only those database instances that are assigned to the service. If NULL, the job class used by this agent will be altered to belong to the default service which is mapped to every instance. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.
agent_name	Identifies the Messaging Gateway agent. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.
username	Specifies the username used for connections to the Oracle Database. NULL is not allowed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged. If a username is specified then a password must also be specified.

**Table 90–21 (Cont.) ALTER\_AGENT Procedure Parameters**

Parameter	Description
password	Specifies the password used for connections to the Oracle Database. NULL is not allowed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged. A password must be specified if a username is specified.
database	Specifies the database connect string used for connections to the Oracle Database. NULL indicates that a local connection should be used. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.  Oracle strongly recommends that a connect string, rather than NULL, be specified. Usually it will be a net service name from tnsnames.ora.
conntype	Specifies the type of connection to the Oracle Database, DBMS_MGWADM.JDBC_OCI or DBMS_MGWADM.JDBC_THIN. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged
initfile	Specifies a Messaging Gateway initialization file used by this agent. NULL indicates that the default initialization file is used. If a value is specified, it should be the full path name of the file. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.
comment	Optional comments for this agent. NULL if a comment is not desired. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.

---

**Note:** The `max_connections` parameter included in previous versions of this subprogram has been deprecated and is non-operational

---

## Usage Notes

- Default values for these configuration parameters are set when the Messaging Gateway agent is installed.
- Changes to the `max_memory` and `max_threads` parameters take effect the next time the Messaging Gateway agent is active. If the Messaging Gateway agent is currently active, then it must be shut down and restarted for the changes to take effect.
- The `service` parameter is used to set an Oracle Scheduler job class attribute. The job class is used to create a Scheduler job that starts the Messaging Gateway agent. An Oracle administrator must create the database service. If the value is NULL, the job class will belong to an internal service that is mapped to all instances.
- The `max_connections` parameter is being deprecated as of the Oracle RDBMS 11g release. The number of messaging connections used by the Messaging Gateway Agent is based on the value of the `max_threads` parameter.
- The `username`, `password`, and `database` parameters specify connection information used by the Messaging Gateway agent for connections to the Oracle Database. An Oracle administrator should create the user and grant it the role `MGW_AGENT_ROLE`.

## ALTER\_JOB Procedure

This procedure alters the properties of a propagation job.

### Syntax

```
DBMS_MGWADM.ALTER_JOB (
  job_name      IN   VARCHAR2,
  rule          IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  transformation IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  exception_queue IN  VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  poll_interval IN   PLS_INTEGER DEFAULT 0,
  options       IN   SYS.MGW_PROPERTIES DEFAULT NULL,
  comments      IN   VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

### Parameters

**Table 90–22 ALTER\_JOB Procedure Parameters**

Parameter	Description
job_name	Identifies the propagation job
rule	Specifies an optional subscription rule used to dequeue messages from the propagation source. The syntax and interpretation of this parameter depend on the propagation type. A NULL value indicates that no subscription rule is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.
transformation	Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for propagation_type.  A NULL value indicates that no transformation is needed. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.
exception_queue	Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. In cases in which no exception queue is associated with the job, propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the propagation type.  A NULL value indicates that no exception queue is used. If DBMS_MGWADM.NO_CHANGE, the current value is unchanged.
poll_interval	Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available.  Values: NULL, 0, or value > 0: <ul style="list-style-type: none"> <li>■ If zero (default), the current value will not be changed.</li> <li>■ If NULL, the current value will be reset and the Messaging Gateway default polling interval will be used. The default polling interval is 5 seconds and can be overridden by the Messaging Gateway initialization file.</li> </ul>



**Table 90–22 (Cont.) ALTER\_JOB Procedure Parameters**

Parameter	Description
options	Optional job properties. If NULL, no options will be changed. If not NULL, then the properties specified in this list are combined with the current optional properties to form a new set of job options.
comments	An optional comment for this agent, or NULL if one is not desired. If DBMS_MGWADM.NO_CHANGE, the current value will not be changed.

## Usage Notes

- If the non-Oracle messaging link being accessed for the propagation job uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.
- The subscriber rule cannot be altered when propagating from a JMS source. Instead, the propagation job must be dropped and re-created with the new rule. For JMS, changing the message selector on a durable subscription is equivalent to deleting and re-creating the subscription.
- Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be DBMS\_MGWADM.NO\_CHANGE (the default value).
- The options parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner; add a new property, replace an existing property, remove an existing property or remove all properties.

---



---

### See Also:

- [SYS.MGW\\_PROPERTY Object Type](#) on page 90-12 for more information about the options parameter
  - [OUTBOUND\\_PROPAGATION Jobs](#) on page 90-35 for outbound propagation parameter interpretation
  - [INBOUND\\_PROPAGATION Jobs](#) on page 90-36 for inbound propagation parameter interpretation
- 
-

## ALTER\_MSGSYSTEM\_LINK Procedure for TIB/Rendezvous

Alters the properties of a TIB/Rendezvous messaging system link.

### Syntax

```
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (
  linkname      IN  VARCHAR2,
  properties    IN  SYS.MGW_TIBRV_PROPERTIES,
  options       IN  SYS.MGW_PROPERTIES DEFAULT NULL,
  comment      IN  VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE );
```

### Parameters

**Table 90–23 ALTER\_MSGSYSTEM\_LINK Procedure Parameters for TIB/Rendezvous**

Parameters	Description
linkname	The messaging system link name
properties	Basic properties for a TIB/Rendezvous messaging system link. If NULL, then no link properties will be changed.
options	Optional link properties. If NULL, then no options will be changed. If not NULL, then the properties specified in this list are combined with the current options properties to form a new set of link options.
comment	A user-specified description, or NULL if one is not desired. If DBMS_MGWADM.NO_CHANGE, then the current value will not be changed.

### Usage Notes

To retain an existing value for a messaging link property with a VARCHAR2 data type, specify DBMS\_MGWADM.NO\_CHANGE for that particular property. To preserve an existing value for a property of another data type, specify NULL for that property.

The options parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

**See Also:** [SYS.MGW\\_PROPERTIES Object Type](#) on page 90-10

Some properties cannot be modified, and this procedure will fail if an attempt is made to alter such a property. For properties and options that can be changed, a few are dynamic, and Messaging Gateway uses the new values immediately. Others require the Messaging Gateway agent to be shut down and restarted before they take effect.

**See Also:** "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide* for more information about the messaging system properties and options

## ALTER\_MSGSYSTEM\_LINK Procedure for WebSphere MQ

This procedure alters the properties of a WebSphere MQ messaging system link.

### Syntax

```
DBMS_MGWADM.ALTER_MSGSYSTEM_LINK (
  linkname IN VARCHAR2,
  properties IN SYS.MGW_MQSERIES_PROPERTIES,
  options IN SYS.MGW_PROPERTIES DEFAULT NULL,
  comment IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE);
```

### Parameters

**Table 90–24 ALTER\_MSGSYSTEM\_LINK Procedure Parameters for WebSphere MQ**

Parameters	Description
linkname	The messaging system link name
properties	Basic properties for a WebSphere MQ messaging system link. If it is NULL, then no link properties are changed.
options	Optional link properties. NULL if no options are changed. If not NULL, then the properties specified in this list are combined with the current options properties to form a new set of link options.
comment	An optional description or NULL if not desired. If DBMS_MGWADM.NO_CHANGE is specified, then the current value is not changed.

### Usage Notes

To retain an existing value for a messaging link property with a VARCHAR2 data type, specify DBMS\_MGWADM.NO\_CHANGE for that particular property. To preserve an existing value for a property of another data type, specify NULL for that property.

The options parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

**See Also:** [SYS.MGW\\_PROPERTIES Object Type](#) on page 90-10

Some properties cannot be modified, and this procedure will fail if an attempt is made to alter such a property. For properties and options that can be changed, a few are dynamic, and Messaging Gateway uses the new values immediately. Others require the Messaging Gateway agent to be shut down and restarted before they take effect.

**See Also:** "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide* for more information about the messaging system properties and options

## ALTER\_PROPAGATION\_SCHEDULE Procedure

This procedure alters a propagation schedule.

---



---

**Note:** This subprogram has been deprecated as a result of improved technology (see [ALTER\\_JOB Procedure](#) on page 90-22), and is retained only for reasons of backward compatibility.

---



---

### Syntax

```
DBMS_MGWADM.ALTER_PROPAGATION_SCHEDULE (
    schedule_id IN VARCHAR2,
    duration     IN NUMBER DEFAULT NULL,
    next_time    IN VARCHAR2 DEFAULT NULL,
    latency      IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 90–25 ALTER\_PROPAGATION\_SCHEDULE Procedure Parameters**

Parameter	Description
schedule_id	Identifies the propagation schedule to be altered
duration	Reserved for future use
next_time	Reserved for future use
latency	Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available in the source queue, then the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available.  Values: NULL or value > 0. If latency is NULL, then the Messaging Gateway agent default polling interval will be used. The default polling interval is 5 seconds, but it can be overridden by the Messaging Gateway initialization file.

### Usage Notes

This procedure always overwrites the existing value for each parameter. If a given parameter is not specified, then the existing values are overwritten with the default value.

## ALTER\_SUBSCRIBER Procedure

This procedure alters the parameters of a subscriber used to consume messages from a source queue for propagation to a destination.

---

**Note:** This subprogram has been deprecated as a result of improved technology (see [ALTER\\_JOB Procedure](#) on page 90-22), and is retained only for reasons of backward compatibility.

---

### Syntax

```
DBMS_MGWADM.ALTER_SUBSCRIBER (
  subscriber_id  IN VARCHAR2,
  rule           IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  transformation IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  exception_queue IN VARCHAR2 DEFAULT DBMS_MGWADM.NO_CHANGE,
  options       IN SYS.MGW_PROPERTIES DEFAULT NULL );
```

### Parameters

**Table 90–26 ALTER\_SUBSCRIBER Procedure Parameters**

Parameter	Description
subscriber_id	Identifies the subscriber to be altered
rule	Specifies an optional subscription rule used by the subscriber to dequeue messages from the source queue. The syntax and interpretation of this parameter depend on the subscriber propagation type.  A NULL value indicates that no subscription rule is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.
transformation	Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the subscriber propagation type.  A NULL value indicates that no transformation is needed. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.
exception_queue	Specifies a queue used for exception message logging. This queue must be on the same messaging system as the propagation source. If no exception queue is associated with the subscriber, then propagation stops if a problem occurs. The syntax and interpretation of this parameter depend on the subscriber propagation type.  A NULL value indicates that no exception queue is used. If DBMS_MGWADM.NO_CHANGE, then the current value is unchanged.  The source queue and exception queue cannot be the same queue.
options	Optional subscriber properties. If NULL, then no options will be changed. If not NULL, then the properties specified in this list are combined with the current optional properties to form a new set of subscriber options.

## Usage Notes

If the non-Oracle messaging link being accessed for the subscriber uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.

When propagating from a JMS source, the subscriber rule cannot be altered. Instead, the subscriber must be removed and added with the new rule. For JMS, changing the message selector on a durable subscription is equivalent to deleting and re-creating the subscription.

Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be `DBMS_MGWADM.NO_CHANGE` (the default value).

The `options` parameter specifies a set of properties used to alter the current optional properties. Each property affects the current property list in a particular manner: add a new property, replace an existing property, remove an existing property, or remove all properties.

### See Also:

- [SYS.MGW\\_PROPERTIES Object Type](#) on page 90-10 for more information on the options parameter
- "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide* for more information about WebSphere MQ subscriber options
- "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide* for more information about TIB/Rendezvous subscriber options
- ["OUTBOUND\\_PROPAGATION Subscribers"](#) on page 90-17 for outbound propagation parameter interpretation
- ["INBOUND\\_PROPAGATION Subscribers"](#) on page 90-18 for inbound propagation parameter interpretation

## CLEANUP\_GATEWAY Procedures

This procedure cleans up Messaging Gateway. The procedure performs cleanup or recovery actions that may be needed when Messaging Gateway is left in some abnormal or unexpected condition. The `MGW_GATEWAY` view lists Messaging Gateway status and configuration information that pertains to the cleanup actions.

### Syntax

```
DBMS_MGWADM.CLEANUP_GATEWAY (
  action      IN   BINARY_INTEGER,
  sarg        IN   VARCHAR2 DEFAULT NULL);
```

```
DBMS_MGWADM.CLEANUP_GATEWAY (
  agent_name  IN   VARCHAR2,
  action      IN   BINARY_INTEGER,
  sarg        IN   VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 90–27 CLEANUP\_GATEWAY Procedure Parameters**

Parameter	Description
<code>action</code>	The cleanup action to be performed. Values: <ul style="list-style-type: none"> <li>▪ <code>DBMS_MGWADM.CLEAN_STARTUP_STATE</code> for Messaging Gateway start up state recovery</li> <li>▪ <code>DBMS_MGWADM.CLEAN_LOG_QUEUES</code> for log queue cleanup</li> <li>▪ <code>DBMS_MGWADM.RESET_SUB_MISSING_LOG_REC</code> for propagation job recovery due to missing log record</li> <li>▪ <code>DBMS_MGWADM.RESET_SUB_MISSING_MESSAGE</code> for propagation job recovery due to missing message</li> </ul>
<code>sarg</code>	Optional argument whose meaning depends on the value specified for <code>action</code> . This should be <code>NULL</code> if it is not used for the specified action.
<code>agent_name</code>	Identifies the Messaging Gateway agent. <code>DBMS_MGWADM.DEFAULT_AGENT</code> specifies the default agent.

### Usage Notes

#### CLEAN\_STARTUP\_STATE

`sarg` is not used and must be `NULL`.

The `CLEAN_STARTUP_STATE` action recovers Messaging Gateway to a known state when the Messaging Gateway agent has crashed or some other abnormal event occurs, and Messaging Gateway cannot be restarted. This should be done only when the Messaging Gateway agent has been started but appears to have crashed or has been nonresponsive for an extended period of time.

The `CLEAN_STARTUP_STATE` action may be needed when the `MGW_GATEWAY` view shows that the `AGENT_STATUS` value is something other than `NOT_STARTED` or `START_SCHEDULED`, and the `AGENT_PING` value is `UNREACHABLE` for an extended period of time.

If the `AGENT_STATUS` value is `BROKEN`, then the Messaging Gateway agent cannot be started until the problem has been resolved and the `CLEAN_STARTUP_STATE` action used to reset the agent status. A `BROKEN` status can indicate that the Messaging Gateway start job detected a Messaging Gateway agent already running. This condition that should never occur under normal use.

Cleanup tasks include:

- Removing the Scheduler job used to start the external Messaging Gateway agent process.
- Setting certain configuration information to a known state. For example, setting the agent status to `NOT_STARTED`.

Execution of this command fails if:

- The agent status is `NOT_STARTED` or `START_SCHEDULED`.
- No shutdown attempt has been made prior to calling this procedure, except if the agent status is `STARTING`.
- The Messaging Gateway agent is successfully contacted.

The assumption is that the agent is active, and this procedure fails. If the agent does not respond after several attempts have been made, then the cleanup tasks are performed. This procedure takes at least several seconds and possibly up to one minute. This is expected behavior under conditions where this particular cleanup action is appropriate and necessary.

---

---

**Note:** Terminate any Messaging Gateway agent process that may still be running after a `CLEAN_STARTUP_STATE` action has been successfully performed. This should be done before calling `DBMS_MGWADM.STARTUP` to start Messaging Gateway. The process is usually named `extprocmgwextproc`.

---

---

## **CLEAN\_LOG\_QUEUES**

`sarg` is not used and must be `NULL`.

The Messaging Gateway agent will clean log queues for all configured messaging system links. The agent will temporarily stop all propagation activity and then remove all obsolete and bad log records from the log queues for all links. The procedure will fail if the Messaging Gateway agent is not running.

This cleanup action is automatically performed each time the Messaging Gateway agent is started.

---

---

**Note:** The `CLEAN_LOG_QUEUES` action is performed only on agent startup. If this procedure is called when the agent is running, then the Messaging Gateway agent ignores it.

---

---

## **RESET\_SUB\_MISSING\_LOG\_REC**

`sarg` specifies a Messaging Gateway job name (or subscriber ID) to be reset. It must not be `NULL`.

The Messaging Gateway agent recovers a Messaging Gateway propagation job that has failed due to a missing log record. The agent will reset the source and destination log records. The procedure will fail if the Messaging Gateway agent is not running.



---

---

**Caution:** If the messages in the source queue had already been propagated to the destination queue, then this action may result in duplicate messages.

---

---

### **RESET\_SUB\_MISSING\_MESSAGE**

`sarg` specifies a Messaging Gateway job name (or subscriber ID) to be reset. It must not be `NULL`.

The Messaging Gateway agent recovers a Messaging Gateway propagation job that has failed due to a missing persistent source message. The agent will treat the message as a non-persistent message and continue processing that propagation job. The procedure will fail if the Messaging Gateway agent is not running.

## CREATE\_AGENT Procedure

This procedure creates a Messaging Gateway agent that will be used to process propagation jobs.

### Syntax

```
DBMS_MGWADM.CREATE_AGENT (
  agent_name      IN   VARCHAR2,
  username        IN   VARCHAR2 DEFAULT NULL,
  password        IN   VARCHAR2 DEFAULT NULL,
  database        IN   VARCHAR2 DEFAULT NULL,
  conntype       IN   VARCHAR2 DEFAULT DBMS_MGWADM.JDBC_OCI,
  max_memory     IN   PLS_INTEGER DEFAULT 64,
  max_threads    IN   PLS_INTEGER DEFAULT 1,
  service        IN   VARCHAR2 DEFAULT NULL,
  initfile       IN   VARCHAR2 DEFAULT NULL,
  comment        IN   VARCHAR2  DEFAULT NULL );
```

### Parameters

**Table 90–28 CREATE\_AGENT Procedure Parameters**

Parameter	Description
agent_name	A name used to identify the agent
username	Specifies the username used for connections to the Oracle Database
password	Specifies the password used for connections to the Oracle Database. A password must be specified if a username is specified.
database	Specifies the database connect string used for connections to the Oracle Database. NULL indicates that a local connection should be used. A value can be specified only if username is specified. Oracle strongly recommends that a connect string, rather than NULL be specified. Usually it will be a net service name from tnsnames.ora.
conntype	Specifies the type of connection to the Oracle Database. Values: DBMS_MGWADM.JDBC_OCI, DBMS_MGWADM.JDBC_THIN
max_memory	Specifies the maximum heap size, in MB, used by the Messaging Gateway agent
max_threads	Specifies the number of messaging threads that the Messaging Gateway agent creates. This determines the number of propagation jobs that the agent can concurrently process.
service	Specifies the database service that the Oracle Scheduler job class used by this agent will have affinity to. In an Oracle RAC environment, this means that the Messaging Gateway agent will only run on those database instances that are assigned to the service. If NULL, then the job class will belong to the default service which is mapped to every instance.
initfile	Specifies a Messaging Gateway initialization file used by this agent. NULL indicates that the default initialization file is used. If a value is specified, it should be the full path name of the file.
comment	An optional comment for this agent. NULL if one is not desired.

## Usage Notes

- The Messaging Gateway automatically configures a default agent when Messaging Gateway is installed. The name of the default agent is `DEFAULT_AGENT`. This procedure can be used to create additional agents.
- The `username`, `password`, and `database` parameters specify connection information used by the Messaging Gateway agent for connections to the Oracle Database. An Oracle administrator should create the database user and grant it the role `MGW_AGENT_ROLE`. It is not mandatory that the connection information be specified when this procedure is called but it must be set before the agent can be started.
- The `service` parameter is used to create an Oracle Scheduler job class. The job class is used to create a Scheduler job that starts the Messaging Gateway agent. An Oracle administrator must create the database service. If the value is `NULL`, the job class will belong to an internal service that is mapped to all instances.

## CREATE\_JOB Procedure

This procedure creates a job used to propagate message from a source to a destination.

### Syntax

```
DBMS_MGWADM.CREATE_JOB (
  job_name          IN  VARCHAR2,
  propagation_type IN  PLS_INTEGER,
  source            IN  VARCHAR2,
  destination       IN  VARCHAR2,
  rule              IN  VARCHAR2 DEFAULT NULL,
  transformation     IN  VARCHAR2 DEFAULT NULL,
  exception_queue   IN  VARCHAR2 DEFAULT NULL,
  poll_interval     IN  PLS_INTEGER DEFAULT NULL,
  options           IN  SYS.MGW_PROPERTIES DEFAULT NULL,
  enabled           IN  BOOLEAN DEFAULT TRUE,
  comments          IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 90–29 CREATE\_JOB Procedure Parameters**

Parameter	Description
job_name	A user defined name to identify the propagation job
propagation_type	Specifies the type of message propagation. <ul style="list-style-type: none"> <li>■ DBMS_MGWADM.OUTBOUND_PROPAGATION for Oracle Streams AQ to non-Oracle propagation.</li> <li>■ DBMS_MGWADM.INBOUND_PROPAGATION for non-Oracle to Oracle Streams AQ propagation.</li> </ul>
source	Specifies the source queue whose messages are to be propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
destination	Specifies the destination queue to which messages are propagated. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
rule	Specifies an optional subscription rule used to dequeue messages from the source queue. This should be NULL if no rule is needed. The syntax and interpretation of this parameter depend on the value specified for propagation_type.
transformation	Specifies the transformation needed to convert between the Oracle Streams AQ payload and an ADT defined by Messaging Gateway. The type of transformation needed depends on the value specified for propagation_type.  If no transformation is specified the Oracle Streams AQ payload type must be supported by Messaging Gateway.
exception_queue	Specifies a queue used for exception message logging purposes. This queue must be on the same messaging system as the propagation source. If NULL, an exception queue will not be used and propagation will stop if a problem occurs. The syntax and interpretation of this parameter depend on the value specified for propagation_type.  The source queue and exception queue cannot be the same queue.

**Table 90–29 (Cont.) CREATE\_JOB Procedure Parameters**

Parameter	Description
<code>poll_interval</code>	Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available.  Values: NULL or value > 0. If NULL, then the Messaging Gateway default polling interval will be used. The default polling interval is 5 seconds and can be overridden by the Messaging Gateway initialization file.
<code>options</code>	Optional job properties, NULL if there are none. Typically these are lesser used configuration properties supported by the messaging system.
<code>enabled</code>	Specifies whether this propagation job is enabled after creation. Values: TRUE, FALSE. <ul style="list-style-type: none"> <li>■ If TRUE (default), the job will be enabled after it is created.</li> <li>■ If FALSE, the job will be disabled after it is created. A propagation job must be enabled and the Messaging Gateway agent running before messages can be propagated.</li> </ul>
<code>comments</code>	An optional comment for this job. NULL if one is not desired.

## Usage Notes

- The job must be enabled and Messaging Gateway agent started in order for messages to be propagated.
- If the non-Oracle messaging link being accessed for the propagation job uses a JMS interface, then the Messaging Gateway agent will use the Oracle JMS interface to access the Oracle Streams AQ queues. Otherwise the native Oracle Streams AQ interface will be used. Parameters are interpreted differently when the Messaging Gateway agent uses Oracle JMS for JMS connections.
- Transformations are not currently supported if the Oracle JMS interface is used for propagation. The transformation parameter must be NULL.

### OUTBOUND\_PROPAGATION Jobs

The parameters for an outbound propagation job are interpreted as follows:

- `source` specifies the local Oracle Streams AQ queue that is the propagation source. This must have syntax of `schema.queue`. This can be either a multiple consumer queue or a single consumer queue.
- `destination` specifies the non-Oracle queue to which messages are propagated. This must have syntax of `registered_queue@message_link`.
- `rule` specifies an optional Oracle Streams AQ subscriber rule if the native Oracle Stream AQ interface is used, or a JMS selector if the Oracle JMS interface is used. If NULL, then no rule or selector is used. This parameter must be NULL if the native Oracle Stream AQ interface is used and the propagation source is a single consumer queue.
- `transformation` specifies the transformation used to convert the Oracle Streams AQ payload to an ADT defined by Messaging Gateway. The full transformation name (`schema.name`) should be used if one is specified.

Messaging Gateway propagation dequeues messages from the Oracle Streams AQ queue using the transformation to convert the Oracle Streams AQ payload to a known ADT defined by Messaging Gateway. The message is then enqueued in the non-Oracle messaging system based on the Messaging Gateway ADT.

- `exception_queue` specifies the name of a local Oracle Streams AQ queue to which messages are moved if an exception occurs. The syntax must be `schema.queue`.

If the native Oracle Streams AQ interface is used and the source is a multiple consumer queue, then a subscriber will be added to the Oracle Streams AQ queue when this procedure is called, whether or not the Messaging Gateway agent is running. The local subscriber will be of the form `sys.aq$_agent('MGW_job_name', NULL, NULL)`.

If the Oracle JMS interface is used, then the Messaging Gateway agent will create a JMS durable subscriber with the name of `MGW_job_name`. If the agent is not running when this procedure is called, then the durable subscriber will be created the next time the agent starts.

The exception queue has the following conditions:

- The user is responsible for creating the Oracle Streams AQ queue to be used as the exception queue.
- The payload type of the source queue and exception queue must match.
- The exception queue must be created as a queue type of `DBMS_AQADM.NORMAL_QUEUE`. Enqueue restrictions prevent Messaging Gateway from using an Oracle Streams AQ queue of type `DBMS_AQADM.EXCEPTION_QUEUE` as a Messaging Gateway exception queue.

### INBOUND\_PROPAGATION Jobs

The parameters for an inbound propagation job are interpreted as follows:

- `source` specifies the non-Oracle queue that is the propagation source. The syntax must be `registered_queue@message_link`.
- `destination` specifies the local Oracle Streams AQ queue to which messages are propagated. The syntax must be `schema.queue`.
- `rule` specifies an optional subscriber rule that is valid for the non-Oracle messaging system. This should be `NULL` if no rule is needed.
- `transformation` specifies the transformation used to convert an ADT defined by Messaging Gateway to the Oracle Streams AQ payload type. The full transformation name (`schema.name`) should be used if one is specified

Messaging Gateway propagation dequeues messages from the non-Oracle messaging system and converts the message body to a known ADT defined by Messaging Gateway. The transformation is used to convert the Messaging Gateway ADT to an Oracle Streams AQ payload type when the message is enqueued to the Oracle Streams AQ queue.

- `exception_queue` specifies the name of a registered non-Oracle queue to which messages are moved if an exception occurs. The syntax must be `registered_queue@message_link`.

Whether or not a subscriber is needed for the source queue depends on the requirements of the non-Oracle messaging system. If a durable subscriber is necessary, then the Messaging Gateway agent will create it. If the agent is not

running when this procedure is called, then the subscriber will be created on the non-Oracle messaging system the next time the agent starts.

The exception queue has the following conditions:

- The exception queue must be a registered non-Oracle queue.
- The source queue and exception queue must use the same messaging system link.

## CREATE\_MSGSYSTEM\_LINK Procedures for TIB/Rendezvous

Creates a link to a TIB/Rendezvous messaging system.

### Syntax

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK (
  linkname      IN  VARCHAR2,
  properties    IN  SYS.MGW_TIBRV_PROPERTIES,
  options       IN  SYS.MGW_PROPERTIES DEFAULT NULL,
  comment       IN  VARCHAR2 DEFAULT NULL );
```

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK (
  linkname      IN  VARCHAR2,
  agent_name    IN  VARCHAR2,
  properties    IN  SYS.MGW_TIBRV_PROPERTIES,
  options       IN  SYS.MGW_PROPERTIES DEFAULT NULL,
  comment       IN  VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 90–30** CREATE\_MSGSYSTEM\_LINK Procedure Parameters for TIB/Rendezvous

Parameter	Description
linkname	A user-defined name to identify this messaging system link
properties	Basic properties of a TIB/Rendezvous messaging system link.
options	Optional link properties. NULL if there are none. These are less frequently used configuration properties supported by the messaging system
comment	A user-specified description. NULL if one is not desired.
agent_name	Specifies the Messaging Gateway agent that will be used to process all propagation jobs associated with this link. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.

### Usage Notes

The Messaging Gateway default agent will process the propagation jobs associated with this link if an agent name is not specified.

**See Also:** "TIB/Rendezvous System Properties" in *Oracle Streams Advanced Queuing User's Guide* for more information about the messaging system properties and options



## CREATE\_MSGSYSTEM\_LINK Procedures for WebSphere MQ

This procedure creates a messaging system link to a WebSphere MQ messaging system.

### Syntax

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
  linkname      IN VARCHAR2,
  properties    IN SYS.MGW_MQSERIES_PROPERTIES,
  options       IN SYS.MGW_PROPERTIES DEFAULT NULL,
  comment       IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_MGWADM.CREATE_MSGSYSTEM_LINK(
  linkname      IN VARCHAR2,
  agent_name    IN VARCHAR2,
  properties    IN SYS.MGW_MQSERIES_PROPERTIES,
  options       IN SYS.MGW_PROPERTIES DEFAULT NULL,
  comment       IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 90–31** CREATE\_MSGSYSTEM\_LINK Procedure Parameters for WebSphere MQ

Parameter	Description
linkname	A user-defined name to identify the messaging system link
properties	Basic properties of a WebSphere MQ messaging system link
options	Optional link properties. NULL if there are none. These are less frequently used configuration properties supported by the messaging system.
comment	A user-specified description. NULL if one is not desired
agent_name	Specifies the Messaging Gateway agent that will be used to process all propagation jobs associated with this link. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.

### Usage Notes

The Messaging Gateway default agent will process the propagation jobs associated with this link if an agent name is not specified.

**See Also:** "WebSphere MQ System Properties" in *Oracle Streams Advanced Queuing User's Guide* for more information about the messaging system properties and options

## DB\_CONNECT\_INFO Procedure

This procedure configures connection information used by the Messaging Gateway default agent for connections to Oracle Database.

---



---

**Note:** This subprogram has been deprecated as a result of improved technology (see [ALTER\\_AGENT Procedures](#) on page 90-20), and is retained only for reasons of backward compatibility.

---



---

### Syntax

```
DBMS_MGWADM.DB_CONNECT_INFO (
  username      IN VARCHAR2,
  password      IN VARCHAR2,
  database      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 90–32** *DB\_CONNECT\_INFO Procedure Parameters*

Parameter	Description
username	The username used for connections to Oracle Database. NULL is not allowed
password	The password used for connections to Oracle Database. NULL is not allowed
database	The database connect string used by the Messaging Gateway agent. NULL indicates that a local connection should be used.  Oracle strongly recommends that a not NULL value be specified. Usually it will be a net service name from <code>tnsnames.ora</code> .

### Usage Notes

The Messaging Gateway agent connects to Oracle Database as the user configured by this procedure. An Oracle administrator should create the user, grant it the role `MGW_AGENT_ROLE`, and then call this procedure to configure Messaging Gateway. Role `MGW_AGENT_ROLE` is used to grant this user special privileges needed to access Messaging Gateway configuration information stored in the database, enqueue or dequeue messages to and from Oracle Streams AQ queues, and perform certain Oracle Streams AQ administration tasks.

## DISABLE\_JOB Procedure

This procedure disables a propagation job.

### Syntax

```
DBMS_MGWADM.DISABLE_JOB (  
    job_name IN VARCHAR2);
```

### Parameters

**Table 90–33** *DISABLE\_JOB Procedure Parameters*

Parameter	Description
job_name	Identifies the propagation job

## DISABLE\_PROPAGATION\_SCHEDULE Procedure

This procedure disables a propagation schedule.

---

---

**Note:** This subprogram has been deprecated as a result of improved technology (see [DISABLE\\_JOB Procedure](#) on page 90-41), and is retained only for reasons of backward compatibility.

---

---

### Syntax

```
DBMS_MGWADM.DISABLE_PROPAGATION_SCHEDULE (  
    schedule_id IN VARCHAR2);
```

### Parameters

**Table 90–34** *DISABLE\_PROPAGATION\_SCHEDULE Procedure Parameters*

Parameter	Description
schedule_id	Identifies the propagation schedule to be disabled

## ENABLE\_JOB Procedure

This procedure enables a propagation job.

### Syntax

```
DBMS_MGWADM.ENABLE_JOB (  
    job_name IN VARCHAR2 );
```

### Parameters

**Table 90–35** *ENABLE\_JOB Procedure Parameters*

Parameter	Description
job_name	Identifies the propagation job

## ENABLE\_PROPAGATION\_SCHEDULE Procedure

This procedure enables a propagation schedule.

---

---

**Note:** This subprogram has been deprecated as a result of improved technology (see [ENABLE\\_JOB Procedure](#) on page 90-43), and is retained only for reasons of backward compatibility.

---

---

### Syntax

```
DBMS_MGWADM.ENABLE_PROPAGATION_SCHEDULE (  
    schedule_id IN VARCHAR2 );
```

### Parameters

**Table 90–36** *ENABLE\_PROPAGATION\_SCHEDULE Procedure Parameters*

Parameter	Description
schedule_id	Identifies the propagation schedule to be enabled

## REGISTER\_FOREIGN\_QUEUE Procedure

This procedure registers a non-Oracle queue entity in Messaging Gateway.

### Syntax

```
DBMS_MGWADM.REGISTER_FOREIGN_QUEUE (
  name           IN VARCHAR2,
  linkname       IN VARCHAR2,
  provider_queue IN VARCHAR2 DEFAULT NULL,
  domain         IN INTEGER DEFAULT NULL,
  options        IN SYS.MGW_PROPERTIES DEFAULT NULL,
  comment        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 90–37 REGISTER\_FOREIGN\_QUEUE Procedure Parameters**

Parameters	Description
name	The registered queue name. This name identifies the foreign queue within Messaging Gateway and need not match the name of the queue in the foreign messaging system.
linkname	The link name for the messaging system on which this queue exists
provider_queue	The message provider (native) queue name. If NULL, then the value provided for the name parameter is used as the provider queue name.
domain	The domain type of the queue. NULL means the domain type is automatically determined based on the messaging system of the queue. DBMS_MGWADM.DOMAIN_QUEUE is for a queue (point-to-point model). DBMS_MGWADM.DOMAIN_TOPIC is for a topic (publish-subscribe model).
options	Optional queue properties
comment	A user-specified description. Can be NULL.

### Usage Notes

This procedure does not create the physical queue in the non-Oracle messaging system. The non-Oracle queue must be created using the administration tools for that messaging system.

**See Also:** For more information when registering queues for the WebSphere MQ messaging system or the TIB/Rendezvous messaging system, specifically "Optional Foreign Queue Configuration Properties" in *Oracle Streams Advanced Queuing User's Guide*.

## REMOVE\_AGENT Procedure

This procedure removes a Messaging Gateway agent.

### Syntax

```
DBMS_MGWADM.REMOVE_AGENT (  
    agent_name IN VARCHAR2 );
```

### Parameters

**Table 90–38 REMOVE\_AGENT Procedure Parameters**

Parameters	Description
agent_name	Identifies the Messaging Gateway agent

### Usage Notes

All messaging system links associated with this Messaging Gateway agent must be removed and the agent must be stopped before it can be removed. The Messaging Gateway default agent cannot be removed.



## REMOVE\_JOB Procedure

This procedure removes a propagation job.

### Syntax

```
DBMS_MGWADM.REMOVE_JOB (
  job_name  IN  VARCHAR2,
  force     IN  PLS_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE);
```

### Parameters

**Table 90–39 REMOVE\_JOB Procedure Parameters**

Parameters	Description
job_name	Identifies the propagation job
force	<p>Specifies whether the procedure should succeed even if Messaging Gateway is not able to perform all cleanup actions pertaining to this propagation job.</p> <p>Values: DBMS_MGWADM.NO_FORCE, DBMS_MGWADM.FORCE</p> <ul style="list-style-type: none"> <li>■ NO_FORCE (default) means the job is not removed if Messaging Gateway is unable to clean up successfully</li> <li>■ FORCE means the job is removed even though all cleanup actions may not be done</li> </ul>

### Usage Notes

- The Messaging Gateway agent uses various resources of the Oracle Database and the non-Oracle messaging system for its propagation work. These resources need to be released when the job is removed. For example, Messaging Gateway may create a durable subscriber on the source queue that should be removed when the job is removed. Therefore, this procedure should normally be called when the Messaging Gateway agent is running and able to access the non-Oracle messaging system associated with this job.
- For outbound propagation, a local subscriber is removed from the Oracle Streams AQ queue when the propagation source is a multiple consumer queue.

## REMOVE\_MSGSYSTEM\_LINK Procedure

This procedure removes a messaging system link for a non-Oracle messaging system.

### Syntax

```
DBMS_MGWADM.REMOVE_MSGSYSTEM_LINK(  
    linkname IN VARCHAR2);
```

### Parameters

**Table 90–40 REMOVE\_MSGSYSTEM\_LINK Procedure Parameters**

Parameters	Description
linkname	The messaging system link name

### Usage Notes

All registered queues associated with this link must be removed before the messaging system link can be removed. This procedure fails if there is a registered foreign (non-Oracle) queue that references this link.

## REMOVE\_OPTION Procedure

This procedure removes a Messaging Gateway configuration option. It can be used to remove an agent option, a messaging link option, or a propagation job option.

### Syntax

```
DBMS_MGWADM.REMOVE_OPTION (
    target_type IN PLS_INTEGER,
    target_name IN VARCHAR2,
    option_name IN VARCHAR2);
```

### Parameters

**Table 90–41 REMOVE\_OPTION Procedure Parameters**

Parameter	Description
target_type	Specifies the target type of the Messaging Gateway entity: <ul style="list-style-type: none"> <li>■ DBMS_MGWADM.AGENT_JAVA_PROP to remove a Java System property for a Messaging Gateway agent</li> <li>■ DBMS_MGWADM.MSGLINK_OPTION to remove a messaging link option</li> <li>■ DBMS_MGWADM.JOB_OPTION to remove a propagation job option</li> </ul>
target_name	Name or identifier of the target. The value for this parameter depends on the value specified for target_type parameter. This must not be NULL.
option_name	Option name. This must not be NULL.

**See Also:** [Table 90–10, "DBMS\\_MGWADM Constants—target\\_type Argument of SET\\_OPTION and REMOVE\\_OPTION Procedures"](#) on page 90-6 regarding options for the option\_type parameter

### Usage Notes

#### DBMS\_MGWADM.AGENT\_JAVA\_PROP Target

The procedure removes an agent option used to set a Java System property when the Messaging Gateway agent is started. The agent must be restarted for the change to take effect.

The parameters are interpreted as follows:

- target\_name specifies the name of the Messaging Gateway agent. DBMS\_MGWADM.DEFAULT\_AGENT can be used for the default agent.
- option\_name specifies the Java System property
- encrypted can be either TRUE or FALSE

#### DBMS\_MGWADM.MSGLINK\_OPTION Target

The procedure removes a single option for a Messaging Gateway messaging system link. This is equivalent to calling DBMS\_MGWADM.ALTER\_MSGSYSTEM\_LINK and using the options parameter to remove an option.

The parameters are interpreted as follows:

- `target_name` specifies the name of the message system link
- `option_name` specifies the option to set
- `encrypted` must be `FALSE`

**DBMS\_MGWADM.JOB\_OPTION Target**

The procedure removes a single option for a Messaging Gateway propagation job. This is equivalent to calling `DBMS_MGWADM.ALTER_JOB` and using the `options` parameter to remove an option.

The parameters are interpreted as follows:

- `target_name` specifies the name of the propagation job
- `option_name` specifies the option to set
- `encrypted` must be `FALSE`

## REMOVE\_SUBSCRIBER Procedure

This procedure removes a subscriber used to consume messages from a source queue for propagation to a destination.

---

**Note:** This subprogram has been deprecated as a result of improved technology (see [REMOVE\\_JOB Procedure](#) on page 90-47), and is retained only for reasons of backward compatibility.

---

### Syntax

```
DBMS_MGWADM.REMOVE_SUBSCRIBER (
  subscriber_id IN VARCHAR2,
  force         IN BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE );
```

### Parameters

**Table 90–42 REMOVE\_SUBSCRIBER Procedure Parameters**

Parameter	Description
subscriber_id	Identifies the subscriber to be removed
force	<p>Specifies whether this procedure should succeed even if Messaging Gateway is not able to perform all cleanup actions pertaining to this subscriber.</p> <p>Values: DBMS_MGWADM.NO_FORCE, DBMS_MGWADM.FORCE</p> <ul style="list-style-type: none"> <li>■ NO_FORCE means the subscriber is not removed if Messaging Gateway is unable to clean up successfully (default)</li> <li>■ FORCE means the subscriber is removed even though all cleanup actions may not be done</li> </ul>

### Usage Notes

- The Messaging Gateway agent uses various resources of Oracle Database and the non-Oracle messaging system for its propagation work. These resources are typically associated with each subscriber and need to be released when the subscriber is no longer needed. Therefore, this procedure should only be called when the Messaging Gateway agent is running and able to access the non-Oracle messaging system associated with this subscriber.
- For outbound propagation, a local subscriber is removed from the Oracle Streams AQ queue.

## RESET\_JOB Procedure

This procedure resets the propagation error state for a propagation job.

### Syntax

```
DBMS_MGWADM.RESET_JOB (  
    job_name IN VARCHAR2);
```

### Parameters

**Table 90–43** *RESET\_JOB Procedure Parameters*

Parameter	Description
job_name	Identifies the propagation job

### Usage Notes

This procedure can be used to reset a propagation job that has been set to a failed state and propagation activities have been stopped. The administrator should correct the problem and then call this procedure to allow the agent to retry the propagation job. The `STATUS` field of the `MGW_JOBS` view indicates the job status.

## RESET\_SUBSCRIBER Procedure

This procedure resets the propagation error state for a subscriber.

---

---

**Note:** This subprogram has been deprecated as a result of improved technology (see [RESET\\_JOB Procedure](#) on page 90-52), and is retained only for reasons of backward compatibility.

---

---

### Syntax

```
DBMS_MGWADM.RESET_SUBSCRIBER (  
    subscriber_id IN VARCHAR2 );
```

### Parameters

**Table 90–44** *RESET\_SUBSCRIBER Procedure Parameters*

Parameter	Description
subscriber_id	Identifies the subscriber

## SCHEDULE\_PROPAGATION Procedure

This procedure schedules message propagation from a source to a destination. The schedule must be enabled and Messaging Gateway started in order for messages to be propagated.

---



---

**Note:** This subprogram has been deprecated as a result of improved technology (see [CREATE\\_JOB Procedure](#) on page 90-34), and is retained only for reasons of backward compatibility.

---



---

### Syntax

```
DBMS_MGWADM.SCHEDULE_PROPAGATION (
  schedule_id      IN VARCHAR2,
  propagation_type IN BINARY_INTEGER,
  source           IN VARCHAR2,
  destination      IN VARCHAR2,
  start_time       IN DATE DEFAULT SYSDATE,
  duration         IN NUMBER DEFAULT NULL,
  next_time        IN VARCHAR2 DEFAULT NULL,
  latency          IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 90–45** *SCHEDULE\_PROPAGATION Procedure Parameters*

Parameter	Description
<code>schedule_id</code>	Specifies a user-defined name that identifies the schedule
<code>propagation_type</code>	Specifies the type of message propagation. <code>DBMS_MGWADM.OUTBOUND_PROPAGATION</code> is for Oracle Streams AQ to non-Oracle propagation. <code>DBMS_MGWADM.INBOUND_PROPAGATION</code> is for non-Oracle to Oracle Streams AQ propagation.
<code>source</code>	Specifies the source queue whose messages are to be propagated. The syntax and interpretation of this parameter depend on the value specified for <code>propagation_type</code> .
<code>destination</code>	Specifies the destination queue to which messages are propagated. The syntax and interpretation of this parameter depend on the value specified for <code>propagation_type</code> .
<code>start_time</code>	Reserved for future use
<code>duration</code>	Reserved for future use
<code>next_time</code>	Reserved for future use
<code>latency</code>	Specifies the polling interval, in seconds, used by the Messaging Gateway agent when checking for messages in the source queue. If no messages are available in the source queue, then the agent will not poll again until the polling interval has passed. Once the agent detects a message it will continue propagating messages as long as any are available.  Values: <code>NULL</code> or value > 0. If <code>latency</code> is <code>NULL</code> , then the Messaging Gateway agent default polling interval will be used. The default polling interval is 5 seconds but it can be overridden by the Messaging Gateway initialization file.



## Usage Notes

For outbound propagation, parameters are interpreted as follows:

- `source` specifies the local Oracle Streams AQ queue from which messages are propagated. This must have a syntax of `schema.queue`.
- `destination` specifies the foreign queue to which messages are propagated. This must have a syntax of `registered_queue@message_link`.

For inbound propagation, parameters are interpreted as follows:

- `source` specifies the foreign queue from which messages are propagated. This must have a syntax of `registered_queue@message_link`.
- `destination` specifies the local Oracle Streams AQ queue to which messages are propagated. This must have a syntax of `schema.queue`.

The schedule is set to an enabled state when it is created.

## SET\_LOG\_LEVEL Procedures

This procedure dynamically alters the Messaging Gateway agent logging level. The Messaging Gateway agent must be running.

### Syntax

```
DBMS_MGWADM.SET_LOG_LEVEL (
    log_level    IN    BINARY_INTEGER);
```

```
DBMS_MGWADM.SET_LOG_LEVEL (
    agent_name   IN    VARCHAR2,
    log_level    IN    BINARY_INTEGER);
```

### Parameters

**Table 90–46** SET\_LOG\_LEVEL Procedure Parameters

Parameter	Description
log_level	Level at which the Messaging Gateway agent logs information. DBMS_MGWADM.BASIC_LOGGING generates the least information while DBMS_MGWADM.TRACE_DEBUG_LOGGING generates the most information.
agent_name	Identifies the Messaging Gateway agent. DBMS_MGWADM.DEFAULT_AGENT specifies the default agent.

**See Also:** [Table 90–3, "DBMS\\_MGWADM Constants—Logging Levels"](#) on page 90-4 for details on the log\_level parameter

## SET\_OPTION Procedure

This procedure sets a Messaging Gateway configuration option. It can be used to set an agent option, a messaging link option, or a propagation job option.

### Syntax

```
DBMS_MGWADM.SET_OPTION (
  target_type  IN  PLS_INTEGER,
  target_name  IN  VARCHAR2,
  option_name  IN  VARCHAR2,
  option_value IN  VARCHAR2,
  encrypted    IN  BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 90–47 SET\_OPTION Procedure Parameters**

Parameter	Description
target_type	Specifies the target type of the Messaging Gateway entity: <ul style="list-style-type: none"> <li>■ DBMS_MGWADM.AGENT_JAVA_PROP to set a Java System property for a Messaging Gateway agent</li> <li>■ DBMS_MGWADM.MSGLINK_OPTION to set a messaging link option</li> <li>■ DBMS_MGWADM.JOB_OPTION to set a propagation job option</li> </ul>
target_name	Name or identifier of the target. The value for this parameter depends on the value specified for target_type parameter. This must not be NULL.
option_name	Option name. This must not be NULL.
option_value	Option value
encrypted	Indicates whether the value should be stored as encrypted: <ul style="list-style-type: none"> <li>■ TRUE if the value should be stored in an encrypted form</li> <li>■ FALSE if the value should be stored in a cleartext form</li> </ul>

**See Also:** [Table 90–10, "DBMS\\_MGWADM Constants—target\\_type Argument of SET\\_OPTION and REMOVE\\_OPTION Procedures"](#) on page 90-6 regarding options for the option\_type parameter

### Usage Notes

#### DBMS\_MGWADM.AGENT\_JAVA\_PROP Target

The procedure will store an agent option used to set a Java System property when the Messaging Gateway agent is started. The agent must be restarted for the change to take effect.

The parameters are interpreted as follows:

- target\_name specifies the name of the Messaging Gateway agent. DBMS\_MGWADM.DEFAULT\_AGENT can be used for the default agent.
- option\_name specifies the Java System property

- `encrypted` can be either `TRUE` or `FALSE`

**DBMS\_MGWADM.MSGLINK\_OPTION Target**

The procedure will set or alter a single option for a Messaging Gateway messaging system link. This is equivalent to calling `DBMS_MGWADM.ALTER_MSGSYSTEM_LINK` and using the options parameter to set an option.

The parameters are interpreted as follows:

- `target_name` specifies the name of the message system link
- `option_name` specifies the option to set
- `encrypted` must be `FALSE`

**DBMS\_MGWADM.JOB\_OPTION Target**

The procedure will set or alter a single option for a Messaging Gateway propagation job. This is equivalent to calling `DBMS_MGWADM.ALTER_JOB` and using the options parameter to set an option.

The parameters are interpreted as follows:

- `target_name` specifies the name of the propagation job
- `option_name` specifies the option to set
- `encrypted` must be `FALSE`

## SHUTDOWN Procedures

This procedure shuts down the Messaging Gateway agent. No propagation activity occurs until Messaging Gateway is restarted.

### Syntax

```
DBMS_MGWADM.SHUTDOWN (
  sdmode          IN BINARY_INTEGER DEFAULT DBMS_MGWADM.SHUTDOWN_NORMAL);
```

```
DBMS_MGWADM.SHUTDOWN (
  agent_name      IN VARCHAR2);
```

### Parameters

**Table 90–48 SHUTDOWN Procedure Parameters**

Parameter	Description
sdmode	The shutdown mode. The only value currently supported is <code>DBMS_MGWADM.SHUTDOWN_NORMAL</code> for normal shutdown. The Messaging Gateway agent may attempt to complete any propagation work currently in progress.
agent_name	Identifies the Messaging Gateway agent. <code>DBMS_MGWADM.DEFAULT_AGENT</code> specifies the default agent.

### Usage Notes

The Messaging Gateway default agent is shut down if no agent name is specified.

## STARTUP Procedures

This procedure starts the Messaging Gateway agent. It must be called before any propagation activity can take place.

### Syntax

```
DBMS_MGWADM.STARTUP (
  instance      IN  BINARY_INTEGER DEFAULT 0,
  force         IN  BINARY_INTEGER DEFAULT DBMS_MGWADM.NO_FORCE);

DBMS_MGWADM.STARTUP (
  agent_name    IN  VARCHAR2);
```

### Parameters

**Table 90–49** *STARTUP Procedure Parameters*

Parameter	Description
instance	Specifies which instance can run the job queue job used to start the Messaging Gateway agent. If this is zero, then the job can be run by any instance.  <b>Caution: This parameter has been deprecated.</b>
force	If this is <code>DBMS_MGWADM.FORCE</code> , then any positive integer is acceptable as the job instance. If this is <code>DBMS_MGWADM.NO_FORCE</code> (the default), then the specified instance must be running; otherwise the routine raises an exception.  <b>Caution: This parameter has been deprecated.</b>
agent_name	Identifies the Messaging Gateway agent. <code>DBMS_MGWADM.DEFAULT_AGENT</code> specifies the default agent.

### Usage Notes

- The Messaging Gateway default agent will be started if an agent name is not specified.
- The `force` and `instance` parameters are no longer used and will be ignored. If the `instance` affinity parameters were being used to start the default agent on a specific instance, the administrator will need to create a database service and then assign that service to the default agent using the `DBMS_MGWADM.ALTER_AGENT` procedure.
- The Messaging Gateway agent cannot be started until an agent user has been configured by the `DBMS_MGWADM.CREATE_AGENT` or `DBMS_MGWADM.ALTER_AGENT` subprograms.

## UNREGISTER\_FOREIGN\_QUEUE Procedure

This procedure removes a non-Oracle queue entity in Messaging Gateway.

### Syntax

```
DBMS_MGWADM.UNREGISTER_FOREIGN_QUEUE (
  name          IN VARCHAR2,
  linkname     IN VARCHAR2);
```

### Parameters

**Table 90–50 UNREGISTER\_FOREIGN\_QUEUE Procedure Parameters**

Parameter	Description
name	The queue name
linkname	The link name for the messaging system on which the queue exists

### Usage Notes

- This procedure does not remove the physical queue in the non-Oracle messaging system.
- All propagation jobs, subscribers and schedules referencing this queue must be removed before it can be unregistered. This procedure fails if a propagation job, subscriber, or propagation schedule references the non-Oracle queue.

## UNSCCHEDULE\_PROPAGATION Procedure

This procedure removes a propagation schedule.

---

---

**Note:** This subprogram has been deprecated as a result of improved technology (see [REMOVE\\_JOB Procedure](#) on page 90-52), and is retained only for reasons of backward compatibility.

---

---

### Syntax

```
DBMS_MGWADM.UNSCHEDULE_PROPAGATION (  
    schedule_id IN VARCHAR2 );
```

### Parameters

**Table 90–51 UNSCHEDULE\_PROPAGATION Procedure Parameters**

Parameter	Description
schedule_id	Identifies the propagation schedule to be removed



DBMS\_MGWMSG provides:

- Object types used by the canonical message types to convert message bodies.
- Methods, constants, and subprograms for working with Messaging Gateway message types.

**See Also:** [Chapter 90, "DBMS\\_MGWADM"](#) which describes the Messaging Gateway administrative interface, `DBMS_MGWADM`

This chapter contains the following topics:

- [Using DBMS\\_MGWMSG](#)
  - Security Model
  - Constants
  - Types
- [Summary of DBMS\\_MGWMSG Subprograms](#)

## Using DBMS\_MGWMSG

- [Security Model](#)
- [Constants](#)
- [Types](#)

## Security Model

The EXECUTE privilege is granted to PUBLIC on all types defined in the DBMS\_MGWMSG package as well as the canonical types. The DBMS\_MGWMSG packages and object types are owned by SYS.

---

---

**Note:** You must run the `catmgw.sql` script to load the Messaging Gateway packages and object types into the database. Refer to the *Oracle Streams Advanced Queuing User's Guide* for information on loading database objects and using DBMS\_MGWMSG.

---

---

## Constants

**Table 91–1 DBMS\_MGWMSG Constants: Value Types and Constants Representing the Type of Value for a SYS.MGW\_NAME\_VALUE\_T Object**

Value	Constant
TEXT_VALUE	CONSTANT BINARY_INTEGER := 1
RAW_VALUE	CONSTANT BINARY_INTEGER := 2
BOOLEAN_VALUE	CONSTANT BINARY_INTEGER := 3
BYTE_VALUE	CONSTANT BINARY_INTEGER := 4
SHORT_VALUE	CONSTANT BINARY_INTEGER := 5
INTEGER_VALUE	CONSTANT BINARY_INTEGER := 6
LONG_VALUE	CONSTANT BINARY_INTEGER := 7
FLOAT_VALUE	CONSTANT BINARY_INTEGER := 8
DOUBLE_VALUE	CONSTANT BINARY_INTEGER := 9
DATE_VALUE	CONSTANT BINARY_INTEGER := 10

**Table 91–2 DBMS\_MGWMSG Constants: Boolean Values—Constants Representing a Boolean as a Numeric Value**

Value	Constant
BOOLEAN_FALSE	CONSTANT BINARY_INTEGER := 0
BOOLEAN_TRUE	CONSTANT BINARY_INTEGER := 1

**Table 91–3 DBMS\_MGWMSG Constants: Case Comparisons**

Value	Constant
CASE_SENSITIVE	CONSTANT BINARY_INTEGER := 0
CASE_INSENSITIVE	CONSTANT BINARY_INTEGER := 1

**Table 91–4 Constants for the TIB/Rendezvous field type**

Value	Constant
TIBRVMSG_BOOL	CONSTANT INTEGER := 1
TIBRVMSG_F32	CONSTANT INTEGER := 2
TIBRVMSG_F64	CONSTANT INTEGER := 3
TIBRVMSG_I8	CONSTANT INTEGER := 4
TIBRVMSG_I16	CONSTANT INTEGER := 5
TIBRVMSG_I32	CONSTANT INTEGER := 6
TIBRVMSG_I64	CONSTANT INTEGER := 7
TIBRVMSG_IPADDR32	CONSTANT INTEGER := 8
TIBRVMSG_IPPORT16	CONSTANT INTEGER := 9
TIBRVMSG_DATETIME	CONSTANT INTEGER := 10
TIBRVMSG_F32ARRAY	CONSTANT INTEGER := 11

**Table 91-4 (Cont.) Constants for the TIB/Rendezvous field type**

<b>Value</b>	<b>Constant</b>
TIBRVMSG_F64ARRAY	CONSTANT INTEGER := 12
TIBRVMSG_I8ARRAY	CONSTANT INTEGER := 13
TIBRVMSG_I16ARRAY	CONSTANT INTEGER := 14
TIBRVMSG_I32ARRAY	CONSTANT INTEGER := 15
TIBRVMSG_I64ARRAY	CONSTANT INTEGER := 16
TIBRVMSG_OPAQUE	CONSTANT INTEGER := 17
TIBRVMSG_STRING	CONSTANT INTEGER := 18
TIBRVMSG_XML	CONSTANT INTEGER := 19

## Types

- [SYS.MGW\\_NAME\\_VALUE\\_T Type](#)
- [SYS.MGW\\_NAME\\_VALUE\\_T Type-Attribute Mapping](#)
- [SYS.MGW\\_NAME\\_TYPE\\_ARRAY\\_T Type](#)
- [SYS.MGW\\_TEXT\\_VALUE\\_T Type](#)
- [SYS.MGW\\_RAW\\_VALUE\\_T Type](#)
- [SYS.MGW\\_BASIC\\_MSG\\_T Type](#)
- [SYS.MGW\\_NUMBER\\_ARRAY\\_T Type](#)
- [SYS.MGW\\_TIBRV\\_FIELD\\_T Type](#)
- [SYS.MGW\\_TIBRV\\_MSG\\_T Type](#)

### SYS.MGW\_NAME\_VALUE\_T Type

This type specifies a named value. The name attribute, type attribute, and one of the <>\_value attributes are typically not NULL.

### Syntax

```

TYPE SYS.MGW_NAME_VALUE_T IS OBJECT(
  name          VARCHAR2(250),
  type          INTEGER,
  integer_value INTEGER,
  number_value  NUMBER,
  text_value    VARCHAR2(4000),
  raw_value     RAW(2000),
  date_value    DATE,

  -- Methods
  STATIC FUNCTION CONSTRUCT
  RETURN SYS.MGW_NAME_VALUE_T,

  STATIC FUNCTION CONSTRUCT_BOOLEAN (
    name  IN VARCHAR2,
    value IN INTEGER )
  RETURN SYS.MGW_NAME_VALUE_T,

  STATIC FUNCTION CONSTRUCT_BYTE (
    name  IN VARCHAR2,
    value IN INTEGER )
  RETURN SYS.MGW_NAME_VALUE_T,

  STATIC FUNCTION CONSTRUCT_SHORT (
    name  IN VARCHAR2,
    value IN INTEGER )
  RETURN SYS.MGW_NAME_VALUE_T,

  STATIC FUNCTION CONSTRUCT_INTEGER (
    name  IN VARCHAR2,
    value IN INTEGER )
  RETURN SYS.MGW_NAME_VALUE_T,

  STATIC FUNCTION CONSTRUCT_LONG (
    name  IN VARCHAR2,
```

```

        value IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_FLOAT (
    name   IN VARCHAR2,
    value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DOUBLE (
    name   IN VARCHAR2,
    value  IN NUMBER )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_TEXT (
    name   IN VARCHAR2,
    value  IN VARCHAR2 )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_RAW (
    name   IN VARCHAR2,
    value  IN RAW )
RETURN SYS.MGW_NAME_VALUE_T,

STATIC FUNCTION CONSTRUCT_DATE (
    name   IN VARCHAR2,
    value  IN DATE )
RETURN SYS.MGW_NAME_VALUE_T );

```

## Attributes

**Table 91-5 SYS.MGW\_NAME\_VALUE\_T Attributes**

Attribute	Description
name	Name associated with the value
type	Value type. Refer to the DBMS_MGWMSG.<>_VALUE constants in <a href="#">Table 91-1</a> . This indicates which Java datatype and class are associated with the value. It also indicates which attribute stores the value.
integer_value	Stores a numeric integer value
number_value	Stores a numeric float or large integer value
text_value	Stores a text value
raw_value	Stores a RAW (bytes) value
date_value	Stores a date value

## SYS.MGW\_NAME\_VALUE\_T Type-Attribute Mapping

[Table 91-6](#) shows the mapping between the value type and the attribute used to store the value.

**Table 91-6 SYS.MGW\_NAME\_VALUE\_T Type Attribute Mapping**

Type	Value Stored in Attribute
DBMS_MGWMSG.TEXT_VALUE	text_value
DBMS_MGWMSG.RAW_VALUE	raw_value
DBMS_MGWMSG.BOOLEAN_VALUE	integer_value

**Table 91–6 (Cont.) SYS.MGW\_NAME\_VALUE\_T Type Attribute Mapping**

Type	Value Stored in Attribute
DBMS_MGWMSG.BYTE_VALUE	integer_value
DBMS_MGWMSG.SHORT_VALUE	integer_value
DBMS_MGWMSG.INTEGER_VALUE	integer_value
DBMS_MGWMSG.LONG_VALUE	number_value
DBMS_MGWMSG.FLOAT_VALUE	number_value
DBMS_MGWMSG.DOUBLE_VALUE	number_value
DBMS_MGWMSG.DATE_VALUE	date_value

## CONSTRUCT Method

This method constructs a new SYS.MGW\_NAME\_VALUE\_T instance. All attributes are assigned a value of NULL.

### Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_NAME_VALUE_T;
```

## CONSTRUCT\_TYPE Methods

These methods construct a new SYS.MGW\_NAME\_VALUE\_T instance initialized with the value of a specific type. Each method sets the name and type attributes and one of the <>\_value attributes, as shown in the mappings in [Table 91–6](#).

### Syntax

```
STATIC FUNCTION CONSTRUCT_<> (
    name    IN VARCHAR2,
    value   IN datatype )
RETURN SYS.MGW_NAME_VALUE_T;
```

### Usage Notes

The construct\_boolean method sets the value to either DBMS\_MGWMSG.BOOLEAN\_TRUE or DBMS\_MGWMSG.BOOLEAN\_FALSE.

## SYS.MGW\_NAME\_TYPE\_ARRAY\_T Type

This type specifies an array of name-value pairs. An object of SYS.MGW\_NAME\_VALUE\_ARRAY\_T type can have up to 1024 elements.

### Syntax

```
TYPE SYS.MGW_NAME_VALUE_ARRAY_T
AS VARRAY (1024) OF SYS.MGW_NAME_VALUE_T;
```

## SYS.MGW\_TEXT\_VALUE\_T Type

This type specifies a TEXT value. It can store a large value as a CLOB or a smaller value (size <= 4000) as VARCHAR2. Only one of the < >\_value attributes should be set.

### Syntax

```
TYPE SYS.MGW_TEXT_VALUE_T IS OBJECT(
    small_value VARCHAR2(4000),
```



```

    large_value CLOB,

-- Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_TEXT_VALUE_T);

```

## Attributes

**Table 91-7 SYS.MGW\_TEXT\_VALUE\_T Attributes**

Attribute	Description
small_value	Small TEXT value. Used for values <= 4000.
large_value	Large TEXT value. Used when the value is too large for the small_value attribute.

## CONSTRUCT Method

This method constructs a new SYS.MGW\_TEXT\_VALUE\_T instance. All attributes are assigned a value of NULL.

### Syntax

```

STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_TEXT_VALUE_T;

```

## SYS.MGW\_RAW\_VALUE\_T Type

This type specifies a RAW value. This type can store a large value as a BLOB or a smaller value (size <= 2000) as RAW. You must set no more than one of the < >\_value attributes.

## Syntax

```

TYPE SYS.MGW_RAW_VALUE_T IS OBJECT(
    small_value RAW(2000),
    large_value BLOB,

--Methods
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_RAW_VALUE_T);

```

## Attributes

**Table 91-8 SYS.MGW\_RAW\_VALUE\_T Attributes**

Attribute	Description
small_value	Small RAW (bytes) value <= 2000
large_value	Large RAW value. Used when the value is too large for the small_value attribute.

## CONSTRUCT Method

This method constructs a new SYS.MGW\_RAW\_VALUE\_T instance. All attributes are assigned a value of NULL.

### Syntax

```

STATIC FUNCTION CONSTRUCT

```

```
RETURN SYS.MGW_RAW_VALUE_T;
```

## SYS.MGW\_BASIC\_MSG\_T Type

This is a canonical type for a basic TEXT or RAW message. Only a single TEXT or RAW value is typically set. An object of this type must not have both TEXT and RAW set to a not NULL value at the same time.

### Syntax

```
TYPE SYS.MGW_BASIC_MSG_T IS OBJECT(
  header      SYS.MGW_NAME_VALUE_ARRAY_T,
  text_body   SYS.MGW_TEXT_VALUE_T,
  raw_body    SYS.MGW_RAW_VALUE_T,

  --Methods
  STATIC FUNCTION CONSTRUCT
  RETURN SYS.MGW_BASIC_MSG_T);
```

### Attributes

**Table 91–9 SYS.MGW\_BASIC\_MSG\_T Attributes**

Attribute	Description
header	Message header information as an array of name-value pairs
text_body	Message body for a TEXT message
raw_body	Message body for a RAW (bytes) message

## CONSTRUCT Method

This method constructs a new SYS.MGW\_BASIC\_MSG\_T instance. All attributes are assigned a value of NULL.

### Syntax

```
STATIC FUNCTION CONSTRUCT
RETURN SYS.MGW_BASIC_MSG_T;
```

## SYS.MGW\_NUMBER\_ARRAY\_T Type

A type that specifies an array of numbers.

### Syntax

```
TYPE SYS.MGW_NUMBER_ARRAY_T AS VARRAY(1024) OF NUMBER;
```

## SYS.MGW\_TIBRV\_FIELD\_T Type

A type representing a TIB/Rendezvous message field, typically used in a read-only fashion to retrieve field information from a SYS.MGW\_TIBRV\_MSG\_T instance.

### Syntax

```
TYPE SYS.MGW_TIBRV_FIELD_T IS OBJECT(
  field_name   VARCHAR2(256),
  field_id     INTEGER,
  field_type   INTEGER,
  number_value NUMBER,
  number_array_value SYS.MGW_NUMBER_ARRAY_T,
```

```

text_value    VARCHAR2 (4000) ,
raw_value     RAW(2000) ,
date_value    DATE,
clob_value    CLOB,
blob_value    BLOB);

```

## Attributes

**Table 91–10 SYS.MGW\_TIBRV\_FIELD\_T Attributes**

Attribute	Description
field_name	Field name. This will be NULL if the field has no name.
field_id	Field identifier. If the field identifier is zero (0), then that field is considered not to have a field identifier. Otherwise the field identifier is a nonzero value that is unique for all fields of that message.
field_type	Field wire format datatype. The DBMS_MGWMSG.TIBRVMSG_<> constants represent valid values for this attribute. The value of this field discriminates which value attribute is used to store the field data.
number_value	Used to store a numeric value
number_array_value	Used to store a numeric array value
text_value	Used to store a small text value
raw_value	Used to store a small raw value
date_value	Used to store a date value
clob_value	Used to store a large text value. This is used when the text data will not fit in text_value, that is, when size is larger than 4000.
blob_value	Used to store a large raw value. This is used when the raw data will not fit in raw_value; that is, when size is larger than 2000.

## SYS.MGW\_TIBRV\_FIELD\_T Type and Attribute Mapping

Table 91–11 describes the mapping in type SYS.MGW\_TIBRV\_FIELD\_T between the field type and attribute used to store the value.

**Table 91–11 SYS.MGW\_TIBRV\_FIELD\_T Type and Attribute Mapping**

Field Type (DBMS_MGWMSG constant)	Value Stored in Attribute
TIBRVMSG_BOOL	number_value
TIBRVMSG_F32	number_value
TIBRVMSG_F64	number_value
TIBRVMSG_I8	number_value
TIBRVMSG_I16	number_value
TIBRVMSG_I32	number_value
TIBRVMSG_I64	number_value
TIBRVMSG_IPADDR32	text_value
TIBRVMSG_IPPORT16	number_value
TIBRVMSG_DATETIME	date_value

**Table 91–11 (Cont.) SYS.MGW\_TIBRV\_FIELD\_T Type and Attribute Mapping**

<b>Field Type (DBMS_MGWMSG constant)</b>	<b>Value Stored in Attribute</b>
TIBRVMSG_F32ARRAY	number_array_value
TIBRVMSG_F64ARRAY	number_array_value
TIBRVMSG_I8ARRAY	number_array_value
TIBRVMSG_I16ARRAY	number_array_value
TIBRVMSG_I32ARRAY	number_array_value
TIBRVMSG_I64ARRAY	number_array_value
TIBRVMSG_OPAQUE	raw_value or blob_value
TIBRVMSG_STRING	text_value or clob_value
TIBRVMSG_XML	raw_value or blob_value

### SYS.MGW\_TIBRV\_MSG\_T Type

A type representing a TIB/Rendezvous message. You must never directly reference the attributes of this type. Instead use the type methods.

### Syntax

```

TYPE SYS.MGW_TIBRV_MSG_T IS OBJECT(
    send_subject    VARCHAR2(256),
    reply_subject   VARCHAR2(256),
    cm_time_limit   NUMBER,
    cm_sender_name  VARCHAR2(256),
    cm_sequence_num NUMBER,
    fields          SYS.MGW_TIBRV_IFIELDS_T,
    clob_data1      CLOB,
    clob_data2      CLOB,
    clob_data3      CLOB,
    blob_data1      BLOB,
    blob_data2      BLOB,
    blob_data3      BLOB,

    STATIC FUNCTION construct
    RETURN SYS.MGW_TIBRV_MSG_T,

    MEMBER PROCEDURE add_bool (
        name IN  VARCHAR2,
        id   IN  INTEGER,
        value IN INTEGER ),

    MEMBER PROCEDURE add_f32 (
        name IN  VARCHAR2,
        id   IN  INTEGER,
        value IN  FLOAT ),

    MEMBER PROCEDURE add_f64 (
        name IN  VARCHAR2,
        id   IN  INTEGER,
        value IN  DOUBLE ),

    MEMBER PROCEDURE add_i8 (
        name IN  VARCHAR2,
        id   IN  INTEGER,

```

```
value IN INTEGER ),

MEMBER PROCEDURE add_i16 (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN INTEGER ),

MEMBER PROCEDURE add_i32 (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN INTEGER ),

MEMBER PROCEDURE add_i64 (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN NUMBER ),

MEMBER PROCEDURE add_ipaddr32 (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN VARCHAR2 ),

MEMBER PROCEDURE add_ipport16 (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN INTEGER ),

MEMBER PROCEDURE add_datetime (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN DATE ),

MEMBER PROCEDURE add_f32array (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_f64array (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i8array (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i16array (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i32array (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_i64array (
    name IN VARCHAR2,
    id IN INTEGER,
```

```
value IN SYS.MGW_NUMBER_ARRAY_T ),

MEMBER PROCEDURE add_string (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN VARCHAR2 ),

MEMBER PROCEDURE add_string (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN CLOB ),

MEMBER PROCEDURE add_opaque (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN RAW ),

MEMBER PROCEDURE add_opaque (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN BLOB ),

MEMBER PROCEDURE add_xml (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN RAW ),

MEMBER PROCEDURE add_xml (
    name IN VARCHAR2,
    id IN INTEGER,
    value IN BLOB ),

MEMBER PROCEDURE set_send_subject (
    value IN VARCHAR2 ),

MEMBER PROCEDURE set_reply_subject (
    value IN VARCHAR2 ),

MEMBER PROCEDURE set_cm_time_limit (
    value IN NUMBER ),

MEMBER PROCEDURE set_cm_sender_name (
    value IN VARCHAR2 ),

MEMBER PROCEDURE set_cm_sequence_num (
    value IN NUMBER ),

MEMBER FUNCTION get_send_subject
RETURN VARCHAR2,

MEMBER FUNCTION get_reply_subject
RETURN VARCHAR2,

MEMBER FUNCTION get_cm_time_limit
RETURN NUMBER,

MEMBER FUNCTION get_cm_sender_name
RETURN VARCHAR2,

MEMBER FUNCTION get_cm_sequence_num
```

```

RETURN NUMBER,

MEMBER FUNCTION get_field_count
RETURN INTEGER,

MEMBER FUNCTION get_field (
    idx    IN INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION get_field_by_name (
    name  IN VARCHAR2 )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION get_field_by_id (
    id    IN INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T,

MEMBER FUNCTION find_field_name (
    name  IN VARCHAR2,
    start_idx IN INTEGER )
RETURN INTEGER,

MEMBER FUNCTION find_field_id (
    id    IN INTEGER,
    start_idx IN INTEGER )
RETURN INTEGER
);

```

## Attributes

**Table 91–12** *SYS.MGW\_TIBRV\_MSG\_T* Type Attributes

Attribute	Description
send_subject	Send subject name
reply_subject	Reply subject name
cm_time_limit	Time limit for a certified message
cm_sender_name	Sender name of a certified message
cm_sequence_num	Sequence number of a certified message
fields	Collection of message fields
clob_data1	Used to store a large text value
clob_data2	Used to store a large text value
clob_data3	Used to store a large text value
blob_data1	Used to store a large raw value
blob_data2	Used to store a large raw value
blob_data3	Used to store a large raw value

## Construct Method

Constructs a new `SYS.MGW_TIBRV_MSG_T` instance. All attributes are set to `NULL`.

### Syntax

```

STATIC FUNCTION construct
RETURN SYS.MGW_TIBRV_MSG_T;

```

## ADD\_<> Methods

Adds a new field to the message.

### Syntax

```
MEMBER PROCEDURE ADD_<> (
    name IN VARCHAR2,
    id   IN INTEGER,
    value IN datatype );
```

### Parameters

**Table 91–13** *SYS.MGW\_TIBRV\_MSG\_T ADD\_<> Method Parameters*

Parameter	Description
name	Field name
id	Field identifier
value	Field data

Table 91–14 shows, for each add method, the field type that will be assigned and valid values for the field data.

**Table 91–14** *MGW\_TIBRV\_MSG\_T Add Method Field Types*

Method Name	Field Type Assigned	Comment
add_bool	TIBRVMSG_BOOL	Valid values: 0 (false), 1 (true)
add_f32	TIBRVMSG_F32	n/a
add_f64	TIBRVMSG_F64	n/a
add_i8	TIBRVMSG_I8	Valid range: -128...127
add_i16	TIBRVMSG_I16	Valid range: -32768...32767
add_i32	TIBRVMSG_I32	Valid range: -2147483648...2147483647
add_i64	TIBRVMSG_I64	n/a
add_ipaddr32	TIBRVMSG_IPADDR32	n/a
add_ipport16	TIBRVMSG_IPPORT16	n/a
add_datetime	TIBRVMSG_DATETIME	n/a
add_f32array	TIBRVMSG_F32ARRAY	n/a
add_f64array	TIBRVMSG_F64ARRAY	n/a
add_i8array	TIBRVMSG_I8ARRAY	Valid range: -128...127
add_i16array	TIBRVMSG_I16ARRAY	Valid range: -32768...32767
add_i32array	TIBRVMSG_I32ARRAY	Valid range: -2147483648...2147483647
add_i64array	TIBRVMSG_I64ARRAY	n/a
add_opaque	TIBRVMSG_OPAQUE	Value stored as RAW if size < 2000; otherwise value stored in BLOB
add_string	TIBRVMSG_STRING	Value stored as VARCHAR2 if size < 4000; otherwise value stored in CLOB



**Table 91–14 (Cont.) MGW\_TIBRV\_MSG\_T Add Method Field Types**

Method Name	Field Type Assigned	Comment
add_xml	TIBRVMSG_XML	Value stored as RAW if size < 2000; otherwise value stored in BLOB

**SET\_<> Methods**

Accessor methods to set an instance attribute to a specific value.

**Syntax**

```
MEMBER PROCEDURE SET_<> (
    value IN datatype );
```

**Parameters****Table 91–15 SYS.MGW\_TIBRV\_MSG\_T SET\_<> Method Parameters**

Parameter	Description
value	Value to be assigned

**GET\_<> Methods**

Accessor methods to retrieve the value for an instance attribute.

**Syntax**

```
MEMBER PROCEDURE GET_<>
RETURN datatype;
```

**Parameters**

None

**Return Values**

Returns the attribute value.

**GET\_FIELD\_COUNT Function**

Gets the number of message fields.

**Syntax**

```
MEMBER PROCEDURE get_field_count
RETURN INTEGER;
```

**Parameters**

None

**Return Values**

Returns the number of fields, or zero (0) if there are none.

**GET\_FIELD Function**

Retrieves field information for the field having a given field collection index. This method should only be called if the GET\_FIELD\_COUNT Function returns a nonzero

value and `idx` must specify a valid collection index; that is,  $1 \leq idx \leq \text{get\_field\_count}()$ .

### Syntax

```
MEMBER PROCEDURE get_field (
    idx    IN    INTEGER )
RETURN SYS.MGW_TIBRV_FIELD_T;
```

### Parameters

**Table 91–16** *SYS.MGW\_TIBRV\_MSG\_T GET\_FIELD Function Parameters*

Parameter	Description
<code>idx</code>	Specifies the 1-based field collection index of the field to retrieve

**Note:** A 1-based index begins at one (1) instead of zero (0).

### Return Values

Returns the field information.

## GET\_FIELD\_BY\_NAME Function

Retrieves field information for the first field that has a given field name. The name comparison is case-sensitive.

### Syntax

```
MEMBER PROCEDURE get_field_by_name (
    name IN  VARCHAR2 )
RETURN SYS.MGW_TIBRV_FIELD_T;
```

### Parameters

**Table 91–17** *SYS.MGW\_TIBRV\_MSG\_T GET\_FIELD\_BY\_NAME Function Parameters*

Parameter	Description
<code>name</code>	Specifies the field name to search for. This can be <code>NULL</code> to find the first field that does not have a field name.

### Return Values

Returns the field information, or `NULL` if no match was found.

## GET\_FIELD\_BY\_ID Function

Retrieves field information for the first field that has a given field identifier.

A field can have either a unique identifier or no identifier. If the field identifier value is zero (0) or `NULL`, then the field is considered to have no identifier. Otherwise, the identifier is a nonzero value that is unique for all the fields of this message.

### Syntax

```
MEMBER PROCEDURE get_field_by_id (
    id    IN    INTEGER )
```

```
RETURN SYS.MGW_TIBRV_FIELD_T;
```

## Parameters

**Table 91-18** *SYS.MGW\_TIBRV\_MSG\_T GET\_FIELD\_BY\_ID Function Parameters*

Parameter	Description
id	Specifies the field identifier to search for. This can be zero (0) or NULL to find the first field that does not have an identifier.

## Return Values

Returns the field information, or NULL if no match was found.

## FIND\_FIELD\_NAME Function

Searches for a field with a given field name, starting from a given index of the field collection. It returns the index of that field. The name comparison is case-sensitive. This function is useful for finding all the fields that have the same name.

## Syntax

```
MEMBER PROCEDURE find_field_name (
    name      IN VARCHAR2,
    start_idx IN INTEGER )
RETURN INTEGER;
```

## Parameters

**Table 91-19** *SYS.MGW\_TIBRV\_MSG\_T FIND\_FIELD\_NAME Function Parameters*

Parameter	Description
name	Specifies the field name to search for. This can be NULL to search for a field that does not have a field name.
start_idx	Specifies the 1-based field collection index from which the search should start.

## Return Values

Returns the field index (> 0) if a match was found, or zero (0) if no match was found.

## FIND\_FIELD\_ID Function

Searches for a field with a given field identifier, starting from a given index of the field collection. It returns the index of that field.

## Syntax

```
MEMBER PROCEDURE find_field_id (
    id      IN INTEGER,
    start_idx IN INTEGER )
RETURN INTEGER;
```

## Parameters

**Table 91–20** *SYS.MGW\_TIBRV\_MSG\_T FIND\_FIELD\_ID Function Parameters*

Parameter	Description
<code>id</code>	Specifies the field identifier to search for. This can be zero (0) or <code>NULL</code> to find a field that does not have an identifier.
<code>start_idx</code>	Specifies the 1-based field collection index from which the search should start.

## Return Values

Returns the field index (> 0) if a match was found, or zero (0) if no match was found.

---

## Summary of DBMS\_MGWMSG Subprograms

**Table 91–21 DBMS\_MGWMSG Package Subprograms**

Subprogram	Description
<a href="#">LCR_TO_XML Function</a> on page 91-22	Converts a SYS.ANYDATA object encapsulating a row LCR (LCR\$_ROW_RECORD) or a DDL LCR (LCR\$_DDL_RECORD) to a SYS.XMLTYPE object
<a href="#">NVARRAY_ADD Procedure</a> on page 91-23	Appends a name-value element to the end of a name-value array
<a href="#">NVARRAY_FIND_NAME Function</a> on page 91-24	Searches a name-value array for the element with the name you specify in p_name
<a href="#">NVARRAY_FIND_NAME_TYPE Function</a> on page 91-25	Searches a name-value array for an element with the name and value type you specify
<a href="#">NVARRAY_GET Function</a> on page 91-26	Gets the name-value element of the name you specify in p_name from a name-value array
<a href="#">NVARRAY_GET_BOOLEAN Function</a> on page 91-27	Gets the value of the name-value array element that you specify in p_name and with the BOOLEAN_VALUE value type
<a href="#">NVARRAY_GET_BYTE Function</a> on page 91-28	Gets the value of the name-value array element that you specify in p_name and with the BYTE_VALUE value type
<a href="#">NVARRAY_GET_DATE Function</a> on page 91-29	Gets the value of the name-value array element that you specify in p_name and with the DATE_VALUE value type
<a href="#">NVARRAY_GET_DOUBLE Function</a> on page 91-30	Gets the value of the name-value array element that you specify in p_name and with the DOUBLE_VALUE value type
<a href="#">NVARRAY_GET_FLOAT Function</a> on page 91-31	Gets the value of the name-value array element that you specify in p_name and with the FLOAT_VALUE value type
<a href="#">NVARRAY_GET_INTEGER Function</a> on page 91-32	Gets the value of the name-value array element that you specify in p_name and with the INTEGER_VALUE value type
<a href="#">NVARRAY_GET_LONG Function</a> on page 91-33	Gets the value of the name-value array element that you specify in p_name and with the LONG_VALUE value type
<a href="#">NVARRAY_GET_RAW Function</a> on page 91-34	Gets the value of the name-value array element that you specify in p_name and with the RAW_VALUE value type
<a href="#">NVARRAY_GET_SHORT Function</a> on page 91-35	Gets the value of the name-value array element that you specify in p_name and with the SHORT_VALUE value type
<a href="#">NVARRAY_GET_TEXT Function</a> on page 91-36	Gets the value of the name-value array element that you specify in p_name and with the TEXT_VALUE value type
<a href="#">XML_TO_LCR Function</a> on page 91-37	Converts a SYS.XMLTYPE object to a SYS.ANYDATA object encapsulating a row LCR (LCR\$_ROW_RECORD) or a DDL LCR (LCR\$_DDL_RECORD)

## LCR\_TO\_XML Function

This function converts a `SYS.ANYDATA` object encapsulating a row LCR (Logical Change Record, in this case a `LCR$_ROW_RECORD`) or a DDL LCR (`LCR$_DDL_RECORD`) to a `SYS.XMLTYPE` object.

**See Also:** [XML\\_TO\\_LCR Function](#) on page 91-37

### Syntax

```
DBMS_MGWMSG.LCR_TO_XML (
    p_anydata IN SYS.ANYDATA )
RETURN SYS.XMLTYPE;
```

### Parameters

**Table 91-22 LCR\_TO\_XML Function Parameters**

Parameter	Description
<code>p_anydata</code>	An <code>ANYDATA</code> object to be converted

### Return Values

Returns a `SYS.XMLTYPE` object.

### Usage Notes

An exception is raised if the encapsulated type `p_anydata` is not an LCR.

## NVARRAY\_ADD Procedure

This procedure appends a name-value element to the end of a name-value array.

### Syntax

```
DBMS_MGWMSG.NVARRAY_ADD (  
    p_array IN OUT SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_value IN     SYS.MGW_NAME_VALUE_T );
```

### Parameters

**Table 91-23 NVARRAY\_ADD Procedure Parameters**

Parameter	Description
p_array	On input, the name-value array instance to modify. If NULL, then a new array is created. On output, the modified name-value array instance.
p_value	The value to add. If NULL, then p_array is not changed.

## NVARRAY\_FIND\_NAME Function

This function searches a name-value array for the element with the name you specify in `p_name`.

### Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

### Parameters

**Table 91–24 NVARRAY\_FIND\_NAME Function Parameters**

Parameters	Description
<code>p_array</code>	The name-value array to search
<code>p_name</code>	The name to find
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns a positive integer that is the array index of the matching element or zero (0) if the specified name is not found.



## NVARRAY\_FIND\_NAME\_TYPE Function

This function searches a name-value array for an element with the name and value type you specify.

### Syntax

```
DBMS_MGWMSG.NVARRAY_FIND_NAME_TYPE (
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name     IN VARCHAR2,
  p_type     IN BINARY_INTEGER
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN BINARY_INTEGER;
```

### Parameters

**Table 91–25 NVARRAY\_FIND\_NAME\_TYPE Function Parameters**

Parameter	Description
p_array	The name-value array to search
p_name	The name to find
p_type	The value type. Refer to the value type constants in <a href="#">Table 91–1</a> on page 91-4.
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

### Return Values

Returns a positive integer that is the array index of the matching element, zero (0) if the specified name is not found, or negative one (-1) if the specified name is found but a type mismatch exists.

## NVARRAY\_GET Function

This function gets the name-value element of the name you specify in `p_name` from a name-value array.

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET (  
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name     IN VARCHAR2,  
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )  
RETURN SYS.MGW_NAME_VALUE_T;
```

### Parameters

**Table 91–26** *NVARRAY\_GET Function Parameters*

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the matching element, or `NULL` if the specified name is not found.

## NVARRAY\_GET\_BOOLEAN Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `BOOLEAN_VALUE` value type .

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_BOOLEAN (
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name     IN VARCHAR2,
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

### Parameters

**Table 91–27 NVARRAY\_GET\_BOOLEAN Function Parameters**

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_BYTE Function

This function gets the value of the name-value array element that you specify in p\_name and with the BYTE\_VALUE value type .

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_BYTE (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

### Parameters

**Table 91-28 NVARRAY\_GET\_BYTE Function Parameters**

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

### Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_DATE Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `DATE_VALUE` value type .

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_DATE (
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name     IN VARCHAR2,
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN DATE;
```

### Parameters

**Table 91–29 NVARRAY\_GET\_DATE Function Parameters**

Parameters	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARARRAY\_GET\_DOUBLE Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `DOUBLE_VALUE` value type .

### Syntax

```
DBMS_MGWMSG.NVARARRAY_GET_DOUBLE (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

### Parameters

**Table 91–30** *NVARARRAY\_GET\_DOUBLE Function Parameters*

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_FLOAT Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `FLOAT_VALUE` value type.

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_FLOAT (
  p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name     IN VARCHAR2,
  p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

### Parameters

**Table 91–31 NVARRAY\_GET\_FLOAT Function Parameters**

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_INTEGER Function

This function gets the value of the name-value array element that you specify in p\_name and with the INTEGER\_VALUE value type.

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_INTEGER (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

### Parameters

**Table 91–32 NVARRAY\_GET\_INTEGER Function Parameters**

Parameter	Description
p_array	The name-value array
p_name	The value name
p_compare	Name comparison method. Values are CASE_SENSITIVE and CASE_INSENSITIVE.

### Return Values

Returns the value, or NULL if either the specified name is not found or a type mismatch exists.



## NVARRAY\_GET\_LONG Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `LONG_VALUE` value type .

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_LONG (
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,
    p_name     IN VARCHAR2,
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN NUMBER;
```

### Parameters

**Table 91–33 NVARRAY\_GET\_LONG Function Parameters**

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_RAW Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `RAW_VALUE` value type .

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_RAW (
  p_array   IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name    IN VARCHAR2,
  p_compare IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN RAW;
```

### Parameters

**Table 91–34 NVARRAY\_GET\_RAW Function Parameters**

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_SHORT Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `SHORT_VALUE` value type.

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_SHORT (
  p_array   IN SYS.MGW_NAME_VALUE_ARRAY_T,
  p_name    IN VARCHAR2,
  p_compare IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )
RETURN INTEGER;
```

### Parameters

**Table 91–35 NVARRAY\_GET\_SHORT Function Parameters**

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## NVARRAY\_GET\_TEXT Function

This function gets the value of the name-value array element that you specify in `p_name` and with the `TEXT_VALUE` value type.

### Syntax

```
DBMS_MGWMSG.NVARRAY_GET_TEXT (  
    p_array    IN SYS.MGW_NAME_VALUE_ARRAY_T,  
    p_name     IN VARCHAR2,  
    p_compare  IN BINARY_INTEGER DEFAULT CASE_SENSITIVE )  
RETURN VARCHAR2;
```

### Parameters

**Table 91–36** *NVARRAY\_GET\_TEXT Function Parameters*

Parameter	Description
<code>p_array</code>	The name-value array
<code>p_name</code>	The value name
<code>p_compare</code>	Name comparison method. Values are <code>CASE_SENSITIVE</code> and <code>CASE_INSENSITIVE</code> .

### Return Values

Returns the value, or `NULL` if either the specified name is not found or a type mismatch exists.

## XML\_TO\_LCR Function

This function converts a `SYS.XMLTYPE` object to a `SYS.ANYDATA` object encapsulating a row LCR (`LCR$_ROW_RECORD`) or a DDL LCR (`LCR$_DDL_RECORD`).

**See Also:** [LCR\\_TO\\_XML Function](#) on page 91-22

### Syntax

```
DBMS_MGWMSG.XML_TO_LCR (  
    p_xmldata IN SYS.XMLTYPE )  
    RETURN SYS.ANYDATA;
```

### Parameters

**Table 91-37 XML\_TO\_LCR Function Parameters**

Parameter	Description
<code>p_xmldata</code>	An <code>XMLTYPE</code> object representing an LCR

### Return Values

Returns a `SYS.ANYDATA` object.

### Usage Notes

An exception is raised if `p_xmldata` cannot be converted to an LCR.



---

---

## DBMS\_MONITOR

The DBMS\_MONITOR package let you use PL/SQL for controlling additional tracing and statistics gathering.

The chapter contains the following topics:

- [Summary of DBMS\\_MONITOR Subprograms](#)

---

## Summary of DBMS\_MONITOR Subprograms

**Table 92–1 DBMS\_MONITOR Package Subprograms**

Subprogram	Description
<a href="#">CLIENT_ID_STAT_DISABLE Procedure</a> on page 92-3	Disables statistic gathering previously enabled for a given Client Identifier
<a href="#">CLIENT_ID_STAT_ENABLE Procedure</a> on page 92-4	Enables statistic gathering for a given Client Identifier
<a href="#">CLIENT_ID_TRACE_DISABLE Procedure</a> on page 92-5	Disables the trace previously enabled for a given Client Identifier globally for the database
<a href="#">CLIENT_ID_TRACE_ENABLE Procedure</a> on page 92-6	Enables the trace for a given Client Identifier globally for the database
<a href="#">DATABASE_TRACE_DISABLE Procedure</a> on page 92-7	Disables SQL trace for the whole database or a specific instance
<a href="#">DATABASE_TRACE_ENABLE Procedure</a> on page 92-8	Enables SQL trace for the whole database or a specific instance
<a href="#">SERV_MOD_ACT_STAT_DISABLE Procedure</a> on page 92-9	Disables statistic gathering enabled for a given combination of Service Name, MODULE and ACTION
<a href="#">SERV_MOD_ACT_STAT_ENABLE Procedure</a> on page 92-10	Enables statistic gathering for a given combination of Service Name, MODULE and ACTION
<a href="#">SERV_MOD_ACT_TRACE_DISABLE Procedure</a> on page 92-12	Disables the trace for ALL enabled instances for a or a given combination of Service Name, MODULE and ACTION name globally
<a href="#">SERV_MOD_ACT_TRACE_ENABLE Procedure</a> on page 92-13	Enables SQL tracing for a given combination of Service Name, MODULE and ACTION globally unless an instance_name is specified
<a href="#">SESSION_TRACE_DISABLE Procedure</a> on page 92-15	Disables the previously enabled trace for a given database session identifier (SID) on the local instance
<a href="#">SESSION_TRACE_ENABLE Procedure</a> on page 92-16	Enables the trace for a given database session identifier (SID) on the local instance



## CLIENT\_ID\_STAT\_DISABLE Procedure

This procedure will disable statistics accumulation for all instances and remove the accumulated results from V\$CLIENT\_STATS view enabled by the CLIENT\_ID\_STAT\_ENABLE Procedure.

### Syntax

```
DBMS_MONITOR.CLIENT_ID_STAT_DISABLE(  
    client_id          IN  VARCHAR2);
```

### Parameters

**Table 92–2** CLIENT\_ID\_STAT\_DISABLE Procedure Parameters

Parameter	Description
client_id	Client Identifier for which statistic aggregation is disabled

### Examples

To disable accumulation:

```
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_DISABLE('janedoe');
```

## CLIENT\_ID\_STAT\_ENABLE Procedure

This procedure enables statistic gathering for a given Client Identifier. Statistics gathering is global for the database and persistent across instance starts and restarts. That is, statistics are enabled for all instances of the same database, including restarts. Statistics are viewable through V\$CLIENT\_STATS views.

### Syntax

```
DBMS_MONITOR.CLIENT_ID_STAT_ENABLE(  
    client_id          IN  VARCHAR2);
```

### Parameters

**Table 92–3** CLIENT\_ID\_STAT\_ENABLE Procedure Parameters

Parameter	Description
client_id	Client Identifier for which statistic aggregation is enabled

### Examples

To enable statistic accumulation for a client with a given client ID:

```
EXECUTE DBMS_MONITOR.CLIENT_ID_STAT_ENABLE('janedoe');
```

## CLIENT\_ID\_TRACE\_DISABLE Procedure

This procedure will disable tracing enabled by the CLIENT\_ID\_TRACE\_ENABLE Procedure.

### Syntax

```
DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE(  
  client_id  IN  VARCHAR2);
```

### Parameters

**Table 92–4** CLIENT\_ID\_TRACE\_DISABLE Procedure Parameters

Parameter	Description
client_id	Client Identifier for which SQL tracing is disabled

### Examples

```
EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE ('janedoe');
```

## CLIENT\_ID\_TRACE\_ENABLE Procedure

This procedure will enable the trace for a given client identifier globally for the database.

### Syntax

```
DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE(
  client_id  IN  VARCHAR2,
  waits     IN  BOOLEAN DEFAULT TRUE,
  binds     IN  BOOLEAN DEFAULT FALSE,
  plan_stat IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 92–5** CLIENT\_ID\_TRACE\_ENABLE Procedure Parameters

Parameter	Description
client_id	Database Session Identifier for which SQL tracing is enabled
waits	If TRUE, wait information is present in the trace
binds	If TRUE, bind information is present in the trace
plan_stat	Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.

### Usage Notes

- The trace will be written to multiple trace files because more than one Oracle shadow process can work on behalf of a given client identifier.
- The tracing is enabled for all instances and persistent across restarts.

### Examples

```
EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE('janedoe', TRUE,
FALSE);
```

## DATABASE\_TRACE\_DISABLE Procedure

This procedure disables SQL trace for the whole database or a specific instance.

### Syntax

```
DBMS_MONITOR.DATABASE_TRACE_DISABLE(  
    instance_name IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 92–6** DATABASE\_TRACE\_DISABLE Procedure Parameters

Parameter	Description
instance_name	Disables tracing for the named instance

## DATABASE\_TRACE\_ENABLE Procedure

This procedure enables SQL trace for the whole database or a specific instance.

### Syntax

```
DBMS_MONITOR.DATABASE_TRACE_ENABLE(  
    waits          IN BOOLEAN DEFAULT TRUE,  
    binds          IN BOOLEAN DEFAULT FALSE,  
    instance_name  IN VARCHAR2 DEFAULT NULL,  
    plan_stat      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 92–7 DATABASE\_TRACE\_ENABLE Procedure Parameters**

Parameter	Description
waits	If TRUE, wait information will be present in the trace
binds	If TRUE, bind information will be present in the trace
instance_name	If set, restricts tracing to the named instance
plan_stat	Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.

## SERV\_MOD\_ACT\_STAT\_DISABLE Procedure

This procedure will disable statistics accumulation and remove the accumulated results from V\$SERV\_MOD\_ACT\_STATS view. Statistics disabling is persistent for the database. That is, service statistics are disabled for instances of the same database (plus dblinks that have been activated as a result of the enable).

### Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_STAT_DISABLE(
  service_name  IN VARCHAR2,
  module_name   IN VARCHAR2,
  action_name   IN VARCHAR2 DEFAULT ALL_ACTIONS);
```

### Parameters

**Table 92-8** *SERV\_MOD\_ACT\_STAT\_DISABLE Procedure Parameters*

Parameter	Description
service_name	Name of the service for which statistic aggregation is disabled
module_name	Name of the MODULE. An additional qualifier for the service. It is a required parameter.
action_name	Name of the ACTION. An additional qualifier for the Service and MODULE name. Omitting the parameter (or supplying ALL_ACTIONS constant) means enabling aggregation for all Actions for a given Service/MODULE combination. In this case, statistics are aggregated on the module level.

### Usage Notes

Regarding statistics gathering, when you change the module or action, the change takes effect when the next user call is executed in the session. For example, if a module is set to 'module 1' in a session, and the module is reset to 'module 2' in a user call in the session, then the module remains 'module 1' during this user call. The module is changed to 'module 2' in the next user call in the session.

## SERV\_MOD\_ACT\_STAT\_ENABLE Procedure

This procedure enables statistic gathering for a given combination of Service Name, MODULE and ACTION. Calling this procedure enables statistic gathering for a hierarchical combination of Service name, MODULE name, and ACTION name on all instances for the same database. Statistics are accessible by means of the V\$SERV\_MOD\_ACT\_STATS view.

### Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE (
    service_name    IN VARCHAR2,
    module_name     IN VARCHAR2,
    action_name     IN VARCHAR2 DEFAULT ALL_ACTIONS);
```

### Parameters

**Table 92–9 SERV\_MOD\_ACT\_STAT\_ENABLE Procedure Parameters**

Parameter	Description
service_name	Name of the service for which statistic aggregation is enabled
module_name	Name of the MODULE. An additional qualifier for the service. It is a required parameter.
action_name	Name of the ACTION. An additional qualifier for the Service and MODULE name. Omitting the parameter (or supplying ALL_ACTIONS constant) means enabling aggregation for all Actions for a given Service/MODULE combination. In this case, statistics are aggregated on the module level.

### Usage Notes

Enabling statistic aggregation for the given combination of Service/Module/Action names is slightly complicated by the fact that the Module/Action values can be empty strings which are indistinguishable from NULLs. For this reason, we adopt the following conventions:

A special constant (unlikely to be a real action names) is defined:

```
ALL_ACTIONS constant VARCHAR2 := '###ALL_ACTIONS';
```

Using ALL\_ACTIONS for an action specification means that aggregation is enabled for all actions with a given module name, while using NULL (or empty string) means that aggregation is enabled for an action whose name is an empty string.

Regarding statistics gathering, when you change the module or action, the change takes effect when the next user call is executed in the session. For example, if a module is set to 'module 1' in a session, and the module is reset to 'module 2' in a user call in the session, then the module remains 'module 1' during this user call. The module is changed to 'module 2' in the next user call in the session.

### Examples

To enable statistic accumulation for a given combination of Service name and MODULE:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE( 'APPS1', 'PAYROLL');
```

To enable statistic accumulation for a given combination of Service name, MODULE and ACTION:



```
EXECUTE  
DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE('APPS1', 'GLEDGER', 'DEBIT_ENTRY');
```

If both of the preceding commands are issued, statistics are accumulated as follows:

- For the APPS1 service, because accumulation for each Service Name is the default.
- For all actions in the PAYROLL Module.
- For the DEBIT\_ENTRY Action within the GLEDGER Module.

## SERV\_MOD\_ACT\_TRACE\_DISABLE Procedure

This procedure will disable the trace at ALL enabled instances for a given combination of Service Name, MODULE, and ACTION name globally.

### Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE(
  service_name  IN  VARCHAR2,
  module_name   IN  VARCHAR2,
  action_name   IN  VARCHAR2 DEFAULT ALL_ACTIONS,
  instance_name IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 92–10** *SERV\_MOD\_ACT\_TRACE\_DISABLE Procedure Parameters*

Parameter	Description
service_name	Name of the service for which tracing is disabled.
module_name	Name of the MODULE. An additional qualifier for the service
action_name	Name of the ACTION. An additional qualifier for the Service and MODULE name.
instance_name	If set, this restricts tracing to the named instance_name

### Usage Notes

Specifying NULL for the module\_name parameter means that statistics will no longer be accumulated for the sessions which do not set the MODULE attribute.

### Examples

To enable tracing for a Service named APPS1:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1',
  DBMS_MONITOR.ALL_MODULES, DBMS_MONITOR.ALL_ACTIONS, TRUE,
  FALSE, NULL);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE('APPS1');
```

To enable tracing for a given combination of Service and MODULE (all ACTIONS):

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1', 'PAYROLL',
  DBMS_MONITOR.ALL_ACTIONS, TRUE, FALSE, NULL);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_DISABLE('APPS1', 'PAYROLL');
```

## SERV\_MOD\_ACT\_TRACE\_ENABLE Procedure

This procedure will enable SQL tracing for a given combination of Service Name, MODULE and ACTION globally unless an instance\_name is specified.

### Syntax

```
DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE(
  service_name    IN VARCHAR2,
  module_name     IN VARCHAR2 DEFAULT ANY_MODULE,
  action_name     IN VARCHAR2 DEFAULT ANY_ACTION,
  waits          IN BOOLEAN DEFAULT TRUE,
  binds           IN BOOLEAN DEFAULT FALSE,
  instance_name  IN VARCHAR2 DEFAULT NULL,
  plan_stat      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 92–11** *SERV\_MOD\_ACT\_TRACE\_ENABLE Procedure Parameters*

Parameter	Description
service_name	Name of the service for which SQL trace is enabled
module_name	Name of the MODULE for which SQL trace is enabled. An optional additional qualifier for the service. If omitted, SQL trace is enabled for all modules and actions in a given service.
action_name	Name of the ACTION for which SQL trace is enabled. An optional additional qualifier for the Service and MODULE name. If omitted, SQL trace is enabled for all actions in a given module.
waits	If TRUE, wait information is present in the trace
binds	If TRUE, bind information is present in the trace
instance_name	If set, this restricts tracing to the named instance_name
plan_stat	Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.

### Usage Notes

- The procedure enables a trace for a given combination of Service, MODULE and ACTION name. The specification is strictly hierarchical: Service Name or Service Name/MODULE, or Service Name, MODULE, and ACTION name must be specified. Omitting a qualifier behaves like a wild-card, so that not specifying an ACTION means all ACTIONS. Using the ALL\_ACTIONS constant achieves the same purpose.
- This tracing is useful when an application MODULE and optionally known ACTION is experiencing poor service levels.
- By default, tracing is enabled globally for the database. The instance\_name parameter is provided to restrict tracing to named instances that are known, for example, to exhibit poor service levels.
- Tracing information is present in multiple trace files and you must use the trcsess tool to collect it into a single file.

- Specifying `NULL` for the `module_name` parameter means that statistics will be accumulated for the sessions which do not set the `MODULE` attribute.

## Examples

To enable tracing for a Service named APPS1:

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1',
        DBMS_MONITOR.ALL_MODULES, DBMS_MONITOR.ALL_ACTIONS, TRUE,
        FALSE, NULL);
```

To enable tracing for a given combination of Service and MODULE (all ACTIONS):

```
EXECUTE DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('APPS1', 'PAYROLL',
        DBMS_MONITOR.ALL_ACTIONS, TRUE, FALSE, NULL);
```

## SESSION\_TRACE\_DISABLE Procedure

This procedure will disable the trace for a given database session at the local instance.

### Syntax

```
DBMS_MONITOR.SESSION_TRACE_DISABLE(
  session_id      IN      BINARY_INTEGER DEFAULT NULL,
  serial_num      IN      BINARY_INTEGER DEFAULT NULL);
```

### Parameters

**Table 92–12** *SESSION\_TRACE\_DISABLE Procedure Parameters*

Parameter	Description
session_id	Database Session Identifier for which SQL trace is disabled
serial_num	Serial number for this session

### Usage Notes

If `serial_num` is NULL but `session_id` is specified, a session with a given `session_id` is no longer traced irrespective of its serial number. If both `session_id` and `serial_num` are NULL, the current user session is no longer traced. It is illegal to specify NULL `session_id` and non-NULL `serial_num`. In addition, the NULL values are default and can be omitted.

### Examples

To enable tracing for a client with a given client session ID:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(7,4634, TRUE, FALSE);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(7,4634);;
```

## SESSION\_TRACE\_ENABLE Procedure

This procedure enables a SQL trace for the given Session ID on the local instance

### Syntax

```
DBMS_MONITOR.SESSION_TRACE_ENABLE (
  session_id  IN  BINARY_INTEGER DEFAULT NULL,
  serial_num  IN  BINARY_INTEGER DEFAULT NULL,
  waits       IN  BOOLEAN DEFAULT TRUE,
  binds       IN  BOOLEAN DEFAULT FALSE,
  plan_stat   IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 92–13** SESSION\_TRACE\_ENABLE Procedure Parameters

Parameter	Description
session_id	Client Identifier for which SQL trace is enabled. If omitted (or NULL), the user's own session is assumed.
serial_num	Serial number for this session. If omitted (or NULL), only the session ID is used to determine a session.
waits	If TRUE, wait information is present in the trace
binds	If TRUE, bind information is present in the trace
plan_stat	Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.

### Usage Notes

The procedure enables a trace for a given database session, and is still useful for client/server applications. The trace is enabled only on the instance to which the caller is connected, since database sessions do not span instances. This tracing is strictly local to an instance.

If `serial_num` is NULL but `session_id` is specified, a session with a given `session_id` is traced irrespective of its serial number. If both `session_id` and `serial_num` are NULL, the current user session is traced. It is illegal to specify NULL `session_id` and non-NULL `serial_num`. In addition, the NULL values are default and can be omitted.

### Examples

To enable tracing for a client with a given client session ID:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(7,4634, TRUE, FALSE);
```

To disable tracing specified in the previous step:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_DISABLE(7,4634);
```

Either

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(5);
```

or

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(5, NULL);
```

traces the session with session ID of 5, while either

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE();
```

or

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL);
```

traces the current user session. Also,

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(NULL, NULL, TRUE, TRUE);
```

traces the current user session including waits and binds. The same can be also expressed using keyword syntax:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE(binds=>TRUE);
```





---

---

## DBMS\_MVIEW

DBMS\_MVIEW enables you to understand capabilities for materialized views and potential materialized views, including their rewrite availability. It also enables you to refresh materialized views that are not part of the same refresh group and purge logs.

---

---

**Note:** DBMS\_MVIEW is a synonym for DBMS\_SNAPSHOT.

---

---

**See Also:**

- *Oracle Database Advanced Replication* for more information about using materialized views in a replication environment
- *Oracle Database Data Warehousing Guide* for more information about using materialized views in a data warehousing environment

This chapter contains the following topics:

- [Using DBMS\\_MVIEW](#)
  - Operational Notes
  - Security Model
  - Rules and Limits
- [Summary of DBMS\\_MVIEW Subprograms](#)

## Using DBMS\_MVIEW

This section contains topics which relate to using the DBMS\_MVIEW package.

- [Operational Notes](#)
- [Security Model](#)
- [Rules and Limits](#)

## Operational Notes

If a query is less than 256 characters long, you can invoke `EXPLAIN_REWRITE` using the `EXECUTE` command from SQL\*Plus. Otherwise, the recommended method is to use a PL/SQL `BEGIN . . END` block, as shown in the examples in `/rdbms/demo/smxrw.sql`.

## Security Model

The `DBMS_MVIEW` package consists of a number of materialized view-related subprograms, each of which has different functionality and privilege requirements. The privilege model is generally based on the invoker's right. Each package subprogram is executed by first checking the privileges against the invoker. If all the required privileges are met, the subprogram will be executed. Otherwise, an insufficient privileges error will be thrown.

## Rules and Limits

The `EXPLAIN_REWRITE` procedure cannot accept queries longer than 32627 characters. These restrictions also apply when passing the defining query of a materialized view to the `EXPLAIN_MVIEW` procedure.

---

## Summary of DBMS\_MVIEW Subprograms

**Table 93–1 DBMS\_MVIEW Package Subprograms**

Subprogram	Description
<a href="#">BEGIN_TABLE_REORGANIZATION Procedure</a> on page 93-7	Performs a process to preserve materialized view data needed for refresh
<a href="#">END_TABLE_REORGANIZATION Procedure</a> on page 93-8	Ensures that the materialized view data for the master table is valid and that the master table is in the proper state
<a href="#">ESTIMATE_MVIEW_SIZE Procedure</a> on page 93-9	Estimates the size of a materialized view that you might create, in bytes and rows
<a href="#">EXPLAIN_MVIEW Procedure</a> on page 93-10	Explains what is possible with a materialized view or potential materialized view
<a href="#">EXPLAIN_REWRITE Procedure</a> on page 93-11	Explains why a query failed to rewrite or why the optimizer chose to rewrite a query with a particular materialized view or materialized views
<a href="#">I_AM_A_REFRESH Function</a> on page 93-13	Returns the value of the I_AM_REFRESH package state
<a href="#">PMARKER Function</a> on page 93-14	Returns a partition marker from a rowid, and is used for Partition Change Tracking (PCT)
<a href="#">PURGE_DIRECT_LOAD_LOG Procedure</a> on page 93-15	Purges rows from the direct loader log after they are no longer needed by any materialized views (used with data warehousing)
<a href="#">PURGE_LOG Procedure</a> on page 93-16	Purges rows from the materialized view log
<a href="#">PURGE_MVIEW_FROM_LOG Procedure</a> on page 93-17	Purges rows from the materialized view log
<a href="#">REFRESH Procedures</a> on page 93-18	Refreshes one or more materialized views that are not members of the same refresh group
<a href="#">REFRESH_ALL_MVIEWS Procedure</a> on page 93-20	Refreshes all materialized views that do not reflect changes to their master table or master materialized view
<a href="#">REFRESH_DEPENDENT Procedures</a> on page 93-21	Refreshes all table-based materialized views that depend on a specified master table or master materialized view, or list of master tables or master materialized views
<a href="#">REGISTER_MVIEW Procedure</a> on page 93-23	Enables the administration of individual materialized views
<a href="#">UNREGISTER_MVIEW Procedure</a> on page 93-25	Enables the administration of individual materialized views once invoked at a master site or master materialized view site to unregister a materialized view

## BEGIN\_TABLE\_REORGANIZATION Procedure

This procedure performs a process to preserve materialized view data needed for refresh. It must be called before a master table is reorganized.

### Syntax

```
DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION (  
  tabowner    IN   VARCHAR2,  
  tabname     IN   VARCHAR2);
```

### Parameters

**Table 93–2** *BEGIN\_TABLE\_REORGANIZATION Procedure Parameters*

Parameter	Description
tabowner	Owner of the table being reorganized
tabname	Name of the table being reorganized

## END\_TABLE\_REORGANIZATION Procedure

This procedure ensures that the materialized view data for the master table is valid and that the master table is in the proper state. It must be called after a master table is reorganized.

### Syntax

```
DBMS_MVIEW.END_TABLE_REORGANIZATION (  
    tabowner    IN    VARCHAR2,  
    tabname     IN    VARCHAR2);
```

### Parameters

**Table 93–3** *END\_TABLE\_REORGANIZATION Procedure Parameters*

Parameter	Description
tabowner	Owner of the table being reorganized
tabname	Name of the table being reorganized



## ESTIMATE\_MVIEW\_SIZE Procedure

This procedure estimates the size of a materialized view that you might create, in bytes and number of rows.

### Syntax

```
DBMS_MVIEW. ESTIMATE_MVIEW_SIZE (
  stmt_id      IN  VARCHAR2,
  select_clause IN  VARCHAR2,
  num_rows     OUT NUMBER,
  num_bytes    OUT NUMBER);
```

### Parameters

**Table 93–4** ESTIMATE\_MVIEW\_SIZE Procedure Parameters

Parameter	Description
stmt_id	Arbitrary string used to identify the statement in an EXPLAIN PLAN
select_clause	The SELECT statement to be analyzed
num_rows	Estimated cardinality
num_bytes	Estimated number of bytes

## EXPLAIN\_MVIEW Procedure

This procedure enables you to learn what is possible with a materialized view or potential materialized view. For example, you can determine if a materialized view is fast refreshable and what types of query rewrite you can perform with a particular materialized view.

Using this procedure is straightforward. You simply call `DBMS_MVIEW.EXPLAIN_MVIEW`, passing in as parameters the schema and materialized view name for an existing materialized view. Alternatively, you can specify the `SELECT` string or `CREATE MATERIALIZED VIEW` statement for a potential materialized view. The materialized view or potential materialized view is then analyzed and the results are written into either a table called `MV_CAPABILITIES_TABLE`, which is the default, or to an array called `MSG_ARRAY`.

The procedure is overloaded:

- The first version is for explaining an existing or potential materialized view with output to `MV_CAPABILITIES_TABLE`.
- The second version is for explaining an existing or potential materialized view with output to a `VARRAY`.

### Syntax

```
DBMS_MVIEW.EXPLAIN_MVIEW (
  mv          IN VARCHAR2,
  statement_id IN VARCHAR2:= NULL);
```

```
DBMS_MVIEW.EXPLAIN_MVIEW (
  mv          IN VARCHAR2,
  msg_array   OUT SYS.ExplainMVArrayType);
```

### Parameters

**Table 93–5** *EXPLAIN\_MVIEW Procedure Parameters*

Parameter	Description
<code>mv</code>	The name of an existing materialized view (optionally qualified with the owner name separated by a ".") or a <code>SELECT</code> statement or a <code>CREATE MATERIALIZED VIEW</code> statement for a potential materialized view.
<code>statement_id</code>	A client-supplied unique identifier to associate output rows with specific invocations of <code>EXPLAIN_MVIEW</code>
<code>msg_array</code>	The PL/SQL <code>VARRAY</code> that receives the output. Use this parameter to direct <code>EXPLAIN_MVIEW</code> 's output to a PL/SQL <code>VARRAY</code> rather than <code>MV_CAPABILITIES_TABLE</code> .

### Usage Notes

You must run the `utlxmlv.sql` script to create `MV_CAPABILITIES_TABLE` in the current schema prior to calling `EXPLAIN_MVIEW` except when you direct output to a `VARRAY`. The script is found in the `ADMIN` directory.

## EXPLAIN\_REWRITE Procedure

This procedure enables you to learn why a query failed to rewrite, or, if it rewrites, which materialized views will be used. Using the results from the procedure, you can take the appropriate action needed to make a query rewrite if at all possible. The query specified in the `EXPLAIN_REWRITE` statement is never actually executed.

A demo file, `xrwutl.sql`, is available to help format the output from `EXPLAIN_REWRITE`.

### Syntax

You can obtain the output from `DBMS_MVIEW.EXPLAIN_REWRITE` in two ways. The first is to use a table, while the second is to create a `VARRAY`. The following shows the basic syntax for using an output table:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
  query          VARCHAR2,
  mv             VARCHAR2(30),
  statement_id   VARCHAR2(30));
```

You can create an output table called `REWRITE_TABLE` by executing the `utl_xrw.sql` script.

The `query` parameter is a text string representing the SQL query. The parameter, `mv`, is a fully qualified materialized view name in the form of `schema.mv`. This is an optional parameter. When it is not specified, `EXPLAIN_REWRITE` returns any relevant messages regarding all the materialized views considered for rewriting the given query. When `schema` is omitted and only `mv` is specified, `EXPLAIN_REWRITE` looks for the materialized view in the current schema.

If you want to direct the output of `EXPLAIN_REWRITE` to a `VARRAY` instead of a table, you should call the procedure as follows:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
  query          [VARCHAR2 | CLOB],
  mv             VARCHAR2(30),
  output_array   SYS.RewriteArrayType);
```

Note that if the query is less than 256 characters long, `EXPLAIN_REWRITE` can be easily invoked with the `EXECUTE` command from `SQL*Plus`. Otherwise, the recommended method is to use a `PL/SQL BEGIN . . . END` block, as shown in the examples in `/rdbms/demo/smxrw*`.

You can also use `EXPLAIN_REWRITE` with multiple materialized views, in which case the syntax will be the same as with a single materialized view, except that the materialized views are specified by a comma-delimited string. For example, to find out whether a given set of materialized views `mv1`, `mv2`, and `mv3` could be used to rewrite the query, `query_txt`, and, if not, why not, use `EXPLAIN_REWRITE` as follows:

```
DBMS_MVIEW.EXPLAIN_REWRITE(query_txt, 'mv1, mv2, mv3')
```

See *Oracle Database Data Warehousing Guide* for more information on using the `EXPLAIN_REWRITE` procedure.

## Parameters

**Table 93–6** *EXPLAIN\_REWRITE Procedure Parameters*

Parameter	Description
<code>query</code>	SQL <code>SELECT</code> statement to be explained
<code>mv</code>	The fully qualified name of an existing materialized view in the form of <code>SCHEMA.MV</code> . For multiple materialized views, you can provide a comma-delimited list of names.
<code>statement_id</code>	A client-supplied unique identifier to distinguish output messages
<code>msg_array</code>	The PL/SQL <code>VARRAY</code> that receives the output. Use this parameter to direct <code>EXPLAIN_REWRITE</code> 's output to a PL/SQL <code>VARRAY</code> .

## Usage Notes

To obtain the output into a table, you must run the `utl_xrw.sql` script before calling `EXPLAIN_REWRITE`. This script creates a table named `REWRITE_TABLE` in the current schema.

## I\_AM\_A\_REFRESH Function

This function returns the value of the I\_AM\_REFRESH package state.

### Syntax

```
DBMS_MVIEW.I_AM_A_REFRESH  
RETURN BOOLEAN;
```

### Return Values

A return value of `true` indicates that all local replication triggers for materialized views are effectively disabled in this session because each replication trigger first checks this state. A return value of `false` indicates that these triggers are enabled.

## PMARKER Function

This function returns a partition marker from a rowid. It is used for Partition Change Tracking (PCT).

### Syntax

```
DBMS_MVIEW.PMARKER(  
    rid IN ROWID)  
RETURN NUMBER;
```

### Parameters

**Table 93–7** *PMARKER Function Parameters*

Parameter	Description
<code>rid</code>	The rowid of a row entry in a master table

## PURGE\_DIRECT\_LOAD\_LOG Procedure

This procedure removes entries from the direct loader log after they are no longer needed for any known materialized view. This procedure usually is used in environments using Oracle's data warehousing technology.

**See Also:** *Oracle Database Data Warehousing Guide* for more information

### Syntax

```
DBMS_MVIEW.PURGE_DIRECT_LOAD_LOG ( ) ;
```

## PURGE\_LOG Procedure

This procedure purges rows from the materialized view log.

### Syntax

```
DBMS_MVIEW.PURGE_LOG (
  master      IN   VARCHAR2,
  num         IN   BINARY_INTEGER := 1,
  flag        IN   VARCHAR2      := 'NOP');
```

### Parameters

**Table 93–8** *PURGE\_LOG Procedure Parameters*

Parameter	Description
<code>master</code>	Name of the master table or master materialized view.
<code>num</code>	<p>Number of least recently refreshed materialized views whose rows you want to remove from materialized view log. For example, the following statement deletes rows needed to refresh the two least recently refreshed materialized views:</p> <pre>DBMS_MVIEW.PURGE_LOG('master_table', 2);</pre> <p>To delete all rows in the materialized view log, indicate a high number of materialized views to disregard, as in this example:</p> <pre>DBMS_MVIEW.PURGE_LOG('master_table', 9999);</pre> <p>This statement completely purges the materialized view log that corresponds to <code>master_table</code> if fewer than 9999 materialized views are based on <code>master_table</code>. A simple materialized view whose rows have been purged from the materialized view log must be completely refreshed the next time it is refreshed.</p>
<code>flag</code>	<p>Specify <code>delete</code> to guarantee that rows are deleted from the materialized view log for at least one materialized view. This parameter can override the setting for the parameter <code>num</code>. For example, the following statement deletes rows from the materialized view log that has dependency rows in the least recently refreshed materialized view:</p> <pre>DBMS_MVIEW.PURGE_LOG('master_table', 1, 'delete');</pre>



## PURGE\_MVIEW\_FROM\_LOG Procedure

This procedure is called on the master site or master materialized view site to delete the rows in materialized view refresh related data dictionary tables maintained at the master for the specified materialized view identified by its `mview_id` or the combination of the `mviewowner`, `mviewname`, and the `mviewsite`. If the materialized view specified is the oldest materialized view to have refreshed from any of the master tables or master materialized views, then the materialized view log is also purged. This procedure does not unregister the materialized view.

### Syntax

```
DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
  mview_id      IN  BINARY_INTEGER |
  mviewowner    IN  VARCHAR2,
  mviewname     IN  VARCHAR2,
  mviewsite     IN  VARCHAR2);
```

---

**Note:** This procedure is overloaded. The `mview_id` parameter is mutually exclusive with the three remaining parameters: `mviewowner`, `mviewname`, and `mviewsite`.

---

### Parameters

**Table 93–9** *PURGE\_MVIEW\_FROM\_LOG Procedure Parameters*

Parameter	Description
<code>mview_id</code>	<p>If you want to execute this procedure based on the identification of the target materialized view, specify the materialized view identification using the <code>mview_id</code> parameter. Query the <code>DBA_BASE_TABLE_MVIEWS</code> view at the materialized view log site for a listing of materialized view IDs.</p> <p>Executing this procedure based on the materialized view identification is useful if the target materialized view is not listed in the list of registered materialized views (<code>DBA_REGISTERED_MVIEWS</code>).</p>
<code>mviewowner</code>	<p>If you do not specify an <code>mview_id</code>, enter the owner of the target materialized view using the <code>mviewowner</code> parameter. Query the <code>DBA_REGISTERED_MVIEWS</code> view at the materialized view log site to view the materialized view owners.</p>
<code>mviewname</code>	<p>If you do not specify an <code>mview_id</code>, enter the name of the target materialized view using the <code>mviewname</code> parameter. Query the <code>DBA_REGISTERED_MVIEWS</code> view at the materialized view log site to view the materialized view names.</p>
<code>mviewsite</code>	<p>If you do not specify an <code>mview_id</code>, enter the site of the target materialized view using the <code>mviewsite</code> parameter. Query the <code>DBA_REGISTERED_MVIEWS</code> view at the materialized view log site to view the materialized view sites.</p>

### Usage Notes

If there is an error while purging one of the materialized view logs, the successful purge operations of the previous materialized view logs are not rolled back. This is to minimize the size of the materialized view logs. In case of an error, this procedure can be invoked again until all the materialized view logs are purged.

## REFRESH Procedures

This procedure refreshes a list of materialized views.

### Syntax

```
DBMS_MVIEW.REFRESH (
  { list          IN      VARCHAR2,
    | tab         IN      DBMS_UTILITY.UNCL_ARRAY, }
  method         IN      VARCHAR2      := NULL,
  rollback_seg   IN      VARCHAR2      := NULL,
  push_deferred_rpc IN    BOOLEAN      := true,
  refresh_after_errors IN    BOOLEAN    := false,
  purge_option   IN      BINARY_INTEGER := 1,
  parallelism    IN      BINARY_INTEGER := 0,
  heap_size      IN      BINARY_INTEGER := 0,
  atomic_refresh IN      BOOLEAN      := true,
  nested         IN      BOOLEAN      := false);
```

---

**Note:** This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

---

### Parameters

**Table 93–10 REFRESH Procedure Parameters**

Parameter	Description
<code>list   tab</code>	<p>Comma-delimited list of materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your local database.</p> <p>Alternatively, you may pass in a PL/SQL index-by table of type <code>DBMS_UTILITY.UNCL_ARRAY</code>, where each element is the name of a materialized view.</p>
<code>method</code>	<p>A string of refresh methods indicating how to refresh the listed materialized views. An <code>f</code> indicates fast refresh, <code>?</code> indicates force refresh, <code>C</code> or <code>c</code> indicates complete refresh, and <code>A</code> or <code>a</code> indicates always refresh. <code>A</code> and <code>C</code> are equivalent. <code>P</code> or <code>p</code> refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables.</p> <p>If a materialized view does not have a corresponding refresh method (that is, if more materialized views are specified than refresh methods), then that materialized view is refreshed according to its default refresh method. For example, consider the following EXECUTE statement within SQL*Plus:</p> <pre>DBMS_MVIEW.REFRESH   ('countries_mv,regions_mv,hr.employees_mv','cf');</pre> <p>This statement performs a complete refresh of the <code>countries_mv</code> materialized view, a fast refresh of the <code>regions_mv</code> materialized view, and a default refresh of the <code>hr.employees</code> materialized view.</p>
<code>rollback_seg</code>	<p>Name of the materialized view site rollback segment to use while refreshing materialized views</p>

**Table 93–10 (Cont.) REFRESH Procedure Parameters**

Parameter	Description
push_deferred_rpc	Used by updatable materialized views only. Set this parameter to <code>true</code> if you want to push changes from the materialized view to its associated master tables or master materialized views before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	If this parameter is <code>true</code> , an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the <code>DEFERROR</code> view for the materialized view's master table or master materialized view. If this parameter is <code>true</code> and <code>atomic_refresh</code> is <code>false</code> , this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
purge_option	If you are using the parallel propagation mechanism (in other words, <code>parallelism</code> is set to 1 or greater), 0 means do not purge, 1 means lazy purge, and 2 means aggressive purge. In most cases, lazy purge is the optimal setting. Set <code>purge</code> to aggressive to trim the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set this parameter to 0 and occasionally execute <code>PUSH</code> with this parameter set to 2 to reduce the queue.
parallelism	0 specifies serial propagation. $n > 1$ specifies parallel propagation with $n$ parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance.  Note: Do not set this parameter unless directed to do so by Oracle Support Services.
atomic_refresh	If this parameter is set to <code>true</code> , then the list of materialized views is refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated.  If this parameter is set to <code>false</code> , then each of the materialized views is refreshed in a separate transaction.  As part of complete refresh, if <code>truncate</code> is used (non-atomic refresh), unique index rebuild is executed. <code>INDEX REBUILD</code> automatically computes statistics. Thus, statistics are updated for truncated tables.
nested	If <code>true</code> , then perform nested refresh operations for the specified set of materialized views. Nested refresh operations refresh all the depending materialized views and the specified set of materialized views based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.

## REFRESH\_ALL\_MVIEWS Procedure

This procedure refreshes all materialized views that have the following properties:

- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view `DBA_MVIEWS`.

This procedure is intended for use with data warehouses.

### Syntax

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS (
  number_of_failures    OUT  BINARY_INTEGER,
  method                IN   VARCHAR2         := NULL,
  rollback_seg         IN   VARCHAR2         := NULL,
  refresh_after_errors IN   BOOLEAN          := false,
  atomic_refresh       IN   BOOLEAN          := true);
```

### Parameters

**Table 93–11 REFRESH\_ALL\_MVIEWS Procedure Parameters**

Parameter	Description
<code>number_of_failures</code>	Returns the number of failures that occurred during processing
<code>method</code>	A single refresh method indicating the type of refresh to perform for each materialized view that is refreshed. <code>F</code> or <code>f</code> indicates fast refresh, <code>?</code> indicates force refresh, <code>C</code> or <code>c</code> indicates complete refresh, and <code>A</code> or <code>a</code> indicates always refresh. <code>A</code> and <code>C</code> are equivalent. If no method is specified, a materialized view is refreshed according to its default refresh method. <code>P</code> or <code>p</code> refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables.
<code>rollback_seg</code>	Name of the materialized view site rollback segment to use while refreshing materialized views
<code>refresh_after_errors</code>	If this parameter is <code>true</code> , an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the <code>DEFERROR</code> view for the materialized view's master table or master materialized view. If this parameter is <code>true</code> and <code>atomic_refresh</code> is <code>false</code> , this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
<code>atomic_refresh</code>	If this parameter is set to <code>true</code> , then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated.  If this parameter is set to <code>false</code> , then each of the refreshed materialized views is refreshed in a separate transaction.

## REFRESH\_DEPENDENT Procedures

This procedure refreshes all materialized views that have the following properties:

- The materialized view depends on a master table or master materialized view in the list of specified masters.
- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA\_MVIEWS.

This procedure is intended for use with data warehouses.

### Syntax

```
DBMS_MVIEW.REFRESH_DEPENDENT (
  number_of_failures  OUT  BINARY_INTEGER,
  { list              IN   VARCHAR2,
    | tab             IN   DBMS_UTILITY.UNCL_ARRAY, }
  method              IN   VARCHAR2      := NULL,
  rollback_seg        IN   VARCHAR2      := NULL,
  refresh_after_errors IN   BOOLEAN       := false,
  atomic_refresh      IN   BOOLEAN       := true,
  nested              IN   BOOLEAN       := false);
```

---

**Note:** This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

---

### Parameters

**Table 93–12** REFRESH\_DEPENDENT Procedure Parameters

Parameter	Description
<code>number_of_failures</code>	Returns the number of failures that occurred during processing
<code>list</code>   <code>tab</code>	Comma-delimited list of master tables or master materialized views on which materialized views can depend. (Synonyms are not supported.) These tables and the materialized views that depend on them can be located in different schemas. However, all of the tables and materialized views must be in your local database.  Alternatively, you may pass in a PL/SQL index-by table of type <code>DBMS_UTILITY.UNCL_ARRAY</code> , where each element is the name of a table.

**Table 93–12 (Cont.) REFRESH\_DEPENDENT Procedure Parameters**

Parameter	Description
method	<p>A string of refresh methods indicating how to refresh the dependent materialized views. All of the materialized views that depend on a particular table are refreshed according to the refresh method associated with that table. F or f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. P or p refreshes by recomputing the rows in the materialized view affected by changed partitions in the detail tables.</p> <p>If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any materialized view that depends on that table is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus:</p> <pre>DBMS_MVIEW.REFRESH_DEPENDENT   ('employees,departments,hr.regions','cf');</pre> <p>performs a complete refresh of the materialized views that depend on the employees table, a fast refresh of the materialized views that depend on the departments table, and a default refresh of the materialized views that depend on the hr.regions table.</p>
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views
refresh_after_errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.
atomic_refresh	<p>If this parameter is set to true, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated.</p> <p>If this parameter is set to false, then each of the refreshed materialized views is refreshed in a separate transaction.</p>
nested	If true, then perform nested refresh operations for the specified set of tables. Nested refresh operations refresh all the depending materialized views of the specified set of tables based on a dependency order to ensure the nested materialized views are truly fresh with respect to the underlying base tables.

## REGISTER\_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to register a materialized view.

Note that, typically, a materialized view is registered automatically during materialized view creation. You should only run this procedure to manually register a materialized view if the automatic registration failed or if the registration information was deleted.

### Syntax

```
DBMS_MVIEW.REGISTER_MVIEW (
  mviewowner  IN  VARCHAR2,
  mviewname   IN  VARCHAR2,
  mviewsite   IN  VARCHAR2,
  mview_id    IN  DATE | BINARY_INTEGER,
  flag        IN  BINARY_INTEGER,
  qry_txt     IN  VARCHAR2,
  rep_type    IN  BINARY_INTEGER := DBMS_MVIEW.REG_UNKNOWN);
```

### Parameters

**Table 93–13 REGISTER\_MVIEW Procedure Parameters**

Parameter	Description
mviewowner	Owner of the materialized view.
mviewname	Name of the materialized view.
mviewsite	Name of the materialized view site for a materialized view registering at an Oracle database version 8.x and higher master site or master materialized view site. This name should not contain any double quotes.
mview_id	The identification number of the materialized view. Specify an Oracle database version 8.x and higher materialized view as a BINARY_INTEGER. Specify an Oracle database version 7 materialized view registering at an Oracle database version 8.x and higher master sites or master materialized view sites as a DATE.
flag	A constant that describes the properties of the materialized view being registered. Valid constants that can be assigned include the following: DBMS_MVIEW.REG_ROWID_MVIEW for a rowid materialized view DBMS_MVIEW.REG_PRIMARY_KEY_MVIEW for a primary key materialized view DBMS_MVIEW.REG_OBJECT_ID_MVIEW for an object id materialized view DBMS_MVIEW.REG_FAST_REFRESHABLE_MVIEW for a materialized view that can be fast refreshed DBMS_MVIEW.REG_UPDATABLE_MVIEW for a materialized view that is updatable A materialized view can have more than one of these properties. In this case, use the plus sign (+) to specify more than one property. For example, if a primary key materialized view can be fast refreshed, you can enter the following for this parameter: DBMS_MVIEW.REG_PRIMARY_KEY_MVIEW + DBMS_MVIEW.REG_FAST_REFRESHABLE_MVIEW You can determine the properties of a materialized view by querying the ALL_MVIEWS data dictionary view.
qry_txt	The first 32,000 bytes of the materialized view definition query.

**Table 93–13 (Cont.) REGISTER\_MVIEW Procedure Parameters**

Parameter	Description
rep_type	<p>Version of the materialized view. Valid constants that can be assigned include the following:</p> <p>DBMS_MVIEW.REG_V7_SNAPSHOT if the materialized view is at an Oracle database version 7 site</p> <ul style="list-style-type: none"> <li>■ DBMS_MVIEW.REG_V8_SNAPSHOT</li> </ul> <p>if the materialized view is at an Oracle database version 8.x or higher site</p> <p>DBMS_MVIEW.REG_UNKNOWN (the default) if you do not know whether the materialized view is at an Oracle database version 7 site or an Oracle database version 8.x (or higher) site</p>

## Usage Notes

This procedure is invoked at the master site or master materialized view site by a remote materialized view site using a remote procedure call. If REGISTER\_MVIEW is called multiple times with the same mviewowner, mviewname, and mviewsite, then the most recent values for mview\_id, flag, and qry\_txt are stored. If a query exceeds the maximum VARCHAR2 size, then qry\_txt contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the value of mview\_id must be looked up in the materialized view data dictionary views by the person who calls the procedure.



## UNREGISTER\_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to unregister a materialized view.

### Syntax

```
DBMS_MVIEW.UNREGISTER_MVIEW (  
    mviewowner      IN  VARCHAR2,  
    mviewname       IN  VARCHAR2,  
    mviewsite       IN  VARCHAR2);
```

### Parameters

**Table 93–14 UNREGISTER\_MVIEW Procedure Parameters**

Parameters	Description
mviewowner	Owner of the materialized view
mviewname	Name of the materialized view
mviewsite	Name of the materialized view site



---

---

## DBMS\_NETWORK\_ACL\_ADMIN

The DBMS\_NETWORK\_ACL\_ADMIN package provides the interface to administer the network Access Control List (ACL).

**See Also:** For more information, see "Managing Fine-grained Access to External Network Services" in Oracle Database Security Guide

The chapter contains the following topics:

- [Using DBMS\\_NETWORK\\_ACL\\_ADMIN](#)
  - Examples
- [Summary of DBMS\\_NETWORK\\_ACL\\_ADMIN Subprograms](#)

---

## Using DBMS\_NETWORK\_ACL\_ADMIN

- [Examples](#)

## Examples

### Example 1

Grant the connect and resolve privileges for host www.us.oracle.com to SCOTT.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(acl          => 'www.xml',
                                     description => 'WWW ACL',
                                     principal   => 'SCOTT',
                                     is_grant    => true,
                                     privilege   => 'connect');

  DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE(acl          => 'www.xml',
                                       principal   => 'SCOTT',
                                       is_grant    => true,
                                       privilege   => 'resolve');

  DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL(acl => 'www.xml',
                                   host => 'www.us.oracle.com');

END;
/
COMMIT;
```

### Example 2

Grant the resolve privilege for www.us.oracle.com to ADAMS. Since an ACL for www.us.oracle.com exists already, just add the privilege for ADAMS.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE(acl          => 'www.xml',
                                       principal   => 'ADAMS',
                                       is_grant    => true,
                                       privilege   => 'resolve');

END;
/
COMMIT;
```

### Example 3

Assign the ACL www.xml to www-proxy.us.oracle.com so that SCOTT and ADAMS can access www-proxy.us.oracle.com also.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL(acl => 'www.xml',
                                   host => 'www-proxy.us.oracle.com');

END;
/
COMMIT;
```

### Example 4

Unassign the ACL from www.us.oracle.com so that no access to www.us.oracle.com is allowed.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.UNASSIGN_ACL(host => 'www.us.oracle.com');

END;
/
COMMIT;
```

**Example 5**

The [DOMAINS Function](#) in the DBMS\_NETWORK\_ACL\_UTILILITY package returns all the domains to which a host belongs. It can be used in conjunction with the [CHECK\\_PRIVILEGE\\_ACLID Function](#) in this package to determine the privilege assignments affecting a user's permission to access a network host. The function [DOMAIN\\_LEVEL Function](#) in the DBMS\_NETWORK\_ACL\_UTILILITY package returns the level of each domain and can be used to order the ACL assignments by their precedence.

For example, for SCOTT's permission to connect to www.us.oracle.com:

```
SELECT host, lower_port, upper_port, acl,
       DECODE(
          DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'SCOTT', 'connect'),
          1, 'GRANTED', 0, 'DENIED', NULL) privilege
FROM dba_network_acls
WHERE host IN
      (SELECT * FROM
       TABLE(DBMS_NETWORK_ACL_UTILILITY.DOMAINS('www.us.oracle.com')))
ORDER BY DBMS_NETWORK_ACL_UTILILITY.DOMAIN_LEVEL(host) desc, lower_port,
         upper_port;
```

HOST	LOWER_PORT	UPPER_PORT	ACL	PRIVILEGE
www.us.oracle.com	80	80	/sys/acls/www.xml	GRANTED
www.us.oracle.com	3000	3999	/sys/acls/www.xml	GRANTED
www.us.oracle.com			/sys/acls/www.xml	GRANTED
*.oracle.com			/sys/acls/all.xml	
*			/sys/acls/all.xml	

**Example 6**

For example, for SCOTT's permission to do domain name resolution for www.us.oracle.com:

```
SELECT host, acl,
       DECODE(
          DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'SCOTT', 'resolve'),
          1, 'GRANTED', 0, 'DENIED', NULL) privilege
FROM dba_network_acls
WHERE host IN
      (SELECT * FROM
       TABLE(DBMS_NETWORK_ACL_UTILILITY.DOMAINS('www.us.oracle.com'))) and
      lower_port IS NULL AND upper_port IS NULL
ORDER BY DBMS_NETWORK_ACL_UTILILITY.DOMAIN_LEVEL(host) desc;
```

HOST	ACL	PRIVILEGE
www.us.oracle.com	/sys/acls/www.xml	GRANTED
*.oracle.com	/sys/acls/all.xml	
*	/sys/acls/all.xml	

Note that the 'resolve' privilege takes effect only in ACLs assigned without any port range (when lower\_port and upper\_port are NULL). For this reason, the example does not include lower\_port and upper\_port columns in the query.

---

## Summary of DBMS\_NETWORK\_ACL\_ADMIN Subprograms

**Table 94–1 DBMS\_NETWORK\_ACL\_ADMIN Package Subprograms**

Subprogram	Description
<a href="#">ADD_PRIVILEGE Procedure</a> on page 94-6	Adds a privilege to grant or deny the network access to the user in an access control list (ACL)
<a href="#">ASSIGN_ACL Procedure</a> on page 94-8	Assigns an access control list (ACL) to a network host, and optionally specific to a TCP port range
<a href="#">ASSIGN_WALLET_ACL Procedure</a> on page 94-10	Assigns an access control list (ACL) to a wallet
<a href="#">CHECK_PRIVILEGE Function</a> on page 94-11	Checks if a privilege is granted to or denied from the user in an access control list (ACL)
<a href="#">CHECK_PRIVILEGE_ACLID Function</a> on page 94-12	Checks if a privilege is granted to or denied from the user in an ACL by specifying the object ID of the access control list
<a href="#">CREATE_ACL Procedure</a> on page 94-13	Creates an access control list (ACL) with an initial privilege setting
<a href="#">DELETE_PRIVILEGE Procedure</a> on page 94-14	Deletes a privilege in an access control list (ACL)
<a href="#">DROP_ACL Procedure</a> on page 94-15	Drops an access control list (ACL)
<a href="#">UNASSIGN_ACL Procedure</a> on page 94-16	Unassigns the access control list (ACL) currently assigned to a network host
<a href="#">UNASSIGN_WALLET_ACL Procedure</a> on page 94-17	Unassigns the access control list (ACL) currently assigned to a wallet

## ADD\_PRIVILEGE Procedure

This procedure adds a privilege to grant or deny the network access to the user. The access control entry (ACE) is created if it does not exist.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
  acl          IN VARCHAR2,
  principal    IN VARCHAR2,
  is_grant     IN BOOLEAN,
  privilege    IN VARCHAR2,
  position     IN PLS_INTEGER DEFAULT NULL,
  start_date   IN TIMESTAMP WITH TIMESTAMP DEFAULT NULL,
  end_date     IN TIMESTAMP WITH TIMESTAMP DEFAULT NULL );
```

### Parameters

**Table 94–2 ADD\_PRIVILEGE Function Parameters**

Parameter	Description
<code>acl</code>	Name of the ACL. Relative path will be relative to <code>"/sys/acls"</code>
<code>principal</code>	Principal (database user or role) to whom the privilege is granted or denied. Case sensitive.
<code>is_grant</code>	Network privilege to be granted or denied - <code>'connect   resolve'</code> (case sensitive). A database user needs the <code>connect</code> privilege to an external network host computer if he or she is connecting using the <code>UTL_TCP</code> , <code>UTL_HTTP</code> , <code>UTL_SMTP</code> , and <code>UTL_MAIL</code> utility packages. To resolve a host name that was given a host IP address, or the IP address that was given a host name, with the <code>UTL_INADDR</code> package, grant the database user the <code>resolve</code> privilege.
<code>privilege</code>	Network privilege to be granted or denied
<code>position</code>	Position (1-based) of the ACE. If a non-NULL value is given, the privilege will be added in a new ACE at the given position and there should not be another ACE for the principal with the same <code>is_grant</code> (grant or deny). If a NULL value is given, the privilege will be added to the ACE matching the principal and the <code>is_grant</code> if one exists, or to the end of the ACL if the matching ACE does not exist.
<code>start_date</code>	Start date of the access control entry (ACE). When specified, the ACE will be valid only on and after the specified date. The <code>start_date</code> will be ignored if the privilege is added to an existing ACE.
<code>end_date</code>	End date of the access control entry (ACE). When specified, the ACE expires after the specified date. The <code>end_date</code> must be greater than or equal to the <code>start_date</code> . The <code>end_date</code> will be ignored if the privilege is added to an existing ACE.

### Usage Notes

To remove the permission, use the [DELETE\\_PRIVILEGE Procedure](#).

### Examples

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
```



```
acl          => 'us-oracle-com-permissions.xml',  
principal   => 'ST_USERS',  
is_grant    => TRUE,  
privilege   => 'connect')  
END;
```

## ASSIGN\_ACL Procedure

This procedure assigns an access control list (ACL) to a host computer, domain, or IP subnet, and if specified, the TCP port range.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (
    acl          IN VARCHAR2,
    host         IN VARCHAR2,
    lower_port   IN PLS_INTEGER DEFAULT NULL,
    upper_port   IN PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 94–3** *ASSIGN\_ACL Function Parameters*

Parameter	Description
acl	Name of the ACL. Relative path will be relative to <code>"/sys/acls"</code> .
host	Host to which the ACL is to be assigned. The host can be the name or the IP address of the host. A wildcard can be used to specify a domain or a IP subnet. The host or domain name is case-insensitive.
lower_port	Lower bound of a TCP port range if not NULL
upper_port	Upper bound of a TCP port range. If NULL, lower_port is assumed.

### Usage Notes

- Only one ACL can be assigned to any host computer, domain, or IP subnet, and if specified, the TCP port range. When you assign a new access control list to a network target, Oracle Database unassigns the previous access control list that was assigned to the same target. However, Oracle Database does not drop the access control list. You can drop the access control list by using the [DROP\\_ACL Procedure](#). To remove an access control list assignment, use the [UNASSIGN\\_ACL Procedure](#).
- The ACL assigned to a domain takes a lower precedence than the other ACLs assigned sub-domains, which take a lower precedence than the ACLs assigned to the individual hosts. So for a given host, for example, "www.us.oracle.com", the following domains are listed in decreasing precedences:
  - www.us.oracle.com
  - \*.us.oracle.com
  - \*.oracle.com
  - \*.com
  - \*

In the same way, the ACL assigned to an subnet takes a lower precedence than the other ACLs assigned smaller subnets, which take a lower precedence than the ACLs assigned to the individual IP addresses. So for a given IP address, for example, "192.168.0.100", the following subnets are listed in decreasing precedences:

- 192.168.0.100
- 192.168.0.\*
- 192.168.\*
- 192.\*
- \*

- The port range is applicable only to the "connect" privilege assignments in the ACL. The "resolve" privilege assignments in an ACL have effects only when the ACL is assigned to a host without a port range.

For the "connect" privilege assignments, an ACL assigned to the host without a port range takes a lower precedence than other ACLs assigned to the same host with a port range.

- When specifying a TCP port range, both `lower_port` and `upper_port` must not be NULL and `upper_port` must be greater than or equal to `lower_port`. The port range must not overlap with any other port ranges for the same host assigned already.
- To remove the assignment, use [UNASSIGN\\_ACL Procedure](#).

## Examples

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL(
    acl      => 'us-oracle-com-permissions.xml',
    host     => '*.us.oracle.com',
    lower_port => 80);
END;
```

## ASSIGN\_WALLET\_ACL Procedure

This procedure assigns an access control list (ACL) to a wallet.

### Syntax

```
UTL_HTTP.ASSIGN_WALLET_ACL (
    acl          IN  VARCHAR2,
    wallet_path  IN  VARCHAR2);
```

### Parameters

**Table 94–4** *ASSIGN\_WALLET\_ACL Procedure Parameters*

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls"
wallet_path	Directory path of the wallet to which the ACL is to be assigned. The path is case-sensitive and of the format <i>file:directory-path</i> .

### Usage Notes

To remove the assignment, use the [UNASSIGN\\_WALLET\\_ACL Procedure](#).

### Examples

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
    acl          => 'wallet-acl.xml',
    description => 'Wallet ACL',
    principal   => 'SCOTT',
    is_grant    => TRUE,
    privilege   => 'use-client-certificates');

  DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE(
    acl          => 'wallet-acl.xml',
    principal   => 'SCOTT',
    is_grant    => TRUE,
    privilege   => 'use-passwords');

  DBMS_NETWORK_ACL_ADMIN.ASSIGN_WALLET_ACL(
    acl          => 'wallet-acl.xml',
    wallet_path => 'file:/oracle/wallets/test_wallet');
END;
```

## CHECK\_PRIVILEGE Function

This function checks if a privilege is granted to or denied from the user in an ACL.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE (
  acl          IN VARCHAR2,
  user         IN VARCHAR2,
  privilege    IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 94–5** CHECK\_PRIVILEGE Function Parameters

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls".
user	User to check against. If the user is NULL, the invoker is assumed. The username is case-sensitive as in the USERNAME column of the ALL_USERS view.
privilege	Network privilege to check

### Return Values

Returns 1 when the privilege is granted; 0 when the privilege is denied; NULL when the privilege is neither granted or denied.

### Examples

```
SELECT DECODE(
  DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE(
    'us-oracle-com-permissions.xml', 'SCOTT', 'resolve'),
  1, 'GRANTED', 0, 'DENIED', NULL) PRIVILEGE
FROM DUAL;
```

## CHECK\_PRIVILEGE\_ACLID Function

This function checks if a privilege is granted to or denied from the user in an ACL by specifying the object ID of the access control list.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID (  
    aclid          IN RAW,  
    user           IN VARCHAR2 DEFAULT NULL)  
    privilege      IN VARCHAR2,  
    RETURN NUMBER;
```

### Parameters

**Table 94–6** CHECK\_PRIVILEGE\_ACLID Function Parameters

Parameter	Description
aclid	Object ID of the ACL
user	User to check against. If the user is NULL, the invoker is assumed. The username is case-sensitive as in the USERNAME column of the ALL_USERS view.
privilege	Network privilege to check

### Return Values

Returns 1 when the privilege is granted; 0 when the privilege is denied; NULL when the privilege is neither granted or denied.

## CREATE\_ACL Procedure

This procedure creates an access control list (ACL) with an initial privilege setting. An ACL must have at least one privilege setting. The ACL has no access control effect unless it is assigned to the network target.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl           IN VARCHAR2,
  description   IN VARCHAR2,
  principal     IN VARCHAR2,
  is_grant      IN BOOLEAN,
  privilege     IN VARCHAR2,
  start_date    IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
  end_date      IN TIMESTAMP WITH TIMEZONE DEFAULT NULL );
```

### Parameters

**Table 94–7 CREATE\_ACL Procedure Parameters**

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls".
description	Description attribute in the ACL
principal	Principal (database user or role) to whom the privilege is granted or denied. Case sensitive.
is_grant	Privilege is granted or not (denied)
privilege	Network privilege to be granted or denied - 'connect   resolve' (case sensitive). A database user needs the connect privilege to an external network host computer if he or she is connecting using the UTL_TCP, UTL_HTTP, UTL_SMTP, and UTL_MAIL utility packages. To resolve a host name that was given a host IP address, or the IP address that was given a host name, with the UTL_INADDR package, grant the database user the resolve privilege.
start_date	Start date of the access control entry (ACE). When specified, the ACE is valid only on and after the specified date.
end_date	End date of the access control entry (ACE). When specified, the ACE expires after the specified date. The end_date must be greater than or equal to the start_date.

### Usage Notes

To drop the access control list, use the [DROP\\_ACL Procedure](#).

### Examples

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
    acl           => 'us-oracle-com-permissions.xml',
    description   => 'Network permissions for *.us.oracle.com',
    principal     => 'SCOTT',
    is_grant      => TRUE,
    privilege     => 'connect');
END;
```

## DELETE\_PRIVILEGE Procedure

This procedure deletes a privilege in an access control list.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.DELETE_PRIVILEGE (  
    acl          IN VARCHAR2,  
    principal    IN VARCHAR2,  
    is_grant     IN BOOLEAN DEFAULT NULL,  
    privilege    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 94–8** *DELETE\_PRIVILEGE Function Parameters*

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls".
principal	Principal (database user or role) for whom all the ACE will be deleted
is_grant	Privilege is granted or not (denied). If a NULL value is given, the deletion is applicable to both granted or denied privileges.
privilege	Network privilege to be deleted. If a NULL value is given, the deletion is applicable to all privileges.

### Examples

```
BEGIN  
    DBMS_NETWORK_ACL_ADMIN.DELETE_PRIVILEGE (  
        acl          => 'us-oracle-com-permissions.xml',  
        principal    => 'ST_USERS')  
END;
```



## DROP\_ACL Procedure

This procedure drops an access control list (ACL).

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.DROP_ACL (
    acl          IN VARCHAR2);
```

### Parameters

**Table 94–9** *DROP\_ACL Procedure Parameters*

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls".

### Examples

```
BEGIN
    DBMS_NETWORK_ACL_ADMIN.DROP_ACL(
        acl => 'us-oracle-com-permissions.xml');
END;
```

## UNASSIGN\_ACL Procedure

This procedure unassigns the access control list (ACL) currently assigned to a network host.

### Syntax

```
DBMS_NETWORK_ACL_ADMIN.UNASSIGN_ACL (
  acl          IN VARCHAR2 DEFAULT NULL,
  host         IN VARCHAR2 DEFAULT NULL,
  lower_port   IN PLS_INTEGER DEFAULT NULL,
  upper_port   IN PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 94–10 UNASSIGN\_ACL Function Parameters**

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls". If ACL is NULL, any ACL assigned to the host is unassigned.
host	Host from which the ACL is to be removed. The host can be the name or the IP address of the host. A wildcard can be used to specify a domain or a IP subnet. The host or domain name is case-insensitive. If host is NULL, the ACL will be unassigned from any host. If both host and acl are NULL, all ACLs assigned to any hosts are unassigned.
lower_port	Lower bound of a TCP port range if not NULL
upper_port	Upper bound of a TCP port range. If NULL, lower_port is assumed.

### Examples

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.UNASSIGN_ACL(
    host         => '*.us.oracle.com',
    lower_port   => 80);
END;
```

## UNASSIGN\_WALLET\_ACL Procedure

This procedure unassigns the access control list (ACL) currently assigned to a wallet.

### Syntax

```
UTL_HTTP.UNASSIGN_WALLET_ACL (
  acl          IN  VARCHAR2 DEFAULT NULL,
  wallet_path  IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 94–11 UNASSIGN\_WALLET\_ACL Procedure Parameters**

Parameter	Description
acl	Name of the ACL. Relative path will be relative to "/sys/acls". If acl is NULL, any ACL assigned to the wallet is unassigned.
wallet_path	Directory path of the wallet to which the ACL is assigned. The path is case-sensitive and of the format <i>file:directory-path</i> . If both acl and wallet_path are NULL, all ACLs assigned to any wallets are unassigned.

### Examples

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.UNASSIGN_WALLET_ACL(
    acl          => 'wallet-acl.xml',
    wallet_path => 'file:/oracle/wallets/test_wallet');
END;
```



---

---

## DBMS\_NETWORK\_ACL\_UTILITY

The `DBMS_NETWORK_ACL_UTILITY` package provides the utility functions to facilitate the evaluation of access control list (ACL) assignments governing TCP connections to network hosts.

**See Also:** For more information, see ""Managing Fine-grained Access to External Network Services"" in *Oracle Database Security Guide*

The chapter contains the following topics:

- [Using DBMS\\_NETWORK\\_ACL\\_UTILITY](#)
  - Examples
- [Summary of DBMS\\_NETWORK\\_ACL\\_UTILITY Subprograms](#)

---

## Using DBMS\_NETWORK\_ACL\_UTILITY

- [Examples](#)

## Examples

The [CONTAINS\\_HOST Function](#) in this package indicates if a domain or subnet contains a given host or IP address. It can be used in conjunction with the [CHECK\\_PRIVILEGE\\_ACLID Function](#) in the [DBMS\\_NETWORK\\_ACL\\_ADMIN](#) package to determine the privilege assignments affecting a user's permission to access a network host. The return value of the [CONTAINS\\_HOST Function](#) in can also be used to order the ACL assignments by their precedence.

### Example 1

For example, for SCOTT's permission to connect to www.hr.example.com:

```
SELECT host, lower_port, upper_port, acl,
       DECODE(
         DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'SCOTT', 'connect'),
         1, 'GRANTED', 0, 'DENIED', NULL) privilege
FROM (SELECT host, acl, aclid, lower_port, upper_port,
            DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('www.hr.example.com', host)
            precedence
      FROM dba_network_acls)
WHERE precedence > 0
ORDER BY precedence DESC, lower_port nulls LAST;
```

HOST	LOWER_PORT	UPPER_PORT	ACL	PRIVILEGE
www.hr.example.com	80	80	/sys/acls/www.xml	GRANTED
www.hr.example.com	3000	3999	/sys/acls/www.xml	GRANTED
www.hr.example.com			/sys/acls/www.xml	GRANTED
*.hr.example.com			/sys/acls/all.xml	
*.example.com			/sys/acls/all.xml	

### Example 2

For example, for SCOTT's permission to do domain name resolution for www.hr.example.com:

```
SELECT host, acl,
       DECODE(
         DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'SCOTT', 'resolve'),
         1, 'GRANTED', 0, 'DENIED', null) privilege
FROM (SELECT host, acl, aclid,
            DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('www.hr.example.com', host)
            precedence
      FROM dba_network_acls
      WHERE lower_port IS NULL AND upper_port IS NULL)
WHERE precedence > 0
ORDER BY precedence DESC;
```

HOST	ACL	PRIVILEGE
www.hr.example.com	/sys/acls/hr-www.xml	GRANTED
*.hr.example.com	/sys/acls/hr-domain.xml	
*.example.com	/sys/acls/corp-domain.xml	

Note that the "resolve" privilege takes effect only in ACLs assigned without any port range (when `lower_port` and `upper_port` are NULL). For this reason, the example does not include `lower_port` and `upper_port` columns in the query.

## Summary of DBMS\_NETWORK\_ACL\_UTILITY Subprograms

**Table 95–1 DBMS\_NETWORK\_ACL\_UTILITY Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CONTAINS_HOST Function</a> on page 95-5	Determines if the given host is equal to or contained in the given host, domain, or subnet
<a href="#">DOMAIN_LEVEL Function</a> on page 95-6	Returns the domain level of the given host name, domain, or subnet
<a href="#">DOMAINS Function</a> on page 95-7	For a given host, this function returns the domains whose ACL assigned is used to determine if a user has the privilege to access the given host or not.
<a href="#">EQUALS_HOST Function</a> on page 95-8	Determines if the two given hosts, domains, or subnets are equal



## CONTAINS\_HOST Function

This function determines if the given host is equal to or contained in the given host, domain, or subnet. It handles different representation of the same IP address or subnet. For example, an IPv4-mapped IPv6 address is considered equal to the IPv4-native address it represents. It does not perform domain name resolution when evaluating the host or domain.

### Syntax

```
DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST (
    host      IN      VARCHAR2,
    domain    IN      VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 95–2 CONTAINS\_HOST Function Parameters**

Parameter	Description
host	Network host
domain	Network host, domain, or subnet

### Return Values

Returns a non-NULL value if the given host is equal to or contained in the related host, domain, or subnet:

- If `domain` is a hostname, returns the level of its domain + 1
- If `domain` is a domain name, returns the domain level
- If `domain` is an IP address or subnet, return the number of significant address bits of the IP address or subnet
- If `domain` is the wildcard "\*", returns 0

The non-NULL value returned indicates the precedence of the domain or subnet for ACL assignment. The higher the value, the higher is the precedence. NULL will be returned if the host is not equal to or contained in the given host, domain or subnet.

### Examples

```
SELECT host, acl, precedence
   FROM (select host, acl,
                DBMS_NETWORK_ACL_UTILITY.CONTAINS_HOST('192.0.2.3', host)
                precedence
          FROM dba_network_acls)
 WHERE precedence > 0
 ORDER BY precedence DESC;
```

HOST	ACL	PRECEDENCE
192.0.2.3	/sys/acls/hr-www.xml	32
::ffff:192.0.2.0/120	/sys/acls/hr-domain.xml	24
::ffff:192.0.0.0/104	/sys/acls/corp-domain.xml	8

## DOMAIN\_LEVEL Function

This function returns the domain level of the given host name, domain, or subnet.

### Syntax

```
DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL (
    host IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 95–3 DOMAIN\_LEVEL Function Parameters**

Parameter	Description
host	Network host, domain, or subnet

### Return Values

The domain level of the given host, domain, or subnet.

### Usage Notes

Note that this function cannot handle IPv6 addresses and subnets, and subnets in CIDR notation.

### Examples

```
SELECT host, acl, domain_level
   FROM (select host, acl,
                DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) domain_level
         FROM dba_network_acls)
 order by domain_level desc;
```

HOST	ACL	DOMAIN_LEVEL
www.hr.example.com	/sys/acls/hr-www.xml	4
*.hr.example.com	/sys/acls/hr-domain.xml	3
*.example.com	/sys/acls/corp-domain.xml	2

## DOMAINS Function

For a given host, this function returns the domains whose ACL assigned determines if a user has the privilege to access the given host or not. When the IP address of the host is given, return the subnets instead.

### Syntax

```
DBMS_NETWORK_ACL_UTILITY.DOMAINS (
    host IN VARCHAR2)
RETURN DOMAIN_TABLE PIPELINED;
```

### Parameters

**Table 95–4** *DOMAINS Function Parameters*

Parameter	Description
host	Network host

### Return Values

The domains or subnets for the given host.

### Usage Notes

Note that this function cannot handle IPv6 addresses. Nor can it generate subnets of arbitrary number of prefix bits for an IPv4 address.

### Examples

```
select * from table(dbms_network_acl_utility.domains('www.hr.example.com'));
```

```
DOMAINS
-----
www.hr.example.com
*.hr.example.com
*.example.com
*.com
*
```

## EQUALS\_HOST Function

This function determines if the two given hosts, domains, or subnets are equal. It handles different representation of the same IP address or subnet. For example, an IPv4-mapped IPv6 address is considered equal to the IPv4- native address it represents. It does not perform domain name resolution when comparing the two hosts or domains.

### Syntax

```
DBMS_NETWORK_ACL_UTILITY.EQUALS_HOST (
    host1     IN     VARCHAR2,
    host2     IN     VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 95–5** EQUALS\_HOST Function Parameters

Parameter	Description
host1	Network host, domain, or subnet to compare
host2	Network host, domain, or subnet to compare

### Return Values

1 if the two hosts, domains, or subnets are equal. 0 otherwise.

### Examples

```
SELECT host, acl
   FROM dba_network_acls
  WHERE DBMS_NETWORK_ACL_UTILITY.EQUALS_HOST('192.0.2.*', host) = 1;
```

```
HOST          ACL
-----
::ffff:192.0.2.0/120  /sys/acls/hr-domain.xml
```

---

---

## DBMS\_OBFUSCATION\_TOOLKIT

DBMS\_OBFUSCATION\_TOOLKIT enables an application to encrypt data using either the Data Encryption Standard (DES) or the Triple DES algorithms.

---

---

**Note:** DBMS\_OBFUSCATION\_TOOLKIT is deprecated. DBMS\_CRYPTO is intended to replace the DBMS\_OBFUSCATION\_TOOLKIT, providing greater ease of use and support for a range of algorithms to accommodate new and existing systems. See [Chapter 39, "DBMS\\_CRYPTO"](#) for more information.

---

---

This chapter contains the following topics:

- [Using DBMS\\_OBFUSCATION\\_TOOLKIT](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_OBFUSCATION Subprograms](#)

---

## Using DBMS\_OBFUSCATION\_TOOLKIT

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)

## Overview

The Data Encryption Standard (DES), also known as the Data Encryption Algorithm (DEA) by the American National Standards Institute (ANSI) and DEA-1 by the International Standards Organization (ISO), has been a worldwide encryption standard for over 20 years. The banking industry has also adopted DES-based standards for transactions between private financial institutions, and between financial institutions and private individuals. DES will eventually be replaced by a new Advanced Encryption Standard (AES).

DES is a symmetric key cipher; that is, the same key is used to encrypt data as well as decrypt data. DES encrypts data in 64-bit blocks using a 56-bit key. The DES algorithm ignores 8 bits of the 64-bit key that is supplied; however, you must supply a 64-bit key to the algorithm.

Triple DES (3DES) is a far stronger cipher than DES; the resulting ciphertext (encrypted data) is much harder to break using an exhaustive search:  $2^{112}$  or  $2^{168}$  attempts instead of  $2^{56}$  attempts. Triple DES is also not as vulnerable to certain types of cryptanalysis as is DES.

## Security Model

Oracle installs this package in the `SYS` schema. You can then grant package access to existing users and roles as needed. The package also grants access to the `PUBLIC` role so no explicit grant needs to be done.



## Operational Notes

- [Key Management](#)
- [Storing the Key in the Database](#)
- [Storing the Key in the Operating System](#)
- [User-Supplied Keys](#)

### Key Management

Key management, including both generation and secure storage of cryptographic keys, is one of the most important aspects of encryption. If keys are poorly chosen or stored improperly, then it is far easier for a malefactor to break the encryption. Rather than using an exhaustive key search attack (that is, cycling through all the possible keys in hopes of finding the correct decryption key), cryptanalysts typically seek weaknesses in the choice of keys, or the way in which keys are stored.

Key generation is an important aspect of encryption. Typically, keys are generated automatically through a random-number generator. Provided that the random number generation is cryptographically secure, this can be an acceptable form of key generation. However, if random numbers are not cryptographically secure, but have elements of predictability, the security of the encryption may be easily compromised.

The DBMS\_OBFUSCATION\_TOOLKIT package includes tools for generating random material that can be used for encryption keys, but it does not provide a mechanism for maintaining them. Care must be taken by the application developer to ensure the secure generation and storage of encryption keys used with this package.

Furthermore, the encryption and decryption done by the DBMS\_OBFUSCATION\_TOOLKIT takes place on the server, not the client. If the key is passed over the connection between the client and the server, the connection must be protected by using network encryption. Otherwise, the key is vulnerable to capture over the wire. See *Oracle Database Advanced Security Administrator's Guide* for information about configuring and using network encryption for Oracle Net.

Key storage is one of the most important, yet difficult aspects of encryption and one of the hardest to manage properly. To recover data encrypted with a symmetric key, the key must be accessible to the application or user seeking to decrypt data. The key needs to be easy enough to retrieve that users can access encrypted data when they need to without significant performance degradation. The key also needs to be secure enough that it is not easily recoverable by unauthorized users trying to access encrypted data that they are not supposed to see.

The three options available are:

- Store the key in the database
- Store the key in the operating system
- Have the user manage the key

### Storing the Key in the Database

Storing the keys in the database cannot always provide bullet-proof security if you are trying to protect data against the DBA accessing encrypted data (since an all-privileged DBA can access tables containing encryption keys), but it can provide security against the casual snooper, or against someone compromising the database files on the operating system. Furthermore, the security you can obtain by storing keys in the database does not have to be bullet-proof in order to be extremely useful.

For example, suppose you want to encrypt an employee's social security number, one of the columns in table EMP. You could encrypt each employee's SSN using a key which is stored in a separate column in EMP. However, anyone with SELECT access on the EMP table could retrieve the encryption key and decrypt the matching social security number. Alternatively, you could store the encryption keys in another table, and use a package to retrieve the correct key for the encrypted data item, based on a primary key-foreign key relationship between the tables.

You can envelope both the DBMS\_OBFUSCATION\_TOOLKIT package and the procedure to retrieve the encryption keys supplied to the package. Furthermore, the encryption key itself could be transformed in some way (for example, XORed with the foreign key to the EMP table) so that the key itself is not stored in easily recoverable form.

Oracle recommends using the wrap utility of PL/SQL to obfuscate the code within a PL/SQL package itself that does the encryption. That prevents people from breaking the encryption by looking at the PL/SQL code that handles keys, calls encrypting routines, and so on. In other words, use the wrap utility to obfuscate the PL/SQL packages themselves. This scheme is secure enough to prevent users with SELECT access to EMP from reading unencrypted sensitive data, and a DBA from easily retrieving encryption keys and using them to decrypt data in the EMP table. It can be made more secure by changing encryption keys regularly, or having a better key storage algorithm (so the keys themselves are encrypted, for example).

### **Storing the Key in the Operating System**

Storing keys in a flat file in the operating system is another option. You can make callouts from PL/SQL, which you can use to retrieve encryption keys. If you store keys in a file and make callouts to retrieve the keys, the security of your encrypted data is only as secure as the protection of the key file on the operating system. Of course, a user retrieving keys from the operating system would have to be able to either access the Oracle database files (to decrypt encrypted data), or be able to gain access to the table in which the encrypted data is stored as a legitimate user.

### **User-Supplied Keys**

If you ask a user to supply the key, it is crucial that you use network encryption, such as that provided by Oracle Advanced Security, so the key is not passed from client to server in the clear. The user must remember the key, or your data is not recoverable.

---

## Summary of DBMS\_OBFUSCATION Subprograms

**Table 96–1 DBMS\_OBFUSCATION Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DES3DECRYPT Procedures and Functions</a> on page 96-8	Generates the decrypted form of the input data
<a href="#">DES3ENCRYPT Procedures and Functions</a> on page 96-10	Generates the encrypted form of the input data by passing it through the Triple DES encryption algorithm
<a href="#">DES3GETKEY Procedures and Functions</a> on page 96-8	Takes a random value and uses it to generate an encryption key, using Triple DES
<a href="#">DESDECRYPT Procedures and Functions</a> on page 96-13	Generates the decrypted form of the input data
<a href="#">DESENCRYPT Procedures and Functions</a> on page 96-15	Generates the encrypted form of the input data
<a href="#">DESGETKEY Procedures and Functions</a> on page 96-17	Takes a random value and uses it to generate an encryption key
<a href="#">MD5 Procedures and Functions</a> on page 96-18	Generates MD5 hashes of data

## DES3DECRYPT Procedures and Functions

These subprograms generate the decrypted form of the input data.

For a discussion of the initialization vector that you can use with this procedure, see the section, "[DES3ENCRYPT Procedures and Functions](#)" on page 96-10.

### Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT (
  input          IN   RAW,
  key            IN   RAW,
  decrypted_data OUT  RAW,
  which         IN   PLS_INTEGER DEFAULT TwoKeyMode
  iv           IN   RAW          DEFAULT NULL);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT (
  input_string  IN   VARCHAR2,
  key_string    IN   VARCHAR2,
  decrypted_string OUT VARCHAR2,
  which        IN   PLS_INTEGER DEFAULT TwoKeyMode
  iv_string    IN   VARCHAR2   DEFAULT NULL);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT (
  input      IN RAW,
  key       IN RAW,
  which     IN PLS_INTEGER DEFAULT TwoKeyMode
  iv      IN RAW          DEFAULT NULL)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT (
  input_string IN VARCHAR2,
  key_string   IN VARCHAR2,
  which       IN PLS_INTEGER DEFAULT TwoKeyMode
  iv_string   IN VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 96–2** *DES3DECRYPT Parameters for Raw Data*

Parameter	Description
input	Data to be decrypted
key	Decryption key
decrypted_data	Decrypted data
which	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.
iv	Initialization vector
input_string	String to be decrypted
key_string	Decryption key string
decrypted_string	Decrypted string
iv_string	Initialization vector

## Usage Notes

If the input data or key given to the DES3DECRYPT procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit."

If the input data given to the DES3DECRYPT procedure is not a multiple of 8 bytes, the procedure raises the error ORA-28232 "Invalid input size for Obfuscation toolkit." ORA-28233 is NOT applicable for the DES3DECRYPT function.

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

If an incorrect value is specified for the WHICH parameter, ORA-28236 "Invalid Triple DES mode" is generated. Only the values 0 (TwoKeyMode) and 1 (ThreeKeyMode) are valid.

## Restrictions

You must supply a single key of either 128 bits for a 2-key implementation (of which only 112 are used), or a single key of 192 bits for a 3-key implementation (of which 168 bits are used). Oracle automatically truncates the supplied key into 56-bit lengths for decryption. This key length is fixed and cannot be altered.

---

---

**Note:** Both the key length limitation and the prevention of multiple encryption passes are requirements of U.S. regulations governing the export of cryptographic products.

---

---

## DES3ENCRYPT Procedures and Functions

These subprograms generate the encrypted form of the input data by passing it through the Triple DES (3DES) encryption algorithm.

Oracle's implementation of 3DES supports either a 2-key or 3-key implementation, in outer cipher-block-chaining (CBC) mode.

### Syntax

```

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt (
  input          IN      RAW,
  key            IN      RAW,
  encrypted_data OUT     RAW,
  which          IN      PLS_INTEGER DEFAULT TwoKeyMode
  iv            IN      RAW          DEFAULT NULL);

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt (
  input_string   IN      VARCHAR2,
  key_string     IN      VARCHAR2,
  encrypted_string OUT   VARCHAR2,
  which          IN      PLS_INTEGER DEFAULT TwoKeyMode
  iv_string      IN      VARCHAR2   DEFAULT NULL);

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt (
  input          IN RAW,
  key            IN RAW,
  which          IN PLS_INTEGER DEFAULT TwoKeyMode
  iv            IN RAW          DEFAULT NULL)
RETURN RAW;

DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt (
  input_string IN VARCHAR2,
  key_string   IN VARCHAR2,
  which        IN PLS_INTEGER DEFAULT TwoKeyMode
  iv_string    IN VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

```

### Parameters

**Table 96–3** *DES3ENCRYPT Parameters Procedure and Function*

Parameter	Description
input	Data to be encrypted.
key	Encryption key.
encrypted_data	Encrypted data.
which	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.
iv	Initialization vector.
input_string	String to be encrypted.
key_string	Encryption key string.
encrypted_string	Encrypted string.
iv_string	Initialization vector.

## Usage Notes

If you are using Oracle's 3DES interface with a 2-key implementation, you must supply a single key of 128 bits as an argument to the `DES3ENCRYPT` procedure. With a 3-key implementation, you must supply a single key of 192 bits. Oracle then breaks the supplied key into two 64-bit keys. As with DES, the 3DES algorithm throws away 8 bits of each derived key. However, you must supply a single 128-bit key for the 2-key 3DES implementation or a single 192-bit key for the 3-key 3DES implementation; otherwise the package will raise an error. The `DES3ENCRYPT` procedure uses the 2-key implementation by default.

You also have the option of providing an *initialization vector* (IV) with the `DES3ENCRYPT` procedure. An IV is a block of random data prepended to the data you intend to encrypt. The IV has no meaning. It is there to make each message unique. Prepending an IV to your input data avoids starting encrypted blocks of data with common header information, which may give cryptanalysts information they can use to decrypt your data.

If the input data or key given to the PL/SQL `DES3ENCRYPT` procedure is empty, then the procedure raises the error `ORA-28231 "Invalid input to Obfuscation toolkit."`

If the input data given to the `DES3ENCRYPT` procedure is not a multiple of 8 bytes, the procedure raises the error `ORA-28232 "Invalid input size for Obfuscation toolkit."`

If you try to double encrypt data using the `DES3ENCRYPT` procedure, then the procedure raises the error `ORA-28233 "Double encryption not supported."`

If the key length is missing or is less than 8 bytes, then the procedure raises the error `ORA-28234 "Key length too short."` Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

If an incorrect value is specified for the `which` parameter, `ORA-28236 "Invalid Triple DES mode"` is generated. Only the values 0 (`TwoKeyMode`) and 1 (`ThreeKeyMode`) are valid.

## Restrictions

The `DES3ENCRYPT` procedure has two restrictions. The first is that the DES key length for encryption is fixed at 128 bits (for 2-key DES) or 192 bits (for 3-key DES); you cannot alter these key lengths.

The second is that you cannot execute multiple passes of encryption using 3DES. (Note: the 3DES algorithm itself encrypts data multiple times; however, you cannot call the `DES3ENCRYPT` function itself more than once to encrypt the same data using 3DES.)

---

---

**Note:** Both the key length limitation and the prevention of multiple encryption passes are requirements of U.S. regulations governing the export of cryptographic products.

---

---

## DES3GETKEY Procedures and Functions

These subprograms take a random value and uses it to generate an encryption key. For Triple DES, you specify the mode so that the returned key has the proper length.

### Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
  which      IN   PLS_INTEGER DEFAULT TwoKeyMode,
  seed       IN   RAW,
  key        OUT  RAW);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
  which      IN   PLS_INTEGER DEFAULT TwoKeyMode,
  seed_string IN  VARCHAR2,
  key        OUT  VARCHAR2);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
  which IN PLS_INTEGER DEFAULT TwoKeyMode,
  seed  IN RAW)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
  which      IN   PLS_INTEGER DEFAULT TwoKeyMode,
  seed_string IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 96–4** *DES3GETKEY Procedure and Function Parameters*

Parameter	Description
which	If = 0, (default), then TwoKeyMode is used. If = 1, then ThreeKeyMode is used.
seed	A value at least 80 characters long.
key	Encryption key.
seed_string	A value at least 80 characters long.
key	Encryption key.



## DESDECRYPT Procedures and Functions

These subprograms generate the decrypted form of the input data.

### Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DESDecrypt (
  input          IN  RAW,
  key            IN  RAW,
  decrypted_data OUT RAW);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESDecrypt (
  input_string   IN  VARCHAR2,
  key_string     IN  VARCHAR2,
  decrypted_string OUT VARCHAR2);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESDecrypt (
  input          IN  RAW,
  key            IN  RAW)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DESDecrypt (
  input_string   IN  VARCHAR2,
  key_string     IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 96–5** *DESDECRYPT Procedure and Function Parameters*

Parameter	Description
input	Data to be decrypted.
key	Decryption key.
decrypted_data	Decrypted data.
input_string	String to be decrypted.
key_string	Decryption key string.
decrypted_string	Decrypted string.

### Usage Notes

If the input data or key given to the PL/SQL DESDECRYPT function is empty, then Oracle raises ORA error 28231 "Invalid input to Obfuscation toolkit."

If the input data given to the DESDECRYPT function is not a multiple of 8 bytes, Oracle raises ORA error 28232 "Invalid input size for Obfuscation toolkit."

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

---

**Note:** ORA-28233 is not applicable to the DESDECRYPT function.

---

## Restrictions

The DES key length for encryption is fixed at 64 bits (of which 56 bits are used); you cannot alter this key length.

---

---

**Note:** The key length limitation is a requirement of U.S. regulations governing the export of cryptographic products.

---

---

## DESENCRYPT Procedures and Functions

These subprograms generate the encrypted form of the input data.

### Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DESEncrypt (
  input          IN   RAW,
  key            IN   RAW,
  encrypted_data OUT  RAW);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESEncrypt (
  input_string   IN   VARCHAR2,
  key_string     IN   VARCHAR2,
  encrypted_string OUT VARCHAR2);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESEncrypt (
  input          IN RAW,
  key            IN RAW)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DESEncrypt (
  input_string IN VARCHAR2,
  key_string  IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 96–6** DESENCRYPT Procedure and Function Parameters

Parameter	Description
input	Data to be encrypted.
key	Encryption key.
encrypted_data	Encrypted data.
input_string	String to be encrypted.
key_string	Encryption key string.
encrypted_string	Encrypted string.

### Usage Notes

The DES algorithm encrypts data in 64-bit blocks using a 56-bit key. The DES algorithm throws away 8 bits of the supplied key (the particular bits which are thrown away is beyond the scope of this documentation). However, when using the algorithm, you must supply a 64-bit key or the package will raise an error.

If the input data or key given to the PL/SQL DESEncrypt procedure is empty, then the procedure raises the error ORA-28231 "Invalid input to Obfuscation toolkit".

If the input data given to the DESENCRYPT procedure is not a multiple of 8 bytes, the procedure raises the error ORA-28232 "Invalid input size for Obfuscation toolkit."

If you try to double-encrypt data using the DESENCRYPT procedure, then the procedure raises the error ORA-28233 "Double encryption not supported."

If the key length is missing or is less than 8 bytes, then the procedure raises the error ORA-28234 "Key length too short." Note that if larger keys are used, extra bytes are ignored. So a 9-byte key will not generate an exception.

## Restrictions

The DESENCRYPT procedure has the following restrictions:

- The DES key length for encryption is fixed at 56 bits; you cannot alter this key length.
- You cannot execute multiple passes of encryption. That is, you cannot re-encrypt previously encrypted data by calling the function twice.

---

---

**Note:** Both the key length limitation and the prevention of multiple encryption passes are requirements of U.S. regulations governing the export of cryptographic products.

---

---

## DESGETKEY Procedures and Functions

These subprograms take a random value and use it to generate an encryption key.

### Syntax

```
DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
  seed      IN  RAW,
  key       OUT RAW);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
  seed_string IN  VARCHAR2,
  key         OUT  VARCHAR2);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
  seed IN RAW)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DESGetKey(
  seed_string IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 96–7** *DESGETKEY Procedure and Function Parameters*

Parameter	Description
seed	A value at least 80 characters long.
key	Encryption key.
seed_string	A value at least 80 characters long.
key	Encryption key.

## MD5 Procedures and Functions

These subprograms generate MD5 hashes of data. The MD5 algorithm ensures data integrity by generating a 128-bit cryptographic message digest value from given data.

### Syntax

```
DBMS_OBFUSCATION_TOOLKIT.MD5 (
  input          IN  RAW,
  checksum       OUT raw_checksum);

DBMS_OBFUSCATION_TOOLKIT.MD5 (
  input_string   IN  VARCHAR2,
  checksum_string OUT varchar2_checksum);

DBMS_OBFUSCATION_TOOLKIT.MD5 (
  input          IN RAW)
RETURN raw_checksum;

DBMS_OBFUSCATION_TOOLKIT.MD5 (
  input_string IN VARCHAR2)
RETURN varchar2_checksum;
```

### Parameters

**Table 96–8 MD5 Procedure and Function Parameters**

Parameter Name	Description
input	Data to be hashed
checksum	128-bit cryptographic message digest
input_string	String to be hashed
checksum_string	128-bit cryptographic message digest

DBMS\_ODCI package contains a single user function related to the use of Data Cartridges.

**See Also:**

- *Oracle Database Data Cartridge Developer's Guide*

This chapter contains the following topic:

- [Summary of DBMS\\_ODCI Subprograms](#)

---

## Summary of DBMS\_ODCI Subprograms

**Table 97–1 DBMS\_ODCI Package Subprograms**

Subprogram	Description
<a href="#">ESTIMATE_CPU_UNITS Function</a> on page 97-3	Returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds)



## ESTIMATE\_CPU\_UNITS Function

This function returns the approximate number of CPU instructions (in thousands) corresponding to a specified time interval (in seconds). This information can be used to associate the CPU cost with a user-defined function for the extensible optimizer.

The function takes as input the elapsed time of the user function, measures CPU units by multiplying the elapsed time by the processor speed of the machine, and returns the approximate number of CPU instructions that should be associated with the user function. For a multiprocessor machine, `ESTIMATE_CPU_UNITS` considers the speed of a single processor.

### Syntax

```
DBMS_ODCI.estimate_cpu_units (
    elapsed_time    NUMBER)
RETURN NUMBER;
```

### Parameters

Parameter	Description
<code>elapsed_time</code>	The elapsed time in seconds that it takes to execute a function.

### Usage Notes

When associating CPU cost with a user-defined function, use the full number of CPU units rather than the number of *thousands* of CPU units returned by `ESTIMATE_CPU_UNITS`; multiply the number returned by `ESTIMATE_CPU_UNITS` by 1,000.



---

---

## DBMS\_OFFLINE\_OG

The DBMS\_OFFLINE\_OG package contains the public interface for offline instantiation of master groups.

This chapter contains the following topics:

- [Documentation of DBMS\\_OFFLINE\\_OG](#)

---

## Documentation of DBMS\_OFFLINE\_OG

For a complete description of this package within the context of Replication, see DBMS\_OFFLINE\_OG in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_OUTLN

The DBMS\_OUTLN package, synonymous with OUTLN\_PKG, contains the functional interface for subprograms associated with the management of stored outlines.

**See Also:** For more information about using the DBMS\_OUTLN package, see "Using Plan Stability" in *Oracle Database Performance Tuning Guide*.

---

---

**Note:** Stored outlines will be desupported in a future release in favor of SQL plan management. In Oracle Database 11g Release 1 (11.1), stored outlines continue to function as in past releases. However, Oracle strongly recommends that you use SQL plan management for new applications. SQL plan management creates SQL plan baselines, which offer superior SQL performance and stability compared with stored outlines.

If you have existing stored outlines, please consider migrating them to SQL plan baselines by using the [LOAD\\_PLANS\\_FROM\\_CURSOR\\_CACHE Functions](#) or [LOAD\\_PLANS\\_FROM\\_SQLSET Function](#) of the DBMS\_SPM package. When the migration is complete, you should disable or remove the stored outlines.

---

---

This chapter contains the following topics:

- [Using DBMS\\_OUTLN](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_OUTLN Subprograms](#)

## Using DBMS\_OUTLN

- [Overview](#)
- [Security Model](#)

## Overview

A stored outline is the stored data that pertains to an execution plan for a given SQL statement. It enables the optimizer to repeatedly re-create execution plans that are equivalent to the plan originally generated along with the outline. The data stored in an outline consists, in part, of a set of hints that are used to achieve plan stability.

Stored outlines will be de-supported in a future release in favor of SQL plan management. As of 11g R1, stored outlines continue to function as in past releases, but Oracle strongly recommends that you use SQL plan management for new applications. SQL plan management creates SQL plan baselines, which offer superior SQL performance and stability compared with stored outlines.

If you have existing stored outlines, please consider migrating them to SQL plan baselines by using the [LOAD\\_PLANS\\_FROM\\_CURSOR\\_CACHE Functions](#) or the [LOAD\\_PLANS\\_FROM\\_SQLSET Function](#) of the [DBMS\\_SPM](#) package. When the migration is complete, you should disable or remove the stored outlines.

## Security Model

DBMS\_OUTLN contains management procedures that should be available to appropriate users only. EXECUTE privilege is not extended to the general user community unless the DBA explicitly does so.

PL/SQL functions that are available for outline management purposes can be executed only by users with EXECUTE privilege on the procedure (or package).



---

## Summary of DBMS\_OUTLN Subprograms

**Table 99-1 DBMS\_OUTLN Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CLEAR_USED Procedure</a> on page 99-6	Clears the outline 'used' flag
<a href="#">CREATE_OUTLINE Procedure</a> on page 99-7	Generates outlines from the shared cursor identified by hash value and child number
<a href="#">DROP_BY_CAT Procedure</a> on page 99-8	Drops outlines that belong to a specified category
<a href="#">DROP_UNUSED Procedure</a> on page 99-9	Drops outlines that have never been applied in the compilation of a SQL statement
<a href="#">EXACT_TEXT_SIGNATURES Procedure</a> on page 99-10	Updates outline signatures to those that compute based on exact text matching
<a href="#">UPDATE_BY_CAT Procedure</a> on page 99-11	Changes the category of outlines in one category to a new category
<a href="#">UPDATE_SIGNATURES Procedure</a> on page 99-12	Updates outline signatures to the current version's signature

## CLEAR\_USED Procedure

This procedure clears the outline 'used' flag.

### Syntax

```
DBMS_OUTLN.CLEAR_USED (  
    name      IN      VARCHAR2);
```

### Parameters

**Table 99–2** *CLEAR\_USED Procedure Parameters*

Parameter	Description
name	Name of the outline.

## CREATE\_OUTLINE Procedure

This procedure generates an outline by reparsing the SQL statement from the shared cursor identified by hash value and child number.

### Syntax

```
DBMS_OUTLN.CREATE_OUTLINE (  
    hash_value     IN NUMBER,  
    child_number  IN NUMBER,  
    category       IN VARCHAR2 DEFAULT 'DEFAULT');
```

### Parameters

**Table 99-3 CREATE\_OUTLINE Procedure Parameters**

Parameter	Description
hash_value	Hash value identifying the target shared cursor.
child_number	Child number of the target shared cursor.
category	Category in which to create outline (optional).

## DROP\_BY\_CAT Procedure

This procedure drops outlines that belong to a particular category. While outlines are put into the `DEFAULT` category unless otherwise specified, users have the option of grouping their outlines into groups called categories.

### Syntax

```
DBMS_OUTLN.DROP_BY_CAT (  
    cat VARCHAR2);
```

### Parameters

**Table 99–4** *DROP\_BY\_CAT Procedure Parameters*

Parameter	Description
cat	Category of outlines to drop.

### Usage Notes

This procedure purges a category of outlines in a single call.

### Examples

This example drops all outlines in the `DEFAULT` category:

```
DBMS_OUTLN.DROP_BY_CAT('DEFAULT');
```

## DROP\_UNUSED Procedure

This procedure drops outlines that have never been applied in the compilation of a SQL statement.

### Syntax

```
DBMS_OUTLN.DROP_UNUSED;
```

### Usage Notes

You can use `DROP_UNUSED` for outlines generated by an application for one-time use SQL statements created as a result of dynamic SQL. These outlines are never used and take up valuable disk space.

## **EXACT\_TEXT\_SIGNATURES Procedure**

This procedure updates outline signatures to those that compute based on exact text matching.

### **Syntax**

```
DBMS. OUTLN. EXACT_TEXT_SIGNATURES ;
```

### **Usage Notes**

This procedure is relevant only for downgrading an outline to 8.1.6 or earlier.

## UPDATE\_BY\_CAT Procedure

This procedure changes the category of all outlines in one category to a new category.

### Syntax

```
DBMS.OUTLN.UPDATE_BY_CAT (  
  oldcat   VARCHAR2 default 'DEFAULT',  
  newcat   VARCHAR2 default 'DEFAULT');
```

### Parameters

**Table 99-5** *UPDATE\_BY\_CAT Procedure Parameters*

Parameter	Description
oldcat	The current category of outlines.
newcat	The new category of outlines.

## UPDATE\_SIGNATURES Procedure

This procedure updates outline signatures to the current version's signature.

### Syntax

```
DBMS. OUTLN. UPDATE_SIGNATURES;
```

### Usage Notes

You should execute this procedure if you have imported outlines generated in an earlier release to ensure that the signatures are compatible with the current release's computation algorithm.



The DBMS\_OUTPUT package enables you to send messages from stored procedures, packages, and triggers. The package is especially useful for displaying PL/SQL debugging information.

This chapter contains the following topics:

- [Using DBMS\\_OUTPUT](#)
  - Overview
  - Security Model
  - Operational Notes
  - Exceptions
  - Rules and Limits
  - Examples
- [Data Structures](#)
  - TABLE Types
  - OBJECT Types
- [Summary of DBMS\\_OUTPUT Subprograms](#)

## Using DBMS\_OUTPUT

This section contains topics which relate to using the DBMS\_OUTPUT package.

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)
- [Exceptions](#)
- [Rules and Limits](#)
- [Examples](#)

## Overview

The package is typically used for debugging, or for displaying messages and reports to SQL\*DBA or SQL\*Plus (such as are produced by applying the SQL command DESCRIBE to procedures).

The [PUT Procedure](#) and [PUT\\_LINE Procedure](#) in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the [GET\\_LINE Procedure](#) and [GET\\_LINES Procedure](#).

If the package is disabled, all calls to subprograms are ignored. In this way, you can design your application so that subprograms are available only when a client is able to process the information.

## Security Model

The `dbmsotpt.sql` script must be run as user `SYS`. This creates the public synonym `DBMS_OUTPUT`, and `EXECUTE` permission on this package is granted to `public`.

---

## Operational Notes

- If you do not call `GET_LINE`, or if you do not display the messages on your screen in SQL\*Plus, the buffered messages are ignored.
- SQL\*Plus calls `GET_LINES` after issuing a SQL statement or anonymous PL/SQL calls.
- Typing `SET SERVEROUTPUT ON` in SQL\*Plus has the effect of invoking `DBMS_OUTPUT.ENABLE (buffer_size => NULL);`  
with no limit on the output.
- You should generally avoid having application code invoke either the [DISABLE Procedure](#) or [ENABLE Procedure](#) because this could subvert the attempt of an external tool like SQL\*Plus to control whether or not to display output.

---

---

**Note:** Messages sent using `DBMS_OUTPUT` are not actually sent until the sending subprogram or trigger completes. There is no mechanism to flush output during the execution of a procedure.

---

---

## Exceptions

DBMS\_OUTPUT subprograms raise the application error ORA-20000, and the output procedures can return the following errors:

**Table 100-1 DBMS\_OUTPUT Errors**

<b>Error</b>	<b>Description</b>
ORU-10027:	Buffer overflow
ORU-10028:	Line length overflow

## Rules and Limits

- The maximum line size is 32767 bytes.
- The default buffer size is 20000 bytes. The minimum size is 2000 bytes and the maximum is unlimited.

## Examples

### Example 1: Using a Trigger to Produce Output

You can use a trigger to print out some output from the debugging process. For example, you could code the trigger to invoke:

```
DBMS_OUTPUT.PUT_LINE('I got here: '||:new.col||' is the new value');
```

If you have enabled the DBMS\_OUTPUT package, then the text produced by this PUT\_LINE would be buffered, and you could, after executing the statement (presumably some INSERT, DELETE, or UPDATE that caused the trigger to fire), retrieve the line of information. For example:

```
BEGIN
  DBMS_OUTPUT.GET_LINE(:buffer, :status);
END;
```

You could then optionally display the buffer on the screen. You repeat calls to GET\_LINE until status comes back as nonzero. For better performance, you should use calls to [GET\\_LINES Procedure](#) which can return an array of lines.

### Example 2: Debugging Stored Procedures and Triggers

The DBMS\_OUTPUT package is commonly used to debug stored procedures and triggers. This package can also be used to enable you to retrieve information about an object and format this output, as shown in ["Example 3: Retrieving Information About an Object"](#) on page 100-9.

This function queries the employee table and returns the total salary for a specified department. The function includes several calls to the PUT\_LINE procedure:

```
CREATE FUNCTION dept_salary (dnum NUMBER) RETURN NUMBER IS
  CURSOR emp_cursor IS
    SELECT sal, comm FROM emp WHERE deptno = dnum;
  total_wages  NUMBER(11, 2) := 0;
  counter      NUMBER(10) := 1;
BEGIN

  FOR emp_record IN emp_cursor LOOP
    emp_record.comm := NVL(emp_record.comm, 0);
    total_wages := total_wages + emp_record.sal
      + emp_record.comm;
    DBMS_OUTPUT.PUT_LINE('Loop number = ' || counter ||
      ' ; Wages = ' || TO_CHAR(total_wages)); /* Debug line */
    counter := counter + 1; /* Increment debug counter */
  END LOOP;
  /* Debug line */
  DBMS_OUTPUT.PUT_LINE('Total wages = ' ||
    TO_CHAR(total_wages));
  RETURN total_wages;

END dept_salary;
```

Assume the EMP table contains the following rows:

EMPNO	SAL	COMM	DEPT
1002	1500	500	20
1203	1000		30
1289	1000		10



```
1347          1000      250      20
```

Assume the user executes the following statements in SQL\*Plus:

```
SET SERVEROUTPUT ON
VARIABLE salary NUMBER;
EXECUTE :salary := dept_salary(20);
```

The user would then see the following information displayed in the output pane:

```
Loop number = 1; Wages = 2000
Loop number = 2; Wages = 3250
Total wages = 3250
```

PL/SQL procedure successfully executed.

### Example 3: Retrieving Information About an Object

In this example, the user has used the EXPLAIN PLAN command to retrieve information about the execution plan for a statement and has stored it in PLAN\_TABLE. The user has also assigned a statement ID to this statement. The example EXPLAIN\_OUT procedure retrieves the information from this table and formats the output in a nested manner that more closely depicts the order of steps undergone in processing the SQL statement.

```

/*****
/* Create EXPLAIN_OUT procedure. User must pass STATEMENT_ID to */
/* to procedure, to uniquely identify statement.                */
*****/
CREATE OR REPLACE PROCEDURE explain_out
(statement_id IN VARCHAR2) AS

-- Retrieve information from PLAN_TABLE into cursor EXPLAIN_ROWS.

CURSOR explain_rows IS
SELECT level, id, position, operation, options,
       object_name
FROM plan_table
WHERE statement_id = explain_out.statement_id
CONNECT BY PRIOR id = parent_id
       AND statement_id = explain_out.statement_id
START WITH id = 0
       ORDER BY id;

BEGIN

-- Loop through information retrieved from PLAN_TABLE:

FOR line IN explain_rows LOOP

-- At start of output, include heading with estimated cost.

IF line.id = 0 THEN
DBMS_OUTPUT.PUT_LINE ('Plan for statement '
|| statement_id
|| ', estimated cost = ' || line.position);
END IF;

-- Output formatted information. LEVEL determines indention level.

DBMS_OUTPUT.PUT_LINE (lpad(' ', 2*(line.level-1)) ||

```

```
        line.operation || ' ' || line.options || ' ' ||  
        line.object_name);  
    END LOOP;  
  
END;
```

**See Also:** [Chapter 223, "UTL\\_FILE"](#)

## Data Structures

The DBMS\_OUTPUT package declares 2 collection types for use with the [GET\\_LINES Procedure](#).

### TABLE Types

[CHARARR Table Type](#)

### OBJECT Types

[DBMSOUTPUT\\_LINESARRAY Object Type](#)

## CHARARR Table Type

This package type is to be used with the [GET\\_LINES Procedure](#) to obtain text submitted through the [PUT Procedure](#) and [PUT\\_LINE Procedure](#).

### Syntax

```
TYPE CHARARR IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

## DBMSOUTPUT\_LINESARRAY Object Type

This package type is to be used with the [GET\\_LINES Procedure](#) to obtain text submitted through the [PUT Procedure](#) and [PUT\\_LINE Procedure](#).

### Syntax

```
TYPE DBMSOUTPUT_LINESARRAY IS  
  VARRAY(2147483647) OF VARCHAR2(32767);
```

## Summary of DBMS\_OUTPUT Subprograms

**Table 100–2** *DBMS\_OUTPUT Package Subprograms*

Subprogram	Description
<a href="#">DISABLE Procedure</a> on page 100-15	Disables message output
<a href="#">ENABLE Procedure</a> on page 100-16	Enables message output
<a href="#">GET_LINE Procedure</a> on page 100-17	Retrieves one line from buffer
<a href="#">GET_LINES Procedure</a> on page 100-18	Retrieves an array of lines from buffer
<a href="#">NEW_LINE Procedure</a> on page 100-19	Terminates a line created with <code>PUT</code>
<a href="#">PUT Procedure</a> on page 100-20	Places a partial line in the buffer
<a href="#">PUT_LINE Procedure</a> on page 100-21	Places line in buffer

---

**Note:** The [PUT Procedure](#) that take a number are obsolete and, while currently supported, are included in this release for legacy reasons only.

---

## DISABLE Procedure

This procedure disables calls to `PUT`, `PUT_LINE`, `NEW_LINE`, `GET_LINE`, and `GET_LINES`, and purges the buffer of any remaining information.

As with the [ENABLE Procedure](#), you do not need to call this procedure if you are using the `SERVEROUTPUT` option of SQL\*Plus.

### Syntax

```
DBMS_OUTPUT.DISABLE;
```

### Pragmas

```
pragma restrict_references (disable, WNDS, RNDS);
```

## ENABLE Procedure

This procedure enables calls to `PUT`, `PUT_LINE`, `NEW_LINE`, `GET_LINE`, and `GET_LINES`. Calls to these procedures are ignored if the `DBMS_OUTPUT` package is not activated.

### Syntax

```
DBMS_OUTPUT.ENABLE (  
    buffer_size IN INTEGER DEFAULT 20000);
```

### Pragmas

```
pragma restrict_references(enable,WNDS,RNDS);
```

### Parameters

**Table 100–3** *ENABLE Procedure Parameters*

Parameter	Description
<code>buffer_size</code>	Upper limit, in bytes, the amount of buffered information. Setting <code>buffer_size</code> to <code>NULL</code> specifies that there should be no limit.

### Usage Notes

- It is not necessary to call this procedure when you use the `SET SERVEROUTPUT` option of `SQL*Plus`.
- If there are multiple calls to `ENABLE`, then `buffer_size` is the last of the values specified. The maximum size is 1,000,000, and the minimum is 2,000 when the user specifies `buffer_size` (`NOT NULL`).
- `NULL` is expected to be the usual choice. The default is 20,000 for backwards compatibility with earlier database versions that did not support unlimited buffering.



## GET\_LINE Procedure

This procedure retrieves a single line of buffered information.

### Syntax

```
DBMS_OUTPUT.GET_LINE (
    line    OUT VARCHAR2,
    status  OUT INTEGER);
```

### Parameters

**Table 100–4** *GET\_LINE Procedure Parameters*

Parameter	Description
line	Returns a single line of buffered information, excluding a final newline character. You should declare the actual for this parameter as VARCHAR2 (32767) to avoid the risk of "ORA-06502: PL/SQL: numeric or value error: character string buffer too small".
status	If the call completes successfully, then the status returns as 0. If there are no more lines in the buffer, then the status is 1.

### Usage Notes

- You can choose to retrieve from the buffer a single line or an array of lines. Call the GET\_LINE procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the GET\_LINES procedure to retrieve an array of lines from the buffer.
- You can choose to automatically display this information if you are using SQL\*Plus by using the special SET SERVEROUTPUT ON command.
- After calling GET\_LINE or GET\_LINES, any lines not retrieved before the next call to PUT, PUT\_LINE, or NEW\_LINE are discarded to avoid confusing them with the next message.

## GET\_LINES Procedure

This procedure retrieves an array of lines from the buffer.

### Syntax

```
DBMS_OUTPUT.GET_LINES (
  lines      OUT   CHARARR,
  numlines   IN OUT INTEGER);

DBMS_OUTPUT.GET_LINES (
  lines      OUT   DBMSOUTPUT_LINESARRAY,
  numlines   IN OUT INTEGER);
```

### Parameters

**Table 100–5** GET\_LINES Procedure Parameters

Parameter	Description
lines	Returns an array of lines of buffered information. The maximum length of each line in the array is 32767 bytes. It is recommended that you use the VARRAY overload version in a 3GL host program to execute the procedure from a PL/SQL anonymous block.
numlines	Number of lines you want to retrieve from the buffer.  After retrieving the specified number of lines, the procedure returns the number of lines actually retrieved. If this number is less than the number of lines requested, then there are no more lines in the buffer.

### Usage Notes

- You can choose to retrieve from the buffer a single line or an array of lines. Call the GET\_LINE procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the GET\_LINES procedure to retrieve an array of lines from the buffer.
- You can choose to automatically display this information if you are using SQL\*Plus by using the special SET SERVEROUTPUT ON command.
- After calling GET\_LINE or GET\_LINES, any lines not retrieved before the next call to PUT, PUT\_LINE, or NEW\_LINE are discarded to avoid confusing them with the next message.

## NEW\_LINE Procedure

This procedure puts an end-of-line marker. The [GET\\_LINE Procedure](#) and the [GET\\_LINES Procedure](#) return "lines" as delimited by "newlines". Every call to the [PUT\\_LINE Procedure](#) or [NEW\\_LINE Procedure](#) generates a line that is returned by [GET\\_LINE\(S\)](#).

### Syntax

```
DBMS_OUTPUT.NEW_LINE;
```

## PUT Procedure

This procedure places a partial line in the buffer.

---



---

**Note:** The PUT procedure that takes a NUMBER is obsolete and, while currently supported, is included in this release for legacy reasons only.

---



---

### Syntax

```
DBMS_OUTPUT.PUT (
    item IN VARCHAR2);
```

### Parameters

**Table 100–6** PUT Procedure Parameters

Parameter	Description
item	Item to buffer.

### Exceptions

**Table 100–7** PUT Procedure Exceptions

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 32767 bytes for each line.

### Usage Notes

- You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT\_LINE.
- When you call PUT\_LINE the item you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW\_LINE. GET\_LINE and GET\_LINES do not return lines that have not been terminated with a newline character.
- If your lines exceed the line limit, you receive an error message.
- Output that you create using PUT or PUT\_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

For example, SQL\*Plus does not display DBMS\_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS\_OUTPUT buffers within the PL/SQL program.

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE ('hello');
  3  DBMS_LOCK.SLEEP (10);
  4  END;
```

## PUT\_LINE Procedure

This procedure places a line in the buffer.

---

**Note:** The PUT\_LINE procedure that takes a NUMBER is obsolete and, while currently supported, is included in this release for legacy reasons only.

---

### Syntax

```
DBMS_OUTPUT.PUT_LINE (
    item IN VARCHAR2);
```

### Parameters

**Table 100–8** PUT\_LINE Procedure Parameters

Parameter	Description
item	Item to buffer.

### Exceptions

**Table 100–9** PUT\_LINE Procedure Exceptions

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 32767 bytes for each line.

### Usage Notes

- You can build a line of information piece by piece by making multiple calls to PUT, or place an entire line of information into the buffer by calling PUT\_LINE.
- When you call PUT\_LINE the item you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW\_LINE. GET\_LINE and GET\_LINES do not return lines that have not been terminated with a newline character.
- If your lines exceeds the line limit, you receive an error message.
- Output that you create using PUT or PUT\_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

For example, SQL\*Plus does not display DBMS\_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS\_OUTPUT buffers within the PL/SQL program. For example:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
    2 DBMS_OUTPUT.PUT_LINE ('hello');
    3 DBMS_LOCK.SLEEP (10);
    4 END;
```



---

---

## DBMS\_PARALLEL\_EXECUTE

The DBMS\_PARALLEL\_EXECUTE package enables the user to incrementally update table data in parallel.

**See Also:**

- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database Reference*

This chapter contains the following topics:

- [Using DBMS\\_PARALLEL\\_EXECUTE](#)
  - Overview
  - Security Model
  - Constants
  - Views
  - Exceptions
  - Examples
- [Summary of DBMS\\_PARALLEL\\_EXECUTE Subprograms](#)

## Using DBMS\_PARALLEL\_EXECUTE

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)
- [Exceptions](#)
- [Examples](#)



## Overview

This package enables the user to incrementally update table data in parallel, in two high level steps:

1. Group sets of rows in the table into smaller sized chunks.
2. Run a user specified statement on these chunks in parallel, and commit when finished processing each chunk.

This package introduces the notion of *parallel execution task*. This task groups the various steps associated with the parallel execution of a PL/SQL block, which is typically updating table data.

All of the package subroutines (except the [GENERATE\\_TASK\\_NAME Function](#) and the [TASK\\_STATUS Procedure](#)) perform a commit.

## Security Model

DBMS\_PARALLEL\_EXECUTE is a SYS-owned package which is granted to PUBLIC.

Any user can create or operate his own parallel execution tasks and access the USER view.

Users who have the ADM\_PARALLEL\_EXECUTE\_TASK role can perform administrative routines (qualified by the prefix ADM\_) and access the DBA view.

Apart from the administrative routines, all the subprograms refer to tasks owned by the current user.

To execute chunks in parallel, the user must have CREATE JOB system privilege.

The CHUNK\_BY\_SQL, RUN\_TASK, and RESUME\_TASK subprograms require a query, and are executed using DBMS\_SQL. Invokers of the DBMS\_SQL interface must ensure that none of the queries passed-in contains SQL injection.

## Constants

The DBMS\_PARALLEL\_EXECUTE package uses the constants shown in following tables:

- [Table 101-1, "DBMS\\_PARALLEL\\_EXECUTE Constants - Chunk Status Value"](#)
- [Table 101-2, "DBMS\\_PARALLEL\\_EXECUTE Constants - Task Status Value"](#)

**Table 101-1 DBMS\_PARALLEL\_EXECUTE Constants - Chunk Status Value**

Constant	Type	Value	Description
ASSIGNED	NUMBER	1	Chunk has been assigned for processing
PROCESSED	NUMBER	2	Chunk has been processed successfully
PROCESSED_ WITH_ERROR	NUMBER	3	Chunk has been processed, but an error occurred during processing
UNASSIGNED	NUMBER	0	Chunk is unassigned

**Table 101-2 DBMS\_PARALLEL\_EXECUTE Constants - Task Status Value**

Constant	Type	Value	Description
CHUNKED	NUMBER	4	Table associated with the task has been chunked, but none of the chunk has been assigned for processing
CHUNKING	NUMBER	2	Table associated with the task is being chunked
CHUNKING_ FAILED	NUMBER	3	Chunking failed
CRASHED	NUMBER	8	Only applicable if parallel execution is used, this occurs if a job slave crashes or if the database crashes during EXECUTE, leaving a chunk in ASSIGNED or UNASSIGNED state.
CREATED	NUMBER	1	The task has been created by the <a href="#">CREATE_TASK Procedure</a>
FINISHED	NUMBER	6	All chunks processed without error
FINISHED_ WITH_ERROR	NUMBER	7	All chunks processed, but with errors in some cases
PROCESSING	NUMBER	5	Part of the chunk assigned for processing, or which has been processed

## Views

The `DBMS_PARALLEL_EXECUTE` package uses views listed in the *Oracle Database Reference*:

- `DBA_PARALLEL_EXECUTE_CHUNKS`
- `DBA_PARALLEL_EXECUTE_TASKS`
- `USER_PARALLEL_EXECUTE_CHUNKS`
- `USER_PARALLEL_EXECUTE_TASKS`

## Exceptions

The following table lists the exceptions raised by DBMS\_PARALLEL\_EXECUTE.

**Table 101-3 Exceptions Raised by DBMS\_PARALLEL\_EXECUTE**

Exception	Error Code	Description
CHUNK_NOT_FOUND	29499	Specified chunk does not exist
DUPLICATE_TASK_NAME	29497	Same task name has been used by an existing task
INVALID_STATE_FOR_CHUNK	29492	Attempts to chunk a table that is not in CREATED or CHUNKING_FAILED state
INVALID_STATE_FOR_REDSUME	29495	Attempts to resume execution, but the task is not in FINISHED_WITH_ERROR or CRASHED state
INVALID_STATE_FOR_RUN	29494	Attempts to execute the task that is not in CHUNKED state
INVALID_STATUS	29493	Attempts to set an invalid value to the chunk status
INVALID_TABLE	29491	Attempts to chunk a table by rowid in cases in which the table is not a physical table, or the table is an IOT
MISSING_ROLE	29490	User does not have the necessary ADM_PARALLEL_EXECUTE role
TASK_NOT_FOUND	29498	Specified task_name does not exist

## Examples

The following examples run on the Human Resources (HR) schema of the Oracle Database Sample Schemas. It requires that the HR schema be created with the `JOB SYSTEM` privilege.

### Chunk by ROWID

This example shows the most common usage of this package. After calling the [RUN\\_TASK Procedure](#), it checks for errors and re-runs in the case of error.

```

DECLARE
  l_sql_stmt VARCHAR2(1000);
  l_try NUMBER;
  l_status NUMBER;
BEGIN

  -- Create the TASK
  DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');

  -- Chunk the table by ROWID
  DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID('mytask', 'HR', 'EMPLOYEES', true,
100);

  -- Execute the DML in parallel
  l_sql_stmt := 'update /*+ ROWID (dda) */ EMPLOYEES e
    SET e.salary = e.salary + 10
    WHERE rowid BETWEEN :start_id AND :end_id';
  DBMS_PARALLEL_EXECUTE.RUN_TASK('mytask', l_sql_stmt, DBMS_SQL.NATIVE,
    parallel_level => 10);

  -- If there is an error, RESUME it for at most 2 times.
  L_try := 0;
  L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
  WHILE(l_try < 2 and L_status != DBMS_PARALLEL_EXECUTE.FINISHED)
  LOOP
    L_try := l_try + 1;
    DBMS_PARALLEL_EXECUTE.RESUME_TASK('mytask');
    L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
  END LOOP;

  -- Done with processing; drop the task
  DBMS_PARALLEL_EXECUTE.DROP_TASK('mytask');

END;
/

```

### Chunk by User-Provided SQL

A user can specify their own chunk algorithm by using the [CREATE\\_CHUNKS\\_BY\\_SQL Procedure](#). This example shows that rows with the same `manager_id` are grouped together and processed in one chunk.

```

DECLARE
  l_chunk_sql VARCHAR2(1000);
  l_sql_stmt VARCHAR2(1000);
  l_try NUMBER;
  l_status NUMBER;
BEGIN

```

```

-- Create the TASK
DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');

-- Chunk the table by MANAGER_ID
l_chunk_sql := 'SELECT distinct manager_id, manager_id FROM employees';
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL('mytask', l_chunk_sql, false);

-- Execute the DML in parallel
-- the WHERE clause contain a condition on manager_id, which is the chunk
-- column. In this case, grouping rows is by manager_id.
l_sql_stmt := 'update /*+ ROWID (dda) */ EMPLOYEES e
              SET e.salary = e.salary + 10
              WHERE manager_id between :start_id and :end_id';
DBMS_PARALLEL_EXECUTE.RUN_TASK('mytask', l_sql_stmt, DBMS_SQL.NATIVE,
                               parallel_level => 10);

-- If there is error, RESUME it for at most 2 times.
L_try := 0;
L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
WHILE(l_try < 2 and L_status != DBMS_PARALLEL_EXECUTE.FINISHED)
Loop
  L_try := l_try + 1;
  DBMS_PARALLEL_EXECUTE.RESUME_TASK('mytask');
  L_status := DBMS_PARALLEL_EXECUTE.TASK_STATUS('mytask');
END LOOP;

-- Done with processing; drop the task
DBMS_PARALLEL_EXECUTE.DROP_TASK('mytask');

end;
/

```

### Executing Chunks in an User-defined Framework

The user can execute chunks in his own defined framework without using the [RUN\\_TASK Procedure](#). This example shows how to use [GET\\_ROWID\\_CHUNK Procedure](#), [EXECUTE IMMEDIATE](#), [SET\\_CHUNK\\_STATUS Procedure](#) to execute the chunks.

```

DECLARE
  l_sql_stmt varchar2(1000);
  l_try number;
  l_status number;
  l_chunk_id number;
  l_start_rowid rowid;
  l_end_rowid rowid;
  l_any_rows boolean;
  CURSOR c1 IS SELECT chunk_id
                FROM user_parallel_execute_chunks
                WHERE task_name = 'mytask'
                AND STATUS IN (DBMS_PARALLEL_EXECUTE.PROCESSED_WITH_ERROR,
                              DBMS_PARALLEL_EXECUTE.ASSIGNED);
BEGIN
  -- Create the Objects, task, and chunk by ROWID
  DBMS_PARALLEL_EXECUTE.CREATE_TASK ('mytask');
  DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID('mytask', 'HR', 'EMPLOYEES', true,
  100);

  l_sql_stmt := 'update /*+ ROWID (dda) */ EMPLOYEES e
                SET e.salary = e.salary + 10
                WHERE rowid BETWEEN :start_id AND :end_id';

```

```
-- Execute the DML in his own framework
--
-- Process each chunk and commit.
-- After processing one chunk, repeat this process until
-- all the chunks are processed.
--
<<main_processing>>
LOOP
  --
  -- Get a chunk to process; if there is nothing to process, then exit the
  -- loop;
  --
  DBMS_PARALLEL_EXECUTE.GET_ROWID_CHUNK('mytask',
                                        l_chunk_id,
                                        l_start_rowid,
                                        l_end_rowid,
                                        l_any_rows);

  IF (l_any_rows = false) THEN EXIT; END IF;

  --
  -- The chunk is specified by start_id and end_id.
  -- Bind the start_id and end_id and then execute it
  --
  -- If no error occurred, set the chunk status to PROCESSED.
  --
  -- Catch any exception. If an exception occurred, store the error num/msg
  -- into the chunk table and then continue to process the next chunk.
  --
  BEGIN
    EXECUTE IMMEDIATE l_sql_stmt using l_start_rowid, l_end_rowid;
    DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS('mytask',l_chunk_id,
      DBMS_PARALLEL_EXECUTE.PROCESSED);
  EXCEPTION WHEN OTHERS THEN
    DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS('mytask', l_chunk_id,
      DBMS_PARALLEL_EXECUTE.PROCESSED_WITH_ERROR, SQLCODE, SQLERRM);
  END;

  --
  -- Finished processing one chunk; Commit here
  --
  COMMIT;
END LOOP;
```



---

## Summary of DBMS\_PARALLEL\_EXECUTE Subprograms

**Table 101-4 DBMS\_PARALLEL\_EXECUTE Package Subprograms**

Subprogram	Description
<a href="#">ADM_DROP_CHUNKS Procedure</a> on page 101-12	Drops all chunks of the specified task owned by the specified owner
<a href="#">ADM_DROP_TASK Procedure</a> on page 101-13	Drops the task of the given user and all related chunks
<a href="#">ADM_TASK_STATUS Procedure</a> on page 101-14	Returns the task status
<a href="#">ADM_STOP_TASK Procedure</a> on page 101-15	Stops the task of the given owner and related job slaves
<a href="#">CREATE_TASK Procedure</a> on page 101-16	Creates a task for the current user
<a href="#">CREATE_CHUNKS_BY_NUMBER_COL Procedure</a> on page 101-17	Chunks the table associated with the given task by the specified column.
<a href="#">CREATE_CHUNKS_BY_ROWID Procedure</a> on page 101-18	Chunks the table associated with the given task by ROWID
<a href="#">CREATE_CHUNKS_BY_SQL Procedure</a> on page 101-19	Chunks the table associated with the given task by means of a user-provided SELECT statement
<a href="#">DROP_TASK Procedure</a> on page 101-20	Drops the task and all related chunks
<a href="#">DROP_CHUNKS Procedure</a> on page 101-21	Drops the task's chunks
<a href="#">GENERATE_TASK_NAME Function</a> on page 101-22	Returns a unique name for a task
<a href="#">GET_NUMBER_COL_CHUNK Procedure</a> on page 101-23	Picks an unassigned NUMBER chunk and changes it to ASSIGNED
<a href="#">GET_ROWID_CHUNK Procedure</a> on page 101-24	Picks an unassigned ROWID chunk and changes it to ASSIGNED
<a href="#">PURGE_PROCESSED_CHUNKS Procedure</a> on page 101-25	Deletes all the processed chunks whose status is PROCESSED or PROCESSED_WITH_ERROR
<a href="#">RESUME_TASK Procedures</a> on page 101-26	Retries the given the task if the <a href="#">RUN_TASK Procedure</a> finished with error, or resumes the task if a crash has occurred.
<a href="#">RUN_TASK Procedure</a> on page 101-28	Executes the specified SQL statement on the chunks in parallel
<a href="#">SET_CHUNK_STATUS Procedure</a> on page 101-30	Sets the status of the chunk
<a href="#">STOP_TASK Procedure</a> on page 101-31	Stops the task and related job slaves
<a href="#">TASK_STATUS Procedure</a> on page 101-32	Returns the task status

## ADM\_DROP\_CHUNKS Procedure

This procedure drops all chunks of the specified task owned by the specified owner.

### Syntax

```
DBMS_PARALLEL_EXECUTE.ADM_DROP_CHUNKS (  
    task_owner      IN  VARCHAR2,  
    task_name       IN  VARCHAR2);
```

### Parameters

**Table 101–5** ADM\_DROP\_CHUNKS Procedure Parameters

Parameter	Description
task_owner	Owner of the task
task_name	Name of the task

## ADM\_DROP\_TASK Procedure

This procedure drops the task of the specified user and all related chunks.

### Syntax

```
DBMS_PARALLEL_EXECUTE.ADM_DROP_TASK (  
    task_owner      IN VARCHAR2,  
    task_name       IN VARCHAR2);
```

### Parameters

**Table 101–6 ADM\_DROP\_TASK Procedure Parameters**

Parameter	Description
task_owner	Owner of the task
task_name	Name of the task

## ADM\_TASK\_STATUS Procedure

This function returns the task status.

### Syntax

```
DBMS_PARALLEL_EXECUTE.ADM_TASK_STATUS (  
    task_owner      IN  VARCHAR2,  
    task_name       IN  VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 101–7** ADM\_TASK\_STATUS Function Parameters

Parameter	Description
task_owner	Owner of the task
task_name	Name of the task

## ADM\_STOP\_TASK Procedure

This procedure stops the task of the specified owner and related job slaves.

### Syntax

```
DBMS_PARALLEL_EXECUTE.ADM_STOP_TASK (  
    task_owner      IN  VARCHAR2,  
    task_name       IN  VARCHAR2);
```

### Parameters

**Table 101–8** ADM\_STOP\_TASK Procedure Parameters

Parameter	Description
task_owner	Owner of the task
task_name	Name of the task

## CREATE\_TASK Procedure

This procedure creates a task for the current user. The pairing of `task_name` and `current_user` must be unique.

### Syntax

```
DBMS_PARALLEL_EXECUTE.CREATE_TASK (  
    task_name      IN   VARCHAR2,  
    comment        IN   VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 101–9 CREATE\_TASK Procedure Parameters**

Parameter	Description
<code>task_name</code>	Name of the task. The <code>task_name</code> can be any string in which related length must be less than or equal to 128 bytes.
<code>comment</code>	Comment field. The comment must be less than 4000 bytes.

## CREATE\_CHUNKS\_BY\_NUMBER\_COL Procedure

This procedure chunks the table (associated with the specified task) by the specified column. The specified column must be a NUMBER column. This procedure takes the MIN and MAX value of the column, and then divide the range evenly according to chunk\_size. The chunks are:

START_ID	END_ID
-----	-----
min_id_val	min_id_val+1*chunk_size-1
min_id_val+1*chunk_size	min_id_val+2*chunk_size-1
...	...
min_id_val+i*chunk_size	max_id_val

### Syntax

```
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_NUMBER_COL (
  task_name      IN  VARCHAR2,
  table_owner    IN  VARCHAR2,
  table_name     IN  VARCHAR2,
  table_column   IN  VARCHAR2,
  chunk_size     IN  NUMBER);
```

### Parameters

**Table 101-10 CREATE\_CHUNKS\_BY\_NUMBER\_COL Procedure Parameters**

Parameter	Description
task_name	Name of the task
table_owner	Owner of the table
table_name	Name of the table
table_column	Name of the NUMBER column
chunk_size	Range of each chunk

## CREATE\_CHUNKS\_BY\_ROWID Procedure

This procedure chunks the table (associated with the specified task) by ROWID. `num_row` and `num_block` are approximate guidance for the size of each chunk. The table to be chunked must be a physical table with physical ROWID having views and table functions. Index Organized Tables are not allowed.

### Syntax

```
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID (  
    task_name      IN  VARCHAR2,  
    table_owner    IN  VARCHAR2,  
    table_name     IN  VARCHAR2,  
    by_row         IN  BOOLEAN,  
    chunk_size     IN  NUMBER);
```

### Parameters

**Table 101-11** CREATE\_CHUNKS\_BY\_ROWID Procedure Parameters

Parameter	Description
<code>task_name</code>	Name of the task
<code>table_owner</code>	Owner of the table
<code>table_name</code>	Name of the table
<code>by_row</code>	TRUE if <code>chunk_size</code> refers to the number of rows, otherwise, <code>chunk_size</code> refers to the number of blocks
<code>chunk_size</code>	Approximate number of rows/blocks to process for each commit cycle



## CREATE\_CHUNKS\_BY\_SQL Procedure

This procedure chunks the table (associated with the specified task) by means of a user-provided SELECT statement. The select statement which returns the range of each chunk must have two columns: `start_id` and `end_id`. If task is to chunk by ROWID, then the two columns must be of ROWID type. If the task is to chunk the table by NUMBER column, then the two columns must be of NUMBER type. The procedure provides the flexibility to users who want to deploy user-defined chunk algorithms.

### Syntax

```
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL (
    task_name      IN  VARCHAR2,
    sql_statement  IN  CLOB,
    by_rowid       IN  BOOLEAN);
```

### Parameters

**Table 101-12 CREATE\_CHUNKS\_BY\_SQL Procedure Parameters**

Parameter	Description
<code>task_name</code>	Name of the task
<code>sql_statment</code>	SQL which returns the chunk ranges
<code>by_rowid</code>	TRUE if the table is chunked by rowids

## DROP\_TASK Procedure

This procedure drops the task and all related chunks.

### Syntax

```
DBMS_PARALLEL_EXECUTE.DROP_TASK (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 101–13** *DROP\_TASK Procedure Parameters*

Parameter	Description
task_name	Name of the task

## DROP\_CHUNKS Procedure

This procedure drops the task's chunks.

### Syntax

```
DBMS_PARALLEL_EXECUTE.DROP_CHUNKS (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 101–14** *DROP\_CHUNKS Procedure Parameters*

Parameter	Description
task_name	Name of the task

## GENERATE\_TASK\_NAME Function

This function returns a unique name for a task. The name is of the form *prefix*N where N is a number from a sequence. If no prefix is specified, the generated name is, by default, be TASK\$\_1, TASK\$\_2, TASK\$\_3, and so on. If 'SCOTT' is specified as the prefix, the name is SCOTT1, SCOTT2, and so on.

### Syntax

```
DBMS_PARALLEL_EXECUTE.GENERATE_TASK_NAME (  
    prefix      IN      VARCHAR2 DEFAULT 'TASK$_'  
    RETURN VARCHAR2;
```

### Parameters

**Table 101–15** GENERATE\_TASK\_NAME Function Parameters

Parameter	Description
prefix	The prefix to use when generating the task name

## GET\_NUMBER\_COL\_CHUNK Procedure

This procedure picks an unassigned NUMBER chunk and changes it to ASSIGNED. If there are no more chunks to assign, `any_rows` is set to FALSE. Otherwise, the `chunk_id`, `start` and `end_id` of the chunk is returned as OUT parameters. The chunk info in `DBMS_PARALLEL_EXECUTE_CHUNKS$` is updated as follows: `STATUS` becomes ASSIGNED; `START_TIMESTAMP` records the current time; `END_TIMESTAMP` is cleared.

**See Also:** [Views](#) on page 101-6

### Syntax

```
DBMS_PARALLEL_EXECUTE.GET_NUMBER_COL_CHUNK (
    task_name      IN VARCHAR2,
    chunk_id       OUT NUMBER,
    start_rowid    OUT ROWID,
    end_id         OUT ROWID,
    any_rows       OUT BOOLEAN);
```

### Parameters

**Table 101–16** GET\_NUMBER\_COL\_CHUNK Procedure Parameters

Parameter	Description
<code>task_name</code>	Name of the task
<code>chunk_id</code>	Chunk_id of the chunk
<code>start_rowid</code>	Start rowid in the returned range
<code>end_id</code>	End rowid in the returned range
<code>any_rows</code>	Indicating if there could be any rows to process in the range

### Usage Notes

If the task is chunked by ROWID, then `get_rowid_range` should be used. If the task is chunked by NUMBER column, then `get_number_col_range` should be used. If the user makes the wrong function call, the returning `chunk_id` and `any_rows` has a valid value but `start/end_(row) id` is NULL.

## GET\_ROWID\_CHUNK Procedure

This procedure picks an unassigned ROWID chunk and changes it to ASSIGNED. If there are no more chunks to assign, any\_rows is set to FALSE. Otherwise, the chunk\_id, start and end\_id of the chunk is returned as OUT parameters. The chunk info in DBMS\_PARALLEL\_EXECUTE\_CHUNKS\$ is updated as follows: STATUS becomes ASSIGNED; START\_TIMESTAMP records the current time; END\_TIMESTAMP is cleared.

**See Also:** [Views](#) on page 101-6

### Syntax

```
DBMS_PARALLEL_EXECUTE.GET_ROWID_CHUNK (
    task_name      IN VARCHAR2,
    chunk_id       OUT NUMBER,
    start_rowid    OUT ROWID,
    end_id         OUT ROWID,
    any_rows       OUT BOOLEAN);
```

### Parameters

**Table 101–17** GET\_ROWID\_CHUNK Procedure Parameters

Parameter	Description
task_name	Name of the task
chunk_id	Chunk_id of the chunk
start_rowid	Start rowid in the returned range
end_id	End rowid in the returned range
any_rows	Indicating if there could be any rows to process in the range

### Usage Notes

If the task is chunked by ROWID, then get\_rowid\_range should be used. If the task is chunked by NUMBER column, then get\_number\_col\_range should be used. If the user makes the wrong function call, the returning chunk\_id and any\_rows still have valid value but start/end\_(row) id is NULL.

## PURGE\_PROCESSED\_CHUNKS Procedure

This procedure deletes all the processed chunks whose status is PROCESSED or PROCESSED\_WITH\_ERROR.

### Syntax

```
DBMS_PARALLEL_EXECUTE.PURGE_PROCESSED_CHUNKS (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 101–18** *PURGE\_PROCESSED\_CHUNKS Procedure Parameters*

Parameter	Description
task_name	Name of the task

## RESUME\_TASK Procedures

This procedure retries the specified the task if the [RUN\\_TASK Procedure](#) finished with error, or resumes the task if a crash has occurred.

The user can only invoke this procedure if the task is in a CRASHED or FINISHED\_WITH\_ERROR state. For a crashed serial execution, the state remains in processing. The FORCE option allows user to resume any task in PROCESSING state. However, it is the user's responsibility to determine that a crash has occurred.

The procedure resumes processing the chunks which have not been processed. Also, chunks which are in PROCESSED\_WITH\_ERROR or ASSIGNED (due to crash) state are processed because those chunks did not commit.

This procedure takes the same argument as the [RUN\\_TASK Procedure](#).

The overload which takes `task_name` as the only input argument re-uses the arguments provided in the previous invoking of the [RUN\\_TASK Procedure](#) or [RESUME\\_TASK Procedures](#).

**See Also:** [Table 101-2, "DBMS\\_PARALLEL\\_EXECUTE Constants - Task Status Value"](#) on page 101-5

### Syntax

```
DBMS_PARALLEL_EXECUTE.RESUME_TASK (
    task_name          IN  VARCHAR2,
    sql_stmt           IN  CLOB,
    language_flag      IN  NUMBER,
    edition            IN  VARCHAR2  DEFAULT NULL,
    apply_crossedition_trigger IN  VARCHAR2  DEFAULT NULL,
    fire_apply_trigger IN  BOOLEAN   DEFAULT TRUE,
    parallel_level     IN  NUMBER   DEFAULT 0,
    job_class          IN  VARCHAR2  DEFAULT 'DEFAULT_JOB_CLASS',
    force              IN  BOOLEAN   DEFAULT FALSE);

DBMS_PARALLEL_EXECUTE.RESUME_TASK (
    task_name          IN  VARCHAR2,
    force              IN  BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 101-19 RESUME\_TASK Procedure Parameters**

Parameter	Description
<code>task_name</code>	Name of the task
<code>sql_stmt</code>	SQL statement; must have <code>:start_id</code> and <code>:end_id</code> placeholder
<code>language_flag</code>	Determines how Oracle handles the SQL statement. The following options are recognized: <ul style="list-style-type: none"> <li>▪ V6 (or 0) specifies version 6 behavior</li> <li>▪ NATIVE (or 1) specifies normal behavior for the database to which the program is connected</li> <li>▪ V7 (or 2) specifies Oracle database version 7 behavior</li> </ul>
<code>edition</code>	Specifies the edition to run the statement in. Default is the current edition.



**Table 101–19 (Cont.) RESUME\_TASK Procedure Parameters**

Parameter	Description
apply_crossedition_trigger	Specifies the unqualified name of a forward crossedition trigger that is to be applied to the specified SQL. The name is resolved using the edition and current_schema setting in which the statement is to be executed. The trigger must be owned by the user who executes the statement.
fire_apply_trigger	Indicates whether the specified apply_crossedition_trigger is itself to be executed, or should only be a guide used in selecting other triggers
parallel_level	Number of parallel jobs; zero if run in serial; NULL uses the default parallelism
job_class	If running in parallel, the jobs all belong to the specified job class
force	If TRUE, do not raise an error if the status is PROCESSING.

## Examples

Suppose the chunk table contains the following chunk ranges:

START_ID	END_ID
-----	-----
1	10
11	20
21	30

And the specified SQL statement is:

```
UPDATE employees
   SET salary = salary + 10
   WHERE e.employee_id BETWEEN :start_id AND :end_id
```

This procedure executes the following statements in parallel:

```
UPDATE employees
   SET salary =.salary + 10 WHERE employee_id BETWEEN 1 and 10;
COMMIT;
```

```
UPDATE employees
   SET salary =.salary + 10 WHERE employee_id between 11 and 20;
COMMIT;
```

```
UPDATE employees
   SET salary =.salary + 10 WHERE employee_id between 21 and 30;
COMMIT;
```

## RUN\_TASK Procedure

This procedure executes the specified statement (`sql_stmt`) on the chunks in parallel. It commits after processing each chunk. The specified statement must have two placeholders called `start_id`, and `end_id` respectively, which represent the range of the chunk to be processed. The types of the placeholder must be `rowid` where `ROWID` based chunking was used, or `NUMBER` where number based chunking was used. The specified statement should not commit unless it is idempotent.

The SQL statement is executed as the current user.

If `apply_crossedition_trigger` is specified, `DBMS_CROSSEDITION_TRIGGER.IS_APPLYING` returns true for the sessions executing the SQL. Therefore, for parallel execution, `DBMS_CROSSEDITION_TRIGGER.IS_APPLYING` returns true in the Job slaves session.

Chunks can be executed in parallel by `DBMS_SCHEDULER` Job slaves. Therefore, parallel execution requires `CREATE JOB` system privilege. The Job slaves is created under the current user. The default number of Job slaves is computed as the product of Oracle parameters `cpu_count` and `parallel_threads_per_cpu`. On a Real Application Clusters installation, the number of Job slaves is the sum of individual settings on each node in the cluster. This procedure returns only when all the chunks are processed. In parallel cases, this procedure returns only when all the Job slaves finished.

### Syntax

```
DBMS_PARALLEL_EXECUTE.RUN_TASK (
    task_name          IN VARCHAR2,
    sql_stmt           IN CLOB,
    language_flag     IN NUMBER,
    edition            IN VARCHAR2 DEFAULT NULL,
    apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL,
    fire_apply_trigger IN BOOLEAN  DEFAULT TRUE,
    parallel_level     IN NUMBER   DEFAULT 0,
    job_class          IN VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS');
```

### Parameters

**Table 101–20 RUN\_TASK Procedure Parameters**

Parameter	Description
<code>task_name</code>	Name of the task
<code>sql_stmt</code>	SQL statement; must have <code>:start_id</code> and <code>:end_id</code> placeholder
<code>language_flag</code>	Determines how Oracle handles the SQL statement. The following options are recognized: <ul style="list-style-type: none"> <li>▪ V6 (or 0) specifies version 6 behavior</li> <li>▪ NATIVE (or 1) specifies normal behavior for the database to which the program is connected</li> <li>▪ V7 (or 2) specifies Oracle database version 7 behavior</li> </ul>
<code>edition</code>	Specifies the edition to run the statement in. Default is the current edition.

**Table 101–20 (Cont.) RUN\_TASK Procedure Parameters**

Parameter	Description
apply_crossedition_trigger	Specifies the unqualified name of a forward crossedition trigger that is to be applied to the specified SQL. The name is resolved using the edition and current_schema setting in which the statement is to be executed. The trigger must be owned by the user executes the statement.
fire_apply_trigger	Indicates whether the specified apply_crossedition_trigger is itself to be executed, or should only be a guide used in selecting other triggers.
parallel_level	Number of parallel jobs; zero if run in serial; NULL uses the default parallelism.\
job_class	If running in parallel, the jobs belong to the specified job class

## Examples

Suppose the chunk table contains the following chunk ranges:

START_ID	END_ID
-----	-----
1	10
11	20
21	30

And the specified SQL statement is:

```
UPDATE employees
   SET salary = salary + 10
   WHERE e.employee_id BETWEEN :start_id AND :end_id
```

This procedure executes the following statements in parallel:

```
UPDATE employees
   SET salary =.salary + 10 WHERE employee_id BETWEEN 1 and 10;
COMMIT;
```

```
UPDATE employees
   SET salary =.salary + 10 WHERE employee_id between 11 and 20;
COMMIT;
```

```
UPDATE employees
   SET salary =.salary + 10 WHERE employee_id between 21 and 30;
COMMIT;
```

## SET\_CHUNK\_STATUS Procedure

This procedure sets the status of the chunk. The `START_TIMESTAMP` and `END_TIMESTAMP` of the chunk is updated according to the new status:

Value of the new Status	Side Effect
-----	-----
UNASSIGNED	<code>START_TIMESTAMP</code> and <code>END_TIMESTAMP</code> will be cleared
ASSIGNED	<code>START_TIMESTAMP</code> will be the current time and <code>END_TIMESTAMP</code> will be cleared.
PROCESSED or PROCESSED_WITH_ERROR	The current time will be recorded in <code>END_TIMESTAMP</code>

**See Also:** [Views](#) on page 101-6

### Syntax

```
DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS (
    task_name      IN VARCHAR2,
    chunk_id       OUT NUMBER,
    status         IN NUMBER,
    err_num        IN NUMBER DEFAULT NULL,
    err_msg        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 101–21 SET\_CHUNK\_STATUS Procedure Parameters**

Parameter	Description
<code>task_name</code>	Name of the task
<code>chunk_id</code>	Chunk_id of the chunk
<code>status</code>	Status of the chunk: UNASSIGNED, ASSIGNED, PROCESSED, PROCESSED_WITH_ERROR
<code>err_num</code>	Error code returned during the processing of the chunk
<code>err_msg</code>	Error message returned during the processing of the chunk

## STOP\_TASK Procedure

This procedure stops the task and related job slaves.

### Syntax

```
DBMS_PARALLEL_EXECUTE.STOP_TASK (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 101–22 STOP\_TASK Procedure Parameters**

Parameter	Description
task_name	Name of the task

## TASK\_STATUS Procedure

This function returns the task status.

### Syntax

```
DBMS_PARALLEL_EXECUTE.TASK_STATUS (  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 101–23** *TASK\_STATUS Procedure Parameters*

Parameter	Description
task_name	Name of the task

The DBMS\_PCLXUTIL package provides intra-partition parallelism for creating partition-wise local indexes. DBMS\_PCLXUTIL circumvents the limitation that, for local index creation, the degree of parallelism is restricted to the number of partitions as only one slave process for each partition is used.

**See Also:** There are several rules concerning partitions and indexes. For more information, see *Oracle Database Concepts* and *Oracle Database Administrator's Guide*.

This chapter contains the following topics:

- [Using DBMS\\_PCLXUTIL](#)
  - Overview
  - Security Model
  - Operational Notes
  - Rules and Limits
- [Summary of DBMS\\_PCLXUTIL Subprograms](#)

---

## Using DBMS\_PCLXUTIL

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)
- [Rules and Limits](#)



## Overview

DBMS\_PCLXUTIL uses the DBMS\_JOB package to provide a greater degree of parallelism for creating a local index for a partitioned table. This is achieved by asynchronous inter-partition parallelism using the background processes (with DBMS\_JOB), in combination with intra-partition parallelism using the parallel query slave processes.

DBMS\_PCLXUTIL works with both range and range-hash composite partitioning.

The DBMS\_PCLXUTIL package can be used during the following DBA tasks:

### 1. Local index creation

The procedure `BUILD_PART_INDEX` assumes that the dictionary information for the local index already exists. This can be done by issuing the create index SQL command with the `UNUSABLE` option.

```
CREATE INDEX <idx_name> on <tab_name>(…) local(…) unusable;
```

This causes the dictionary entries to be created without "building" the index itself, the time consuming part of creating an index. Now, invoking the procedure `BUILD_PART_INDEX` causes a concurrent build of local indexes with the specified degree of parallelism.

```
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
```

For composite partitions, the procedure automatically builds local indexes for all subpartitions of the composite table.

### 2. Local index maintenance

By marking desired partitions usable or unusable, the `BUILD_PART_INDEX` procedure also enables selective rebuilding of local indexes. The `force_opt` parameter provides a way to override this and build local indexes for all partitions.

```
ALTER INDEX <idx_name> local(…) unusable;
```

Rebuild only the desired (sub)partitions (that are marked unusable):

```
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,FALSE);
```

Rebuild all (sub)partitions using `force_opt = TRUE`:

```
EXECUTE dbms_pclxutil.build_part_index(4,4,<tab_name>,<idx_name>,TRUE);
```

A progress report is produced, and the output appears on screen when the program is ended (because the `DBMS_OUTPUT` package writes messages to a buffer first, and flushes the buffer to the screen only upon termination of the program).

## **Security Model**

This utility can be run only as table owner, and not as any other user.

## Operational Notes

DBMS\_PCLXUTIL submits a job for each partition. It is the responsibility of the user/dba to control the number of concurrent jobs by setting the `INIT.ORA` parameter `JOB_QUEUE_PROCESSES` correctly. There is minimal error checking for correct syntax. Any errors are reported in the job queue process trace files.

## Rules and Limits

---

---

**Note:** For range partitioning, the minimum compatibility mode is 8.0; for range-hash composite partitioning, the minimum compatibility mode is 8*i*.

---

---

Because DBMS\_PCLXUTIL uses the DBMS\_JOB package, you must be aware of the following limitations pertaining to DBMS\_JOB:

- You must decide appropriate values for the `job_queue_processes` initialization parameter. Clearly, if the job processes are not started before calling `BUILD_PART_INDEX()`, then the package will not function properly. The background processes are specified by the following `init.ora` parameters:  

```
job_queue_processes=n    #the number of background processes = n
```
- Failure conditions are reported only in the trace files (a DBMS\_JOB limitation), making it impossible to give interactive feedback to the user. This package prints a failure message, removes unfinished jobs from the queue, and requests the user to take a look at the `j*.trc` trace files.

## Summary of DBMS\_PCLXUTIL Subprograms

---

**Table 102–1** *DBMS\_PCLXUTIL Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">BUILD_PART_INDEX Procedure</a> on page 102-8	Provides intra-partition parallelism for creating partition-wise local indexes

## BUILD\_PART\_INDEX Procedure

This procedure provides intra-partition parallelism for creating partition-wise local indexes.

### Syntax

```
DBMS_PCLXUTIL.BUILD_PART_INDEX (
  jobs_per_batch IN NUMBER   DEFAULT 1,
  procs_per_job  IN NUMBER   DEFAULT 1,
  tab_name       IN VARCHAR2 DEFAULT NULL,
  idx_name       IN VARCHAR2 DEFAULT NULL,
  force_opt      IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 102–2 BUILD\_PART\_INDEX Procedure Parameters**

Parameter	Description
jobs_per_batch	The number of concurrent partition-wise "local index builds".
procs_per_job	The number of parallel query slaves to be utilized for each local index build (1 <= procs_per_job <= max_slaves).
tab_name	The name of the partitioned table (an exception is raised if the table does not exist or not partitioned).
idx_name	The name given to the local index (an exception is raised if a local index is not created on the table tab_name).
force_opt	If TRUE, then force rebuild of all partitioned indexes; otherwise, rebuild only the partitions marked 'UNUSABLE'.

### Usage Notes

This utility can be run only as table owner, and not as any other user.

### Examples

Suppose a table PROJECT is created with two partitions PROJ001 and PROJ002, along with a local index IDX.

A call to the procedure BUILD\_PART\_INDEX (2, 4, 'PROJECT', 'IDX', TRUE) produces the following output:

```
SQLPLUS> EXECUTE dbms_pclxutil.build_part_index(2,4,'PROJECT','IDX',TRUE);
Statement processed.
INFO: Job #21 created for partition PROJ002 with 4 slaves
INFO: Job #22 created for partition PROJ001 with 4 slaves
```

The `DBMS_PIPE` package lets two or more sessions in the same instance communicate. Oracle pipes are similar in concept to the pipes used in UNIX, but Oracle pipes are not implemented using the operating system pipe mechanisms.

This chapter contains the following topics:

- [Using DBMS\\_PIPE](#)
  - Overview
  - Security Model
  - Constants
  - Operational Notes
  - Exceptions
  - Examples
- [Summary of DBMS\\_PIPE Subprograms](#)

## Using DBMS\_PIPE

---

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Operational Notes](#)
- [Exceptions](#)
- [Examples](#)



## Overview

Pipe functionality has several potential applications:

- **External service interface:** You can communicate with user-written services that are external to the RDBMS. This can be done effectively in a shared server process, so that several instances of the service are executing simultaneously. Additionally, the services are available asynchronously. The requestor of the service does not need to block a waiting reply. The requestor can check (with or without time out) at a later time. The service can be written in any of the 3GL languages that Oracle supports.
- **Independent transactions:** The pipe can communicate to a separate session which can perform an operation in an independent transaction (such as logging an attempted security violation detected by a trigger).
- **Alerters (non-transactional):** You can post another process without requiring the waiting process to poll. If an "after-row" or "after-statement" trigger were to alert an application, then the application would treat this alert as an indication that the data probably changed. The application would then read the data to get the current value. Because this is an "after" trigger, the application would want to do a "SELECT FOR UPDATE" to make sure it read the correct data.
- **Debugging:** Triggers and stored procedures can send debugging information to a pipe. Another session can keep reading out of the pipe and display it on the screen or write it to a file.
- **Concentrator:** This is useful for multiplexing large numbers of users over a fewer number of network connections, or improving performance by concentrating several user-transactions into one DBMS transaction.

## Security Model

Security can be achieved by use of `GRANT EXECUTE` on the `DBMS_PIPE` package by creating a pipe using the `private` parameter in the `CREATE_PIPE` function and by writing cover packages that only expose particular features or pipenames to particular users or roles.

Depending upon your security requirements, you may choose to use either [Public Pipes](#) or [Private Pipes](#).

## Constants

```
maxwait  constant integer := 86400000; /* 1000 days */
```

This is the maximum time to wait attempting to send or receive a message.

## Operational Notes

Information sent through Oracle pipes is buffered in the system global area (SGA). All information in pipes is lost when the instance is shut down.

---

---

**Caution: Pipes are independent of transactions. Be careful using pipes when transaction control can be affected.**

---

---

The operation of DBMS\_PIPE is considered with regard to the following topics:

- [Public Pipes](#)
- [Writing and Reading Pipes](#)
- [Private Pipes](#)

### Public Pipes

You may create a public pipe either implicitly or explicitly. For *implicit* public pipes, the pipe is automatically created when it is referenced for the first time, and it disappears when it no longer contains data. Because the pipe descriptor is stored in the SGA, there is some space usage overhead until the empty pipe is aged out of the cache.

You create an *explicit* public pipe by calling the CREATE\_PIPE function with the private flag set to FALSE. You must deallocate explicitly-created pipes by calling the REMOVE\_PIPE function.

The domain of a public pipe is the schema in which it was created, either explicitly or implicitly.

### Writing and Reading Pipes

Each public pipe works asynchronously. Any number of schema users can write to a public pipe, as long as they have EXECUTE permission on the DBMS\_PIPE package, and they know the name of the public pipe. However, once buffered information is read by one user, it is emptied from the buffer, and is not available for other readers of the same pipe.

The sending session builds a message using one or more calls to the PACK\_MESSAGE procedure. This procedure adds the message to the session's local message buffer. The information in this buffer is sent by calling the SEND\_MESSAGE function, designating the pipe name to be used to send the message. When SEND\_MESSAGE is called, all messages that have been stacked in the local buffer are sent.

A process that wants to receive a message calls the RECEIVE\_MESSAGE function, designating the pipe name from which to receive the message. The process then calls the UNPACK\_MESSAGE procedure to access each of the items in the message.

### Private Pipes

You explicitly create a private pipe by calling the CREATE\_PIPE function. Once created, the private pipe persists in shared memory until you explicitly deallocate it by calling the REMOVE\_PIPE function. A private pipe is also deallocated when the database instance is shut down.

You cannot create a private pipe if an implicit pipe exists in memory and has the same name as the private pipe you are trying to create. In this case, `CREATE_PIPE` returns an error.

Access to a private pipe is restricted to:

- Sessions running under the same userid as the creator of the pipe
- Stored subprograms executing in the same userid privilege domain as the pipe creator
- Users connected as `SYSDBA`

An attempt by any other user to send or receive messages on the pipe, or to remove the pipe, results in an immediate error. Any attempt by another user to create a pipe with the same name also causes an error.

As with public pipes, you must first build your message using calls to `PACK_MESSAGE` before calling `SEND_MESSAGE`. Similarly, you must call `RECEIVE_MESSAGE` to retrieve the message before accessing the items in the message by calling `UNPACK_MESSAGE`.

## Exceptions

DBMS\_PIPE package subprograms can return the following errors:

**Table 103–1 DBMS\_PIPE Errors**

<b>Error</b>	<b>Description</b>
ORA-23321:	Pipename may not be null. This can be returned by the CREATE_PIPE function, or any subprogram that takes a pipe name as a parameter.
ORA-23322:	Insufficient privilege to access pipe. This can be returned by any subprogram that references a private pipe in its parameter list.

## Examples

- [Example 1: Debugging - PL/SQL](#)
- [Example 3: Execute System Commands](#)
- [Example 4: External Service Interface](#)

### Example 1: Debugging - PL/SQL

This example shows the procedure that a PL/SQL program can call to place debugging information in a pipe.

```
CREATE OR REPLACE PROCEDURE debug (msg VARCHAR2) AS
    status NUMBER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE(LENGTH(msg));
    DBMS_PIPE.PACK_MESSAGE(msg);
    status := DBMS_PIPE.SEND_MESSAGE('plsql_debug');
    IF status != 0 THEN
        raise_application_error(-20099, 'Debug error');
    END IF;
END debug;
```

### Example 2: Debugging - Pro\*C

The following Pro\*C code receives messages from the PLSQL\_DEBUG pipe in the previous example, and displays the messages. If the Pro\*C session is run in a separate window, then it can be used to display any messages that are sent to the debug procedure from a PL/SQL program executing in a separate session.

```
#include <stdio.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR username[20];
    int      status;
    int      msg_length;
    char     retval[2000];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

void sql_error();

main()
{
    -- Prepare username:
    strcpy(username.arr, "SCOTT/TIGER");
    username.len = strlen(username.arr);

    EXEC SQL WHENEVER SQLERROR DO sql_error();
    EXEC SQL CONNECT :username;

    printf("connected\n");

    -- Start an endless loop to look for and print messages on the pipe:
    FOR (;;)

```

```

    {
        EXEC SQL EXECUTE
            DECLARE
                len INTEGER;
                typ INTEGER;
                sta INTEGER;
                chr VARCHAR2(2000);
            BEGIN
                chr := '';
                sta := dbms_pipe.receive_message('plsql_debug');
                IF sta = 0 THEN
                    DBMS_PIPE.UNPACK_MESSAGE(len);
                    DBMS_PIPE.UNPACK_MESSAGE(chr);
                END IF;
                :status := sta;
                :retval := chr;
                IF len IS NOT NULL THEN
                    :msg_length := len;
                ELSE
                    :msg_length := 2000;
                END IF;
            END;
        END-EXEC;
        IF (status == 0)
            printf("\n%.*s\n", msg_length, retval);
        ELSE
            printf("abnormal status, value is %d\n", status);
        }
    }

void sql_error()
{
    char msg[1024];
    int rlen, len;
    len = sizeof(msg);
    sqlglm(msg, &len, &rlen);
    printf("ORACLE ERROR\n");
    printf("%.*s\n", rlen, msg);
    exit(1);
}

```

### Example 3: Execute System Commands

This example shows PL/SQL and Pro\*C code let a PL/SQL stored procedure (or anonymous block) call PL/SQL procedures to send commands over a pipe to a Pro\*C program that is listening for them.

The Pro\*C program sleeps and waits for a message to arrive on the named pipe. When a message arrives, the Pro\*C program processes it, carrying out the required action, such as executing a UNIX command through the *system()* call or executing a SQL command using embedded SQL.

DAEMON.SQL is the source code for the PL/SQL package. This package contains procedures that use the DBMS\_PIPE package to send and receive message to and from the Pro\*C daemon. Note that full handshaking is used. The daemon always sends a message back to the package (except in the case of the STOP command). This is valuable, because it allows the PL/SQL procedures to be sure that the Pro\*C daemon is running.



You can call the DAEMON packaged procedures from an anonymous PL/SQL block using SQL\*Plus or Enterprise Manager. For example:

```
SQLPLUS> variable rv number
SQLPLUS> execute :rv := DAEMON.EXECUTE_SYSTEM('ls -la');
```

On a UNIX system, this causes the Pro\*C daemon to execute the command *system("ls -la")*.

Remember that the daemon needs to be running first. You might want to run it in the background, or in another window beside the SQL\*Plus or Enterprise Manager session from which you call it.

The DAEMON.SQL also uses the DBMS\_OUTPUT package to display the results. For this example to work, you must have execute privileges on this package.

DAEMON.SQL Example. This is the code for the PL/SQL DAEMON package:

```
CREATE OR REPLACE PACKAGE daemon AS
  FUNCTION execute_sql(command VARCHAR2,
                      timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  PROCEDURE stop(timeout NUMBER DEFAULT 10);
END daemon;
/
CREATE OR REPLACE PACKAGE BODY daemon AS

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
    RETURN NUMBER IS

    status      NUMBER;
    result      VARCHAR2(20);
    command_code NUMBER;
    pipe_name   VARCHAR2(30);
  BEGIN
    pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

    DBMS_PIPE.PACK_MESSAGE('SYSTEM');
    DBMS_PIPE.PACK_MESSAGE(pipe_name);
    DBMS_PIPE.PACK_MESSAGE(command);
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20010,
        'Execute_system: Error while sending. Status = ' ||
        status);
    END IF;

    status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20011,
        'Execute_system: Error while receiving.
        Status = ' || status);
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(result);
    IF result <> 'done' THEN
```

```
        RAISE_APPLICATION_ERROR(-20012,
        'Execute_system: Done not received.');
```

---

```
END IF;

DBMS_PIPE.UNPACK_MESSAGE(command_code);
DBMS_OUTPUT.PUT_LINE('System command executed. result = ' ||
                    command_code);
RETURN command_code;
END execute_system;

FUNCTION execute_sql(command VARCHAR2,
                    timeout NUMBER DEFAULT 10)
RETURN NUMBER IS

    status      NUMBER;
    result      VARCHAR2(20);
    command_code NUMBER;
    pipe_name   VARCHAR2(30);

BEGIN
    pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

    DBMS_PIPE.PACK_MESSAGE('SQL');
    DBMS_PIPE.PACK_MESSAGE(pipe_name);
    DBMS_PIPE.PACK_MESSAGE(command);
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
        RAISE_APPLICATION_ERROR(-20020,
        'Execute_sql: Error while sending. Status = ' || status);
    END IF;

    status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);

    IF status <> 0 THEN
        RAISE_APPLICATION_ERROR(-20021,
        'execute_sql: Error while receiving.
        Status = ' || status);
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(result);
    IF result <> 'done' THEN
        RAISE_APPLICATION_ERROR(-20022,
        'execute_sql: done not received.');
```

---

```
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(command_code);
    DBMS_OUTPUT.PUT_LINE
        ('SQL command executed. sqlcode = ' || command_code);
    RETURN command_code;
END execute_sql;

PROCEDURE stop(timeout NUMBER DEFAULT 10) IS
    status NUMBER;
BEGIN
    DBMS_PIPE.PACK_MESSAGE('STOP');
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
        RAISE_APPLICATION_ERROR(-20030,
        'stop: error while sending. status = ' || status);
    END IF;
```

```

    END stop;
END daemon;

```

daemon.pc Example. This is the code for the Pro\*C daemon. You must precompile this using the Pro\*C Precompiler, Version 1.5.x or later. You must also specify the USERID and SQLCHECK options, as the example contains embedded PL/SQL code.

---



---

**Note:** To use a VARCHAR output host variable in a PL/SQL block, you must initialize the length component before entering the block.

---



---

```
proc iname=daemon userid=scott/tiger sqlcheck=semantics
```

Then C-compile and link in the normal way.

```

#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
    char *uid = "scott/tiger";
    int status;
    VARCHAR command[20];
    VARCHAR value[2000];
    VARCHAR return_name[30];
EXEC SQL END DECLARE SECTION;

void
connect_error()
{
    char msg_buffer[512];
    int msg_length;
    int buffer_size = 512;

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlglm(msg_buffer, &buffer_size, &msg_length);
    printf("Daemon error while connecting:\n");
    printf("%. *s\n", msg_length, msg_buffer);
    printf("Daemon quitting.\n");
    exit(1);
}

void
sql_error()
{
    char msg_buffer[512];
    int msg_length;
    int buffer_size = 512;

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlglm(msg_buffer, &buffer_size, &msg_length);
    printf("Daemon error while executing:\n");
    printf("%. *s\n", msg_length, msg_buffer);
    printf("Daemon continuing.\n");
}

main()
{
    command.len = 20; /*initialize length components*/

```

```
value.len = 2000;
return_name.len = 30;
EXEC SQL WHENEVER SQLERROR DO connect_error();
EXEC SQL CONNECT :uid;
printf("Daemon connected.\n");

EXEC SQL WHENEVER SQLERROR DO sql_error();
printf("Daemon waiting...\n");
while (1) {
    EXEC SQL EXECUTE
        BEGIN
            :status := DBMS_PIPE.RECEIVE_MESSAGE('daemon');
            IF :status = 0 THEN
                DBMS_PIPE.UNPACK_MESSAGE(:command);
            END IF;
        END;
    END-EXEC;
    IF (status == 0)
    {
        command.arr[command.len] = '\0';
        IF (!strcmp((char *) command.arr, "STOP"))
        {
            printf("Daemon exiting.\n");
            break;
        }

        ELSE IF (!strcmp((char *) command.arr, "SYSTEM"))
        {
            EXEC SQL EXECUTE
                BEGIN
                    DBMS_PIPE.UNPACK_MESSAGE(:return_name);
                    DBMS_PIPE.UNPACK_MESSAGE(:value);
                END;
            END-EXEC;
            value.arr[value.len] = '\0';
            printf("Will execute system command '%s'\n", value.arr);

            status = system(value.arr);
            EXEC SQL EXECUTE
                BEGIN
                    DBMS_PIPE.PACK_MESSAGE('done');
                    DBMS_PIPE.PACK_MESSAGE(:status);
                    :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
                END;
            END-EXEC;

            IF (status)
            {
                printf
                    ("Daemon error while responding to system command.");
                printf(" status: %d\n", status);
            }
        }
        ELSE IF (!strcmp((char *) command.arr, "SQL")) {
            EXEC SQL EXECUTE
                BEGIN
                    DBMS_PIPE.UNPACK_MESSAGE(:return_name);
                    DBMS_PIPE.UNPACK_MESSAGE(:value);
                END;
            END-EXEC;
```

```

value.arr[value.len] = '\0';
printf("Will execute sql command '%s'\n", value.arr);

EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL EXECUTE IMMEDIATE :value;
status = sqlca.sqlcode;

EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL EXECUTE
  BEGIN
    DBMS_PIPE.PACK_MESSAGE('done');
    DBMS_PIPE.PACK_MESSAGE(:status);
    :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
  END;
END-EXEC;

IF (status)
{
  printf("Daemon error while responding to sql command.");
  printf("  status: %d\n", status);
}
}
ELSE
{
  printf
    ("Daemon error: invalid command '%s' received.\n",
     command.arr);
}
}
ELSE
{
  printf("Daemon error while waiting for signal.");
  printf("  status = %d\n", status);
}
}
EXEC SQL COMMIT WORK RELEASE;
exit(0);

```

#### Example 4: External Service Interface

Put the user-written 3GL code into an OCI or Precompiler program. The program connects to the database and executes PL/SQL code to read its request from the pipe, computes the result, and then executes PL/SQL code to send the result on a pipe back to the requestor.

Below is an example of a stock service request. The recommended sequence for the arguments to pass on the pipe for all service requests is:

protocol_version	VARCHAR2	- '1', 10 bytes or less
returnpipe	VARCHAR2	- 30 bytes or less
service	VARCHAR2	- 30 bytes or less
arg1	VARCHAR2/NUMBER/DATE	
...		
argn	VARCHAR2/NUMBER/DATE	

The recommended format for returning the result is:

success	VARCHAR2	- 'SUCCESS' if OK, otherwise error message
arg1	VARCHAR2/NUMBER/DATE	
...		

```
argn                                VARCHAR2/NUMBER/DATE
```

The "stock price request server" would do, using OCI or PRO\* (in pseudo-code):

```
<loop forever>
BEGIN dbms_stock_server.get_request(:stocksymbol); END;
<figure out price based on stocksymbol (probably from some radio
  signal), set error if can't find such a stock>
BEGIN dbms_stock_server.return_price(:error, :price); END;
```

A client would do:

```
BEGIN :price := stock_request('YOURCOMPANY'); end;
```

The stored procedure, `dbms_stock_server`, which is called by the preceding "stock price request server" is:

```
CREATE OR REPLACE PACKAGE dbms_stock_server IS
  PROCEDURE get_request(symbol OUT VARCHAR2);
  PROCEDURE return_price(errormsg IN VARCHAR2, price IN VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY dbms_stock_server IS
  returnpipe  VARCHAR2(30);

  PROCEDURE returnerror(reason VARCHAR2) IS
    s INTEGER;
  BEGIN
    dbms_pipe.pack_message(reason);
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error: ' || to_char(s) ||
        ' sending on pipe');
    END IF;
  END;

  PROCEDURE get_request(symbol OUT VARCHAR2) IS
    protocol_version VARCHAR2(10);
    s                 INTEGER;
    service           VARCHAR2(30);
  BEGIN
    s := dbms_pipe.receive_message('stock_service');
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error: ' || to_char(s) ||
        ' reading pipe');
    END IF;
    dbms_pipe.unpack_message(protocol_version);
    IF protocol_version <> '1' THEN
      raise_application_error(-20000, 'Bad protocol: ' ||
        protocol_version);
    END IF;
    dbms_pipe.unpack_message(returnpipe);
    dbms_pipe.unpack_message(service);
    IF service != 'getprice' THEN
      returnerror('Service ' || service || ' not supported');
    END IF;
    dbms_pipe.unpack_message(symbol);
  END;

  PROCEDURE return_price(errormsg in VARCHAR2, price in VARCHAR2) IS
    s INTEGER;
```

```

BEGIN
  IF errormsg IS NULL THEN
    dbms_pipe.pack_message('SUCCESS');
    dbms_pipe.pack_message(price);
  ELSE
    dbms_pipe.pack_message(errormsg);
  END IF;
  s := dbms_pipe.send_message(returnpipe);
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error: ' || to_char(s) ||
      ' sending on pipe');
  END IF;
END;

```

The procedure called by the client is:

```

CREATE OR REPLACE FUNCTION stock_request (symbol VARCHAR2)
RETURN VARCHAR2 IS
  s      INTEGER;
  price  VARCHAR2(20);
  errormsg VARCHAR2(512);
BEGIN
  dbms_pipe.pack_message('1'); -- protocol version
  dbms_pipe.pack_message(dbms_pipe.unique_session_name); -- return pipe
  dbms_pipe.pack_message('getprice');
  dbms_pipe.pack_message(symbol);
  s := dbms_pipe.send_message('stock_service');
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error: ' || to_char(s) ||
      ' sending on pipe');
  END IF;
  s := dbms_pipe.receive_message(dbms_pipe.unique_session_name);
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error: ' || to_char(s) ||
      ' receiving on pipe');
  END IF;
  dbms_pipe.unpack_message(errormsg);
  IF errormsg <> 'SUCCESS' THEN
    raise_application_error(-20000, errormsg);
  END IF;
  dbms_pipe.unpack_message(price);
  RETURN price;
END;

```

You would typically only GRANT EXECUTE on DBMS\_STOCK\_SERVICE to the stock service application server, and would only GRANT EXECUTE on stock\_request to those users allowed to use the service.

**See Also:** [Chapter 19, "DBMS\\_ALERT"](#)

---

## Summary of DBMS\_PIPE Subprograms

**Table 103–2 DBMS\_PIPE Package Subprograms**

Subprogram	Description
<a href="#">CREATE_PIPE Function</a> on page 103-19	Creates a pipe (necessary for private pipes)
<a href="#">NEXT_ITEM_TYPE Function</a> on page 103-21	Returns datatype of next item in buffer
<a href="#">PACK_MESSAGE Procedures</a> on page 103-22	Builds message in local buffer
<a href="#">PURGE Procedure</a> on page 103-24	Purges contents of named pipe
<a href="#">RECEIVE_MESSAGE Function</a> on page 103-25	Copies message from named pipe into local buffer
<a href="#">REMOVE_PIPE Function</a> on page 103-28	Removes the named pipe
<a href="#">RESET_BUFFER Procedure</a> on page 103-27	Purges contents of local buffer
<a href="#">SEND_MESSAGE Function</a> on page 103-29	Sends message on named pipe: This implicitly creates a public pipe if the named pipe does not exist
<a href="#">UNIQUE_SESSION_NAME Function</a> on page 103-31	Returns unique session name
<a href="#">UNPACK_MESSAGE Procedures</a> on page 103-32	Accesses next item in buffer



## CREATE\_PIPE Function

This function explicitly creates a public or private pipe. If the `private` flag is `TRUE`, then the pipe creator is assigned as the owner of the private pipe.

Explicitly-created pipes can only be removed by calling `REMOVE_PIPE`, or by shutting down the instance.

### Syntax

```
DBMS_PIPE.CREATE_PIPE (
    pipename      IN VARCHAR2,
    maxpipesize   IN INTEGER DEFAULT 8192,
    private       IN BOOLEAN DEFAULT TRUE)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(create_pipe,WNDS,RNDS);
```

### Parameters

**Table 103–3 CREATE\_PIPE Function Parameters**

Parameter	Description
<code>pipename</code>	<p>Name of the pipe you are creating.</p> <p>You must use this name when you call <code>SEND_MESSAGE</code> and <code>RECEIVE_MESSAGE</code>. This name must be unique across the instance.</p> <p>Caution: Do not use pipe names beginning with <code>ORA\$</code>. These are reserved for use by procedures provided by Oracle. Pipename should not be longer than 128 bytes, and is case insensitive. At this time, the name cannot contain Globalization Support characters.</p>
<code>maxpipesize</code>	<p>The maximum size allowed for the pipe, in bytes.</p> <p>The total size of all of the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default <code>maxpipesize</code> is 8192 bytes.</p> <p>The <code>maxpipesize</code> for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of <code>SEND_MESSAGE</code> with larger values cause the <code>maxpipesize</code> to be increased. Callers with a smaller value use the existing, larger value.</p>
<code>private</code>	<p>Uses the default, <code>TRUE</code>, to create a private pipe.</p> <p>Public pipes can be implicitly created when you call <code>SEND_MESSAGE</code>.</p>

## Return Values

**Table 103–4 CREATE\_PIPE Function Return Values**

Return	Description
0	Successful.  If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains.  If a user connected as SYSDBA/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.
ORA-23322	Failure due to naming conflict.  If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

## Exceptions

**Table 103–5 CREATE\_PIPE Function Exception**

Exception	Description
Null pipe name	Permission error: Pipe with the same name already exists, and you are not allowed to use it.

## NEXT\_ITEM\_TYPE Function

This function determines the datatype of the next item in the local message buffer.

After you have called `RECEIVE_MESSAGE` to place pipe information in a local buffer, call `NEXT_ITEM_TYPE`.

### Syntax

```
DBMS_PIPE.NEXT_ITEM_TYPE  
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (next_item_type, WNDS, RNDS);
```

### Return Values

**Table 103–6** *NEXT\_ITEM\_TYPE* Function Return Values

Return	Description
0	No more items
6	NUMBER
9	VARCHAR2
11	ROWID
12	DATE
23	RAW

## PACK\_MESSAGE Procedures

This procedure builds your message in the local message buffer. To send a message, first make one or more calls to `PACK_MESSAGE`. Then, call `SEND_MESSAGE` to send the message in the local buffer on the named pipe.

The procedure is overloaded to accept items of type `VARCHAR2`, `NCHAR`, `NUMBER`, `DATE`, `RAW` and `ROWID` items. In addition to the data bytes, each item in the buffer requires one byte to indicate its type, and two bytes to store its length. One additional byte is needed to terminate the message. The overhead for all types other than `VARCHAR` is 4 bytes.

### Syntax

```
DBMS_PIPE.PACK_MESSAGE (
    item IN VARCHAR2);
```

```
DBMS_PIPE.PACK_MESSAGE (
    item IN NCHAR);
```

```
DBMS_PIPE.PACK_MESSAGE (
    item IN NUMBER);
```

```
DBMS_PIPE.PACK_MESSAGE (
    item IN DATE);
```

```
DBMS_PIPE.PACK_MESSAGE_RAW (
    item IN RAW);
```

```
DBMS_PIPE.PACK_MESSAGE_ROWID (
    item IN ROWID);
```

### Pragmas

```
pragma restrict_references(pack_message,WNDS,RNDS);
pragma restrict_references(pack_message_raw,WNDS,RNDS);
pragma restrict_references(pack_message_rowid,WNDS,RNDS);
```

### Parameters

**Table 103–7 PACK\_MESSAGE Procedure Parameters**

Parameter	Description
item	Item to pack into the local message buffer.

### Usage Notes

In Oracle database version 8.x, the char-set-id (2 bytes) and the char-set-form (1 byte) are stored with each data item. Therefore, the overhead when using Oracle database version 8.x is 7 bytes.

When you call `SEND_MESSAGE` to send this message, you must indicate the name of the pipe on which you want to send the message. If this pipe already exists, then you must have sufficient privileges to access this pipe. If the pipe does not already exist, then it is created automatically.

**Exceptions**

ORA-06558 is raised if the message buffer overflows (currently 4096 bytes). Each item in the buffer takes one byte for the type, two bytes for the length, plus the actual data. There is also one byte needed to terminate the message.

## PURGE Procedure

This procedure empties the contents of the named pipe.

An empty implicitly-created pipe is aged out of the shared global area according to the least-recently-used algorithm. Thus, calling `PURGE` lets you free the memory associated with an implicitly-created pipe.

### Syntax

```
DBMS_PIPE.PURGE (  
    pipename IN VARCHAR2);
```

### Pragmas

```
pragma restrict_references (purge, WNDS, RNDS);
```

### Parameters

**Table 103–8** *PURGE Procedure Parameters*

Parameter	Description
<code>pipename</code>	Name of pipe from which to remove all messages.  The local buffer may be overwritten with messages as they are discarded. Pipename should not be longer than 128 bytes, and is case-insensitive.

### Usage Notes

Because `PURGE` calls `RECEIVE_MESSAGE`, the local buffer might be overwritten with messages as they are purged from the pipe. Also, you can receive an `ORA-23322` (insufficient privileges) error if you attempt to purge a pipe with which you have insufficient access rights.

### Exceptions

Permission error if pipe belongs to another user.

## RECEIVE\_MESSAGE Function

This function copies the message into the local message buffer.

### Syntax

```
DBMS_PIPE.RECEIVE_MESSAGE (
    pipename      IN VARCHAR2,
    timeout       IN INTEGER      DEFAULT maxwait)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (receive_message, WNDS, RNDS);
```

### Parameters

**Table 103–9 RECEIVE\_MESSAGE Function Parameters**

Parameter	Description
pipename	Name of the pipe on which you want to receive a message. Names beginning with ORA\$ are reserved for use by Oracle
timeout	Time to wait for a message, in seconds.  The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days). A timeout of 0 lets you read without blocking.

### Return Values

**Table 103–10 RECEIVE\_MESSAGE Function Return Values**

Return	Description
0	Success
1	Timed out. If the pipe was implicitly-created and is empty, then it is removed.
2	Record in the pipe is too large for the buffer. (This should not happen.)
3	An interrupt occurred.
ORA-23322	User has insufficient privileges to read from the pipe.

### Usage Notes

To receive a message from a pipe, first call `RECEIVE_MESSAGE`. When you receive a message, it is removed from the pipe; hence, a message can only be received once. For implicitly-created pipes, the pipe is removed after the last record is removed from the pipe.

If the pipe that you specify when you call `RECEIVE_MESSAGE` does not already exist, then Oracle implicitly creates the pipe and waits to receive the message. If the message does not arrive within a designated timeout interval, then the call returns and the pipe is removed.

After receiving the message, you must make one or more calls to `UNPACK_MESSAGE` to access the individual items in the message. The `UNPACK_MESSAGE` procedure is

overloaded to unpack items of type DATE, NUMBER, VARCHAR2, and there are two additional procedures to unpack RAW and ROWID items. If you do not know the type of data that you are attempting to unpack, then call NEXT\_ITEM\_TYPE to determine the type of the next item in the buffer.

## Exceptions

**Table 103–11** *RECEIVE\_MESSAGE Function Exceptions*

<b>Exception</b>	<b>Description</b>
Null pipe name	Permission error. Insufficient privilege to remove the record from the pipe. The pipe is owned by someone else.



## RESET\_BUFFER Procedure

This procedure resets the `PACK_MESSAGE` and `UNPACK_MESSAGE` positioning indicators to 0.

Because all pipes share a single buffer, you may find it useful to reset the buffer before using a new pipe. This ensures that the first time you attempt to send a message to your pipe, you do not inadvertently send an expired message remaining in the buffer.

### Syntax

```
DBMS_PIPE.RESET_BUFFER;
```

### Pragmas

```
pragma restrict_references(reset_buffer,WNDS,RNDS);
```

## REMOVE\_PIPE Function

This function removes explicitly-created pipes.

Pipes created implicitly by `SEND_MESSAGE` are automatically removed when empty. However, pipes created explicitly by `CREATE_PIPE` are removed only by calling `REMOVE_PIPE`, or by shutting down the instance. All unconsumed records in the pipe are removed before the pipe is deleted.

This is similar to calling `PURGE` on an implicitly-created pipe.

### Syntax

```
DBMS_PIPE.REMOVE_PIPE (
    pipename IN VARCHAR2)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(remove_pipe,WNDS,RNDS);
```

### Parameters

**Table 103–12 REMOVE\_PIPE Function Parameters**

Parameter	Description
pipename	Name of pipe that you want to remove.

### Return Values

**Table 103–13 REMOVE\_PIPE Function Return Values**

Return	Description
0	Success  If the pipe does not exist, or if the pipe already exists and the user attempting to remove it is authorized to do so, then Oracle returns 0, indicating success, and any data remaining in the pipe is removed.
ORA-23322	Insufficient privileges.  If the pipe exists, but the user is not authorized to access the pipe, then Oracle signals error ORA-23322, indicating insufficient privileges.

### Exceptions

**Table 103–14 REMOVE\_PIPE Function Exception**

Exception	Description
Null pipe name	Permission error: Insufficient privilege to remove pipe. The pipe was created and is owned by someone else.

## SEND\_MESSAGE Function

This function sends a message on the named pipe.

The message is contained in the local message buffer, which was filled with calls to `PACK_MESSAGE`. You can create a pipe explicitly using `CREATE_PIPE`, otherwise, it is created implicitly.

### Syntax

```
DBMS_PIPE.SEND_MESSAGE (
    pipename      IN VARCHAR2,
    timeout       IN INTEGER DEFAULT MAXWAIT,
    maxpipesize   IN INTEGER DEFAULT 8192)
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (send_message, WNDS, RNDS);
```

### Parameters

**Table 103–15 SEND\_MESSAGE Function Parameters**

Parameter	Description
<code>pipename</code>	<p>Name of the pipe on which you want to place the message.</p> <p>If you are using an explicit pipe, then this is the name that you specified when you called <code>CREATE_PIPE</code>.</p> <p>Caution: Do not use pipe names beginning with 'ORA\$'. These names are reserved for use by procedures provided by Oracle. Pipename should not be longer than 128 bytes, and is case-insensitive. At this time, the name cannot contain Globalization Support characters.</p>
<code>timeout</code>	<p>Time to wait while attempting to place a message on a pipe, in seconds.</p> <p>The default value is the constant <code>MAXWAIT</code>, which is defined as 86400000 (1000 days).</p>
<code>maxpipesize</code>	<p>Maximum size allowed for the pipe, in bytes.</p> <p>The total size of all the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default is 8192 bytes.</p> <p>The <code>maxpipesize</code> for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of <code>SEND_MESSAGE</code> with larger values cause the <code>maxpipesize</code> to be increased. Callers with a smaller value simply use the existing, larger value.</p> <p>Specifying <code>maxpipesize</code> as part of the <code>SEND_MESSAGE</code> procedure eliminates the need for a separate call to open the pipe. If you created the pipe explicitly, then you can use the optional <code>maxpipesize</code> parameter to override the creation pipe size specifications.</p>

## Return Values

**Table 103–16 SEND\_MESSAGE Function Return Values**

Return	Description
0	<p>Success.</p> <p>If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains.</p> <p>If a user connected as SYSDBS/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.</p>
1	<p>Timed out.</p> <p>This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used. If the pipe was implicitly-created and is empty, then it is removed.</p>
3	<p>An interrupt occurred.</p> <p>If the pipe was implicitly created and is empty, then it is removed.</p>
ORA-23322	<p>Insufficient privileges.</p> <p>If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.</p>

## Exceptions

**Table 103–17 SEND\_MESSAGE Function Exception**

Exception	Description
Null pipe name	Permission error. Insufficient privilege to write to the pipe. The pipe is private and owned by someone else.

## UNIQUE\_SESSION\_NAME Function

This function receives a name that is unique among all of the sessions that are currently connected to a database.

Multiple calls to this function from the same session always return the same value. You might find it useful to use this function to supply the `PIPENAME` parameter for your `SEND_MESSAGE` and `RECEIVE_MESSAGE` calls.

### Syntax

```
DBMS_PIPE.UNIQUE_SESSION_NAME  
RETURN VARCHAR2;
```

### Pragmas

```
pragma restrict_references(unique_session_name,WNDS,RNDS,WNPS);
```

### Return Values

This function returns a unique name. The returned name can be up to 30 bytes.

## UNPACK\_MESSAGE Procedures

This procedure retrieves items from the buffer.

After you have called `RECEIVE_MESSAGE` to place pipe information in a local buffer, call `UNPACK_MESSAGE`.

---



---

**Note:** The `UNPACK_MESSAGE` procedure is overloaded to return items of type `VARCHAR2`, `NCHAR`, `NUMBER`, or `DATE`. There are two additional procedures to unpack `RAW` and `ROWID` items.

---



---

### Syntax

```
DBMS_PIPE.UNPACK_MESSAGE (
    item OUT VARCHAR2);
```

```
DBMS_PIPE.UNPACK_MESSAGE (
    item OUT NCHAR);
```

```
DBMS_PIPE.UNPACK_MESSAGE (
    item OUT NUMBER);
```

```
DBMS_PIPE.UNPACK_MESSAGE (
    item OUT DATE);
```

```
DBMS_PIPE.UNPACK_MESSAGE_RAW (
    item OUT RAW);
```

```
DBMS_PIPE.UNPACK_MESSAGE_ROWID (
    item OUT ROWID);
```

### Pragmas

```
pragma restrict_references (unpack_message, WNDS, RNDS);
pragma restrict_references (unpack_message_raw, WNDS, RNDS);
pragma restrict_references (unpack_message_rowid, WNDS, RNDS);
```

### Parameters

**Table 103–18 UNPACK\_MESSAGE Procedure Parameters**

Parameter	Description
<code>item</code>	Argument to receive the next unpacked item from the local message buffer.

### Exceptions

ORA-06556 or 06559 are generated if the buffer contains no more items, or if the item is not of the same type as that requested.

---

---

## DBMS\_PREDICTIVE\_ANALYTICS

Data mining can discover useful information buried in vast amounts of data. However, it is often the case that both the programming interfaces and the data mining expertise required to obtain these results are too complex for use by the wide audiences that can obtain benefits from using Oracle Data Mining.

The `DBMS_PREDICTIVE_ANALYTICS` package addresses both of these complexities by automating the entire data mining process from data preprocessing through model building to scoring new data. This package provides an important tool that makes data mining possible for a broad audience of users, in particular, business analysts.

**See Also:** *Oracle Data Mining Concepts* for an overview of Oracle predictive analytics, including information about the Oracle Spreadsheet Add-In for Predictive Analytics.

This chapter contains the following topics:

- [Using DBMS\\_PREDICTIVE\\_ANALYTICS](#)
  - [Overview](#)
  - [Security Model](#)
- [Summary of DBMS\\_PREDICTIVE\\_ANALYTICS Subprograms](#)

---

## Using DBMS\_PREDICTIVE\_ANALYTICS

This section contains topics that relate to using the DBMS\_PREDICTIVE\_ANALYTICS package.

- [Overview](#)
- [Security Model](#)



## Overview

Data mining, according to a commonly used process model, requires the following steps:

1. Understand the business problem.
2. Understand the data.
3. Prepare the data for mining.
4. Create models using the prepared data.
5. Evaluate the models.
6. Deploy and use the model to score new data.

DBMS\_PREDICTIVE\_ANALYTICS automates parts of step 3 — 5 of this process.

Predictive analytics procedures analyze and prepare the input data, create and test mining models using the input data, and then use the input data for scoring. The results of scoring are returned to the user. The models and supporting objects are not preserved after the operation completes.

## Security Model

The `DBMS_PREDICTIVE_ANALYTICS` package is owned by user `SYS` and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

The `DBMS_PREDICTIVE_ANALYTICS` package exposes APIs which are leveraged by the Oracle Data Mining option. Users who wish to invoke procedures in this package require the `CREATE MINING MODEL` system privilege (as well as the `CREATE TABLE` and `CREATE VIEW` system privilege).

## Summary of DBMS\_PREDICTIVE\_ANALYTICS Subprograms

**Table 104-1** *DBMS\_PREDICTIVE\_ANALYTICS Package Subprograms*

<b>Subprogram</b>	<b>Purpose</b>
<a href="#">EXPLAIN Procedure</a> on page 104-6	Ranks attributes in order of influence in explaining a target column.
<a href="#">PREDICT Procedure</a> on page 104-8	Predicts the value of a target column based on values in the input data.
<a href="#">PROFILE Procedure</a> on page 104-10	Generates rules that identify the records that have the same target value.

## EXPLAIN Procedure

The `EXPLAIN` procedure identifies the attributes that are important in explaining the variation in values of a target column.

The input data must contain some records where the target value is known (not `NULL`). These records are used by the procedure to train a model that calculates the attribute importance.

---

**Note:** `EXPLAIN` supports `DATE` and `TIMESTAMP` data types in addition to the numeric, character, and nested data types supported by Oracle Data Mining models.

Data requirements for Oracle Data Mining are described in *Oracle Data Mining Application Developer's Guide*.

---

The `EXPLAIN` procedure creates a result table that lists the attributes in order of their explanatory power. The result table is described in the Usage Notes.

### Syntax

```
DBMS_PREDICTIVE_ANALYTICS.EXPLAIN (
    data_table_name      IN VARCHAR2,
    explain_column_name  IN VARCHAR2,
    result_table_name    IN VARCHAR2,
    data_schema_name     IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 104–2** *EXPLAIN Procedure Parameters*

Parameter	Description
<code>data_table_name</code>	Name of input table or view
<code>explain_column_name</code>	Name of the column to be explained
<code>result_table_name</code>	Name of the table where results are saved
<code>data_schema_name</code>	Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.

### Usage Notes

The `EXPLAIN` procedure creates a result table with the columns described in [Table 104–3](#).

**Table 104–3 EXPLAIN Procedure Result Table**

Column Name	Data Type	Description
ATTRIBUTE_NAME	VARCHAR2 (30)	Name of a column in the input data; all columns except the explained column are listed in the result table.
EXPLANATORY_VALUE	NUMBER	Value indicating how useful the column is for determining the value of the explained column. Higher values indicate greater explanatory power. Value can range from 0 to 1.  An individual column's explanatory value is independent of other columns in the input table. The values are based on how strong each individual column correlates with the explained column. The value is affected by the number of records in the input table, and the relations of the values of the column to the values of the explain column.  An explanatory power value of 0 implies there is no useful correlation between the column's values and the explain column's values. An explanatory power of 1 implies perfect correlation; such columns should be eliminated from consideration for PREDICT. In practice, an explanatory power equal to 1 is rarely returned.
RANK	NUMBER	Ranking of explanatory power. Rows with equal values for explanatory_value have the same rank. Rank values are not skipped in the event of ties.

## Example

The following example performs an EXPLAIN operation on the SUPPLEMENTARY\_DEMOGRAPHICS table of Sales History.

```
--Perform EXPLAIN operation
BEGIN
  DBMS_PREDICTIVE_ANALYTICS.EXPLAIN(
    data_table_name => 'supplementary_demographics',
    explain_column_name => 'home_theater_package',
    result_table_name => 'demographics_explain_result');
END;
/
--Display results
SELECT * FROM demographics_explain_result;
```

ATTRIBUTE_NAME	EXPLANATORY_VALUE	RANK
Y_BOX_GAMES	.524311073	1
YRS_RESIDENCE	.495987246	2
HOUSEHOLD_SIZE	.146208506	3
AFFINITY_CARD	.0598227	4
EDUCATION	.018462703	5
OCCUPATION	.009721543	6
FLAT_PANEL_MONITOR	.00013733	7
PRINTER_SUPPLIES	0	8
OS_DOC_SET_KANJI	0	8
BULK_PACK_DISKETTES	0	8
BOOKKEEPING_APPLICATION	0	8
COMMENTS	0	8
CUST_ID	0	8

The results show that Y\_BOX\_GAMES, YRS\_RESIDENCE, and HOUSEHOLD\_SIZE are the best predictors of HOME\_THEATER\_PACKAGE.

## PREDICT Procedure

The PREDICT procedure predicts the values of a target column.

The input data must contain some records where the target value is known (not NULL). These records are used by the procedure to train and test a model that makes the predictions.

---

**Note:** PREDICT supports DATE and TIMESTAMP data types in addition to the numeric, character, and nested data types supported by Oracle Data Mining models.

Data requirements for Oracle Data Mining are described in *Oracle Data Mining Application Developer's Guide*.

---

The PREDICT procedure creates a result table that contains a predicted target value for every record. The result table is described in the Usage Notes.

### Syntax

```
DBMS_PREDICTIVE_ANALYTICS.PREDICT (
  accuracy          OUT NUMBER,
  data_table_name   IN VARCHAR2,
  case_id_column_name IN VARCHAR2,
  target_column_name IN VARCHAR2,
  result_table_name IN VARCHAR2,
  data_schema_name  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 104–4 PREDICT Procedure Parameters**

Parameter	Description
accuracy	Output parameter that returns the predictive confidence, a measure of the accuracy of the predicted values. The predictive confidence for a categorical target is the most common target value; the predictive confidence for a numerical target is the mean.
data_table_name	Name of the input table or view.
case_id_column_name	Name of the column that uniquely identifies each case (record) in the input data.
target_column_name	Name of the column to predict.
result_table_name	Name of the table where results will be saved.
data_schema_name	Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.

### Usage Notes

The PREDICT procedure creates a result table with the columns described in [Table 104–5](#).

**Table 104–5 PREDICT Procedure Result Table**

Column Name	Data Type	Description
Case ID column name	VARCHAR2 or NUMBER	The name of the case ID column in the input data.
PREDICTION	VARCHAR2 or NUMBER	The predicted value of the target column for the given case.
PROBABILITY	NUMBER	For classification (categorical target), the probability of the prediction. For regression problems (numerical target), this column contains NULL.

---

**Note:** Make sure that the name of the case ID column is not 'PREDICTION' or 'PROBABILITY'.

---

Predictions are returned for all cases whether or not they contained target values in the input.

Predicted values for known cases may be interesting in some situations. For example, you could perform deviation analysis to compare predicted values and actual values.

## Example

The following example performs a PREDICT operation and displays the first 10 predictions. The results show an accuracy of 79% in predicting whether each customer has an affinity card.

```
--Perform PREDICT operation
DECLARE
    v_accuracy NUMBER(10,9);
BEGIN
    DBMS_PREDICTIVE_ANALYTICS.PREDICT(
        accuracy          => v_accuracy,
        data_table_name   => 'supplementary_demographics',
        case_id_column_name => 'cust_id',
        target_column_name => 'affinity_card',
        result_table_name => 'pa_demographics_predict_result');
    DBMS_OUTPUT.PUT_LINE('Accuracy = ' || v_accuracy);
END;
/
```

Accuracy = .788696903

```
--Display results
SELECT * FROM pa_demographics_predict_result WHERE rownum < 10;
```

```

CUST_ID PREDICTION PROBABILITY
-----
101501          1 .834069848
101502          0 .991269965
101503          0 .99978311
101504          1 .971643388
101505          1 .541754127
101506          0 .803719133
101507          0 .999999303
101508          0 .999999987
101509          0 .999953074
```

## PROFILE Procedure

The `PROFILE` procedure generates rules that describe the cases (records) from the input data. For example, if a target column `CHURN` has values 'Yes' and 'No', `PROFILE` generates a set of rules describing the expected outcomes. Each profile includes a rule, record count, and a score distribution.

The input data must contain some cases where the target value is known (not `NULL`). These cases are used by the procedure to build a model that calculates the rules.

---

**Note:** `PROFILE` does not support nested types or dates.

Data requirements for Oracle Data Mining are described in *Oracle Data Mining Application Developer's Guide*.

---

The `PROFILE` procedure creates a result table that specifies rules (profiles) and their corresponding target values. The result table is described in the Usage Notes.

### Syntax

```
DBMS_PREDICTIVE_ANALYTICS.PROFILE (
  data_table_name          IN VARCHAR2,
  target_column_name      IN VARCHAR2,
  result_table_name       IN VARCHAR2,
  data_schema_name        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 104–6 PROFILE Procedure Parameters**

Parameter	Description
<code>data_table_name</code>	Name of the table containing the data to be analyzed.
<code>target_column_name</code>	Name of the target column.
<code>result_table_name</code>	Name of the table where the results will be saved.
<code>data_schema_name</code>	Name of the schema where the input table or view resides and where the result table is created. Default: the current schema.

### Usage Notes

The `PROFILE` procedure creates a result table with the columns described in [Table 104–7](#).

**Table 104–7 PROFILE Procedure Result Table**

Column Name	Data Type	Description
<code>PROFILE_ID</code>	NUMBER	A unique identifier for this profile (rule).
<code>RECORD_COUNT</code>	NUMBER	The number of records described by the profile.
<code>DESCRIPTION</code>	<code>SYS.XMLTYPE</code>	The profile rule. See " <a href="#">XML Schema for Profile Rules</a> ".

#### XML Schema for Profile Rules

The `DESCRIPTION` column of the result table contains XML that conforms to the following XSD:



```

<xs:element name="SimpleRule">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="PREDICATE"/>
      <xs:element ref="ScoreDistribution" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="optional"/>
    <xs:attribute name="score" type="xs:string" use="required"/>
    <xs:attribute name="recordCount" type="NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>

```

## Example

This example generates a rule describing customers who are likely to use an affinity card (target value is 1) and a set of rules describing customers who are not likely to use an affinity card (target value is 0). The rules are based on only two predictors: education and occupation.

```

SET serveroutput ON
SET trimspool ON
SET pages 10000
SET long 10000
SET pagesize 10000
SET linesize 150
CREATE VIEW cust_edu_occ_view AS
    SELECT cust_id, education, occupation, affinity_card
    FROM sh.supplementary_demographics;
BEGIN
  DBMS_PREDICTIVE_ANALYTICS.PROFILE(
    DATA_TABLE_NAME => 'cust_edu_occ_view',
    TARGET_COLUMN_NAME => 'affinity_card',
    RESULT_TABLE_NAME => 'profile_result');
END;
/

```

This example generates eight rules in the result table `profile_result`. Seven of the rules suggest a target value of 0; one rule suggests a target value of 1. The `score` attribute on a rule identifies the target value.

This `SELECT` statement returns all the rules in the result table.

```

SELECT a.profile_id, a.record_count, a.description.getstringval()
FROM profile_result a;

```

This `SELECT` statement returns the rules for a target value of 0.

```

SELECT *
FROM profile_result t
WHERE extractvalue(t.description, '/SimpleRule/@score') = 0;

```

To obtain more readable output, you can cut and paste the XML for a rule into a text file, save it with the `.xml` extension, and view the rule in a browser. The eight rules generated by this example are displayed as follows.

```

<SimpleRule id="1" score="0" recordCount="443">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
    </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">

```

```

    <Array type="string">< Bach." "Assoc-V" "HS-grad"
    </Array>
  </SimpleSetPredicate>
</CompoundPredicate>
<ScoreDistribution value="0" recordCount="297" />
<ScoreDistribution value="1" recordCount="146" />
</SimpleRule>

<SimpleRule id="2" score="0" recordCount="18">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"10th" "11th" "12th" "1st-4th" "5th-6th" "7th-8th" "9th" "Presch."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="18" />
</SimpleRule>

<SimpleRule id="3" score="0" recordCount="458">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Assoc-A" "Bach."
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="248" />
  <ScoreDistribution value="1" recordCount="210" />
</SimpleRule>

<SimpleRule id="4" score="1" recordCount="276">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Armed-F" "Exec." "Prof." "Protec."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Masters" "PhD" "Profsc"
      </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="1" recordCount="183" />
  <ScoreDistribution value="0" recordCount="93" />
</SimpleRule>

<SimpleRule id="5" score="0" recordCount="307">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Assoc-A" "Bach." "Masters" "PhD" "Profsc"
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Crafts" "Sales" "TechSup" "Transp."

```

```

    </Array>
  </SimpleSetPredicate>
</CompoundPredicate>
<ScoreDistribution value="0" recordCount="184" />
<ScoreDistribution value="1" recordCount="123" />
</SimpleRule>

<SimpleRule id="6" score="0" recordCount="243">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">"Assoc-A" "Bach." "Masters" "PhD" "Profsc"
    </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"?" "Cleric." "Farming" "Handler" "House-s" "Machine" "Other"
    </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="197" />
  <ScoreDistribution value="1" recordCount="46" />
</SimpleRule>

<SimpleRule id="7" score="0" recordCount="2158">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">
        "10th" "11th" "12th" "1st-4th" "5th-6th" "7th-8th" "9th" "< Bach." "Assoc-V" "HS-grad"
        "Presch."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"?" "Cleric." "Crafts" "Farming" "Machine" "Sales" "TechSup" " Transp."
    </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="1819"/>
  <ScoreDistribution value="1" recordCount="339"/>
</SimpleRule>

<SimpleRule id="8" score="0" recordCount="597">
  <CompoundPredicate booleanOperator="and">
    <SimpleSetPredicate field="EDUCATION" booleanOperator="isIn">
      <Array type="string">
        "10th" "11th" "12th" "1st-4th" "5th-6th" "7th-8th" "9th" "< Bach." "Assoc-V" "HS-grad"
        "Presch."
      </Array>
    </SimpleSetPredicate>
    <SimpleSetPredicate field="OCCUPATION" booleanOperator="isIn">
      <Array type="string">"Handler" "House-s" "Other"
    </Array>
    </SimpleSetPredicate>
  </CompoundPredicate>
  <ScoreDistribution value="0" recordCount="572"/>
  <ScoreDistribution value="1" recordCount="25"/>
</SimpleRule>

```



---

---

## DBMS\_PREPROCESSOR

The DBMS\_PREPROCESSOR package provides an interface to print or retrieve the source text of a PL/SQL unit in its post-processed form.

This package contains the following topics

- [Using DBMS\\_PREPROCESSOR](#)
  - Overview
  - Operating Notes
- [Data Structures](#)
  - Table Types
- [Summary of DBMS\\_PREPROCESSOR Subprograms](#)

## Using DBMS\_PREPROCESSOR

- [Overview](#)
- [Operating Notes](#)

## Overview

There are three styles of subprograms.

1. Subprograms that take a schema name, a unit type name, and the unit name.
2. Subprograms that take a `VARCHAR2` string which contains the source text of an arbitrary PL/SQL compilation unit.
3. Subprograms that take a `VARCHAR2` index-by table which contains the segmented source text of an arbitrary PL/SQL compilation unit.

Subprograms of the first style are used to print or retrieve the post-processed source text of a stored PL/SQL unit. The user must have the privileges necessary to view the original source text of this unit. The user must also specify the schema in which the unit is defined, the type of the unit, and the name of the unit. If the schema is null, then the current user schema is used. If the status of the stored unit is `VALID` and the user has the required privilege, then the post-processed source text is guaranteed to be the same as that of the unit the last time it was compiled.

Subprograms of the second or third style are used to generate post-processed source text in the current user schema. The source text is passed in as a single `VARCHAR2` string in the second style, or as a `VARCHAR2` index-by table in the third style. The source text can represent an arbitrary PL/SQL compilation unit. A typical usage is to pass the source text of an anonymous block and generate its post-processed source text in the current user schema. The third style can be useful when the source text exceeds the `VARCHAR2` length limit.

## Operating Notes

- For subprograms of the first style, the status of the stored PL/SQL unit does not need to be `VALID`. Likewise, the source text passed in as a `VARCHAR2` string or a `VARCHAR2` index-by table may contain compile time errors. If errors are found when generating the post-processed source, the error message text will also appear at the end of the post-processed source text. In some cases, the preprocessing can be aborted because of errors. When this happens, the post-processed source text will appear to be incomplete and the associated error message can help to indicate that an error has occurred during preprocessing.
- For subprograms of the second or third style, the source text can represent any arbitrary PL/SQL compilation unit. However, the source text of a valid PL/SQL compilation unit cannot include commonly used prefixes such as `CREATE` or `REPLACE`. In general, the input source should be syntactically prepared in a way as if it were obtained from the `ALL_SOURCE` view. The following list gives some examples of valid initial syntax for some PL/SQL compilation units.

anonymous block	(BEGIN   DECLARE) ...
package	PACKAGE <name> ...
package body	PACKAGE BODY <name> ...
procedure	PROCEDURE <name> ...
function	FUNCTION <name> ...
type	TYPE <name> ...
type body	TYPE BODY <name> ...
trigger	(BEGIN   DECLARE) ...

If the source text represents a named PL/SQL unit that is valid, that unit will not be created after its post-processed source text is generated.

- If the text of a wrapped PL/SQL unit is obtained from the `ALL_SOURCE` view, the keyword `WRAPPED` always immediately follows the name of the unit, as in this example:

```
PROCEDURE "some proc" WRAPPED
a000000
b2
...
```

If such source text is presented to one of the [GET\\_POST\\_PROCESSED\\_SOURCE Functions](#) or to one of the [PRINT\\_POST\\_PROCESSED\\_SOURCE Procedures](#), the exception `DBMS_PREPROCESSOR.WRAPPED_INPUT` is raised.



## Data Structures

The DBMS\_PREPROCESSOR package defines a TABLE type.

### Table Types

[SOURCE\\_LINES\\_T Table Type](#)

## SOURCE\_LINES\_T Table Type

This table type stores lines of post-processed source text. It is used to hold PL/SQL source text both before and after it is processed. It is especially useful in cases in which the amount of text exceeds 32K.

### Syntax

```
TYPE source_lines_t IS  
    TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

## Summary of DBMS\_PREPROCESSOR Subprograms

**Table 105-1 DBMS\_PREPROCESSOR Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_POST_PROCESSED_SOURCE Functions</a> on page 105-8	Returns the post-processed source text
<a href="#">PRINT_POST_PROCESSED_SOURCE Procedures</a> on page 105-10	Prints post-processed source text

## GET\_POST\_PROCESSED\_SOURCE Functions

This overloaded function returns the post-processed source text. The different functionality of each form of syntax is presented along with the definition.

### Syntax

Returns post-processed source text of a stored PL/SQL unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (
    object_type    IN VARCHAR2,
    schema_name    IN VARCHAR2,
    object_name    IN VARCHAR2)
RETURN source_lines_t;
```

Returns post-processed source text of a compilation unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (
    source         IN VARCHAR2)
RETURN source_lines_t;
```

Returns post-processed source text of an INDEX-BY table containing the source text of the compilation unit:

```
DBMS_PREPROCESSOR.GET_POST_PROCESSED_SOURCE (
    source         IN source_lines_t)
RETURN source_lines_t;
```

### Parameters

**Table 105–2** GET\_POST\_PROCESSED\_SOURCE Function Parameters

Parameter	Description
object_type	Must be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER. Case sensitive.
schema_name	The schema name. Case insensitive unless a quoted identifier is used. If NULL, use current schema.
object_name	The name of the object. The object_type is always case insensitive. Case insensitive unless a quoted identifier is used.
source	The source text of the compilation unit
source_lines_t	INDEX-BY table containing the source text of the compilation unit. The source text is a concatenation of all the non-NULL INDEX-BY table elements in ascending index order.

### Return Values

The function returns an INDEX-BY table containing the lines of the post-processed source text starting from index 1.

### Usage Notes

- Newline characters are not removed.
- Each line in the post-processed source text is mapped to a row in the INDEX-BY table.
- In the post-processed source, unselected text will have blank lines.

## Exceptions

**Table 105-3** *GET\_POST\_PROCESSED\_SOURCE* Function Exceptions

Exception	Description
ORA-24234	Insufficient privileges or object does not exist
ORA-24235	Bad value for object type. Should be one of PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, TYPE, TYPE, BODY or TRIGGER.
ORA-24236	The source text is empty
ORA-00931	Missing identifier. The object_name should not be NULL.
ORA-06502	Numeric or value error: <ul style="list-style-type: none"><li>Character string buffer too small</li><li>A line is too long (&gt; 32767 bytes)</li></ul>

## PRINT\_POST\_PROCESSED\_SOURCE Procedures

This overloaded procedure calls `DBMS_OUTPUT.PUT_LINE` to let you view post-processed source text. The different functionality of each form of syntax is presented along with the definition.

### Syntax

Prints post-processed source text of a stored PL/SQL unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
  object_type  IN VARCHAR2,
  schema_name  IN VARCHAR2,
  object_name  IN VARCHAR2);
```

Prints post-processed source text of a compilation unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
  source       IN VARCHAR2);
```

Prints post-processed source text of an `INDEX-BY` table containing the source text of the compilation unit:

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE (
  source       IN source_lines_t);
```

### Parameters

**Table 105–4 PRINT\_POST\_PROCESSED\_SOURCE Procedure Parameters**

Parameter	Description
<code>object_type</code>	Must be one of <code>PACKAGE</code> , <code>PACKAGE BODY</code> , <code>PROCEDURE</code> , <code>FUNCTION</code> , <code>TYPE</code> , <code>TYPE, BODY</code> or <code>TRIGGER</code> . Case sensitive.
<code>schema_name</code>	The schema name. Case insensitive unless a quoted identifier is used. If <code>NULL</code> , use current schema.
<code>object_name</code>	The name of the object. The <code>object_type</code> is always case insensitive. Case insensitive unless a quoted identifier is used.
<code>source</code>	The source text of the compilation unit
<code>source_lines_t</code>	<code>INDEX-BY</code> table containing the source text of the compilation unit. The source text is a concatenation of all the non- <code>NULL</code> <code>INDEX-BY</code> table elements in ascending index order.

### Exceptions

**Table 105–5 PRINT\_POST\_PROCESSED\_SOURCE Procedure Exceptions**

Exception	Description
ORA-24234	Insufficient privileges or object does not exist
ORA-24235	Bad value for object type. Should be one of <code>PACKAGE</code> , <code>PACKAGE BODY</code> , <code>PROCEDURE</code> , <code>FUNCTION</code> , <code>TYPE</code> , <code>TYPE, BODY</code> or <code>TRIGGER</code> .
ORA-24236	The source text is empty
ORA-00931	Missing identifier. The <code>object_name</code> should not be <code>NULL</code> .

**Table 105-5 (Cont.) PRINT\_POST\_PROCESSED\_SOURCE Procedure Exceptions**

<b>Exception</b>	<b>Description</b>
ORA-06502	Numeric or value error: <ul style="list-style-type: none"><li>▪ Character string buffer too small</li><li>▪ A line is too long (&gt; 32767 bytes)</li></ul>

**Usage Notes**

The index-by table may contain holes. NULL elements are ignored when doing the concatenation.





The DBMS\_PROFILER package provides an interface to profile existing PL/SQL applications and identify performance bottlenecks. You can then collect and persistently store the PL/SQL profiler data.

This chapter contains the following topics:

- [Using DBMS\\_PROFILER](#)
  - Overview
  - Security Model
  - Operational Notes
  - Exceptions
- [Summary of DBMS\\_PROFILER Subprograms](#)

---

## Using DBMS\_PROFILER

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)
- [Exceptions](#)

## Overview

This package enables the collection of profiler (performance) data for performance improvement or for determining code coverage for PL/SQL applications. Application developers can use code coverage data to focus their incremental testing efforts.

With this interface, you can generate profiling information for all named library units that are executed in a session. The profiler gathers information at the PL/SQL virtual machine level. This information includes the total number of times each line has been executed, the total amount of time that has been spent executing that line, and the minimum and maximum times that have been spent on a particular execution of that line.

---

**Note:** It is possible to infer the code coverage figures for PL/SQL units for which data has been collected.

---

The profiling information is stored in database tables. This enables querying on the data: you can build customizable reports (summary reports, hottest lines, code coverage data, and so on. And you can analyze the data.

The `PROFTAB.SQL` script creates tables with the columns, datatypes, and definitions as shown in [Table 106–1](#), [Table 106–2](#), and [Table 106–3](#).

**Table 106–1 Columns in Table `PLSQL_PROFILER_RUNS`**

Column	Datatype	Definition
<code>runid</code>	NUMBER PRIMARY KEY	Unique run identifier from <code>plsql_profiler_runnumber</code>
<code>related_run</code>	NUMBER	Runid of related run (for client/server correlation)
<code>run_owner</code>	VARCHAR2 (32)	User who started run
<code>run_date</code>	DATE	Start time of run
<code>run_comment</code>	VARCHAR2 (2047)	User provided comment for this run
<code>run_total_time</code>	NUMBER	Elapsed time for this run in nanoseconds
<code>run_system_info</code>	VARCHAR2 (2047)	Currently unused
<code>run_comment1</code>	VARCHAR2 (2047)	Additional comment
<code>spare1</code>	VARCHAR2 (256)	Unused

**Table 106–2 Columns in Table `PLSQL_PROFILER_UNITS`**

Column	Datatype	Definition
<code>runid</code>	NUMBER	Primary key, references <code>plsql_profiler_runs</code> ,
<code>unit_number</code>	NUMBER	Primary key, internally generated library unit #
<code>unit_type</code>	VARCHAR2 (32)	Library unit type
<code>unit_owner</code>	VARCHAR2 (32)	Library unit owner name
<code>unit_name</code>	VARCHAR2 (32)	Library unit name timestamp on library unit
<code>unit_timestamp</code>	DATE	In the future will be used to detect changes to unit between runs

**Table 106–2 (Cont.) Columns in Table PLSQL\_PROFILER\_UNITS**

Column	Datatype	Definition
total_time	NUMBER	Total time spent in this unit in nanoseconds. The profiler does not set this field, but it is provided for the convenience of analysis tools.
spare1	NUMBER	Unused
spare2	NUMBER	Unused

**Table 106–3 Columns in Table PLSQL\_PROFILER\_DATA**

Column	Datatype	Definition
runid	NUMBER	Primary key, unique (generated) run identifier
unit_number	NUMBER	Primary key, internally generated library unit number
line#	NUMBER	Primary key, not null, line number in unit
total_occur	NUMBER	Number of times line was executed
total_time	NUMBER	Total time spent executing line in nanoseconds
min_time	NUMBER	Minimum execution time for this line in nanoseconds
max_time	NUMBER	Maximum execution time for this line in nanoseconds
spare1	NUMBER	Unused
spare2	NUMBER	Unused
spare3	NUMBER	Unused
spare4	NUMBER	Unused

With Oracle database version 8.x, a sample textual report writer(`profrep.sql`) is provided with the PL/SQL demo scripts.

## Security Model

The profiler only gathers data for units for which a user has `CREATE` privilege; you cannot use the package to profile units for which `EXECUTE ONLY` access has been granted. In general, if a user can debug a unit, the same user can profile it. However, a unit can be profiled whether or not it has been compiled `DEBUG`. Oracle advises that modules that are being profiled should be compiled `DEBUG`, since this provides additional information about the unit in the database.

---

---

**Note:** `DBMS_PROFILER` treats any program unit that is compiled in `NATIVE` mode as if you do not have `CREATE` privilege, that is, you will not get any output.

---

---

## Operational Notes

- [Typical Run](#)
- [Two Methods of Exception Generation](#)

### Typical Run

Improving application performance is an iterative process. Each iteration involves the following steps:

1. Running the application with one or more benchmark tests with profiler data collection enabled.
2. Analyzing the profiler data and identifying performance problems.
3. Fixing the problems.

The PL/SQL profiler supports this process using the concept of a "run". A run involves running the application through benchmark tests with profiler data collection enabled. You can control the beginning and the ending of a run by calling the `START_PROFILER` and `STOP_PROFILER` functions.

A typical run involves:

- Starting profiler data collection in the run.
- Executing PL/SQL code for which profiler and code coverage data is required.
- Stopping profiler data collection, which writes the collected data for the run into database tables

---

---

**Note:** The collected profiler data is not automatically stored when the user disconnects. You must issue an explicit call to the `FLUSH_DATA` or the `STOP_PROFILER` function to store the data at the end of the session. Stopping data collection stores the collected data.

---

---

As the application executes, profiler data is collected in memory data structures that last for the duration of the run. You can call the `FLUSH_DATA` function at intermediate points during the run to get incremental data and to free memory for allocated profiler data structures.

Flushing the collected data involves storing collected data in database tables. The tables should already exist in the profiler user's schema. The `PROFTAB.SQL` script creates the tables and other data structures required for persistently storing the profiler data.

Note that running `PROFTAB.SQL` drops the current tables. The `PROFTAB.SQL` script is in the `RDBMS/ADMIN` directory. Some PL/SQL operations, such as the first execution of a PL/SQL unit, may involve I/O to catalog tables to load the byte code for the PL/SQL unit being executed. Also, it may take some time executing package initialization code the first time a package procedure or function is called.

To avoid timing this overhead, "warm up" the database before collecting profile data. To do this, run the application once without gathering profiler data.

You can allow profiling across all users of a system, for example, to profile all users of a package, independent of who is using it. In such cases, the `SYSADMIN` should use a modified `PROFLOAD.SQL` script which:

- Creates the profiler tables and sequence
- Grants SELECT/INSERT/UPDATE on those tables and sequence to all users
- Defines public synonyms for the tables and sequence

---

---

**Note:** Do not alter the actual fields of the tables.

---

---

**See Also:** ["FLUSH\\_DATA Function and Procedure"](#) on page 106-10.

## Two Methods of Exception Generation

Each routine in this package has two versions that allow you to determine how errors are reported.

- A function that returns success/failure as a status value and will never raise an exception
- A procedure that returns normally if it succeeds and raises an exception if it fails

In each case, the parameters of the function and procedure are identical. Only the method by which errors are reported differs. If there is an error, there is a correspondence between the error codes that the functions return, and the exceptions that the procedures raise.

To avoid redundancy, the following section only provides details about the functional form.

## Exceptions

**Table 106–4** *DBMS\_PROFILER Exceptions*

Exception	Description
<code>version_mismatch</code>	Corresponds to <code>error_version</code> .
<code>profiler_error</code>	Corresponds to either "error_param" or "error_io".

A 0 return value from any function denotes successful completion; a nonzero return value denotes an error condition. The possible errors are as follows:

- 'A subprogram was called with an incorrect parameter.'  
`error_param` constant `binary_integer` := 1;
- 'Data flush operation failed. Check whether the profiler tables have been created, are accessible, and that there is adequate space.'  
`error_io` constant `binary_integer` := 2;
- There is a mismatch between package and database implementation. Oracle returns this error if an incorrect version of the `DBMS_PROFILER` package is installed, and if the version of the profiler package cannot work with this database version. The only recovery is to install the correct version of the package.  
`error_version` constant `binary_integer` := -1;



---

## Summary of DBMS\_PROFILER Subprograms

**Table 106–5** *DBMS\_PROFILER Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">FLUSH_DATA Function and Procedure</a> on page 106-10	Flushes profiler data collected in the user's session
<a href="#">GET_VERSION Procedure</a> on page 106-11	Gets the version of this API
<a href="#">INTERNAL_VERSION_CHECK Function</a> on page 106-12	Verifies that this version of the DBMS_PROFILER package can work with the implementation in the database
<a href="#">PAUSE_PROFILER Function and Procedure</a> on page 106-13	Pauses profiler data collection
<a href="#">RESUME_PROFILER Function and Procedure</a> on page 106-14	Resumes profiler data collection
<a href="#">START_PROFILER Functions and Procedures</a> on page 106-15	Starts profiler data collection in the user's session
<a href="#">STOP_PROFILER Function and Procedure</a> on page 106-16	Stops profiler data collection in the user's session

## FLUSH\_DATA Function and Procedure

This function flushes profiler data collected in the user's session. The data is flushed to database tables, which are expected to preexist.

---

---

**Note:** Use the `PROFTAB.SQL` script to create the tables and other data structures required for persistently storing the profiler data.

---

---

### Syntax

```
DBMS_PROFILER.FLUSH_DATA  
    RETURN BINARY_INTEGER;
```

```
DBMS_PROFILER.FLUSH_DATA;
```

## GET\_VERSION Procedure

This procedure gets the version of this API.

### Syntax

```
DBMS_PROFILER.GET_VERSION (  
    major OUT BINARY_INTEGER,  
    minor OUT BINARY_INTEGER);
```

### Parameters

**Table 106–6** *GET\_VERSION Procedure Parameters*

Parameter	Description
major	Major version of DBMS_PROFILER.
minor	Minor version of DBMS_PROFILER.

## INTERNAL\_VERSION\_CHECK Function

This function verifies that this version of the DBMS\_PROFILER package can work with the implementation in the database.

### Syntax

```
DBMS_PROFILER.INTERNAL_VERSION_CHECK  
RETURN BINARY_INTEGER;
```

## PAUSE\_PROFILER Function and Procedure

This function pauses profiler data collection.

### Syntax

```
DBMS_PROFILER.PAUSE_PROFILER  
    RETURN BINARY_INTEGER;  
  
DBMS_PROFILER.PAUSE_PROFILER;
```

## RESUME\_PROFILER Function and Procedure

This function resumes profiler data collection.

### Syntax

```
DBMS_PROFILER.RESUME_PROFILER  
    RETURN BINARY_INTEGER;  
  
DBMS_PROFILER.RESUME_PROFILER;
```

## START\_PROFILER Functions and Procedures

This function starts profiler data collection in the user's session.

There are two overloaded forms of the `START_PROFILER` function; one returns the run number of the started run, as well as the result of the call. The other does not return the run number. The first form is intended for use with GUI-based tools controlling the profiler.

### Syntax

```
DBMS_PROFILER.START_PROFILER(
    run_comment    IN VARCHAR2 := sysdate,
    run_comment1   IN VARCHAR2 := '',
    run_number     OUT BINARY_INTEGER)
RETURN BINARY_INTEGER;
```

```
DBMS_PROFILER.START_PROFILER(
    run_comment IN VARCHAR2 := sysdate,
    run_comment1 IN VARCHAR2 := '')
RETURN BINARY_INTEGER;
```

```
DBMS_PROFILER.START_PROFILER(
    run_comment    IN VARCHAR2 := sysdate,
    run_comment1   IN VARCHAR2 := '',
    run_number     OUT BINARY_INTEGER);
```

```
DBMS_PROFILER.START_PROFILER(
    run_comment IN VARCHAR2 := sysdate,
    run_comment1 IN VARCHAR2 := '');
```

### Parameters

**Table 106–7** *START\_PROFILER Function Parameters*

Parameter	Description
<code>run_comment</code>	Each profiler run can be associated with a comment. For example, the comment could provide the name and version of the benchmark test that was used to collect data.
<code>run_number</code>	Stores the number of the run so you can store and later recall the run's data.
<code>run_comment1</code>	Allows you to make interesting comments about the run.

## STOP\_PROFILER Function and Procedure

This function stops profiler data collection in the user's session.

This function has the side effect of flushing data collected so far in the session, and it signals the end of a run.

### Syntax

```
DBMS_PROFILER.STOP_PROFILER  
    RETURN BINARY_INTEGER;  
  
DBMS_PROFILER.STOP_PROFILER;
```



---

---

## DBMS\_PROPAGATION\_ADM

The DBMS\_PROPAGATION\_ADM package, one of a set of Oracle Streams packages, provides administrative interfaces for configuring a propagation from a source queue to a destination queue.

This chapter contains the following topics:

- [Using DBMS\\_PROPAGATION\\_ADM](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_PROPAGATION\\_ADM Subprograms](#)

---

## Using DBMS\_PROPAGATION\_ADM

This section contains topics which relate to using the DBMS\_CAPTURE\_ADM package.

- [Overview](#)
- [Security Model](#)

## Overview

This package provides interfaces to start, stop, and configure a propagation.

**See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and propagations

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

When the DBMS\_PROPAGATION\_ADM package is used to manage an Oracle Streams configuration, it requires that the user is granted the privileges of an Oracle Streams administrator.

**See Also:** *Oracle Streams Concepts and Administration* for information about configuring an Oracle Streams administrator

---

## Summary of DBMS\_PROPAGATION\_ADM Subprograms

**Table 107-1 DBMS\_PROPAGATION\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ALTER_PROPAGATION Procedure</a> on page 107-6	Adds, alters, or removes a rule set for a propagation
<a href="#">CREATE_PROPAGATION Procedure</a> on page 107-8	Creates a propagation and specifies the source queue, destination queue, and rule set for the propagation
<a href="#">DROP_PROPAGATION Procedure</a> on page 107-11	Drops a propagation
<a href="#">START_PROPAGATION Procedure</a> on page 107-13	Starts a propagation
<a href="#">STOP_PROPAGATION Procedure</a> on page 107-14	Stops a propagation

---

**Note:** All subprograms commit unless specified otherwise.

---

## ALTER\_PROPAGATION Procedure

This procedure adds, alters, or removes a rule set for a propagation.

**See Also:** *Oracle Streams Concepts and Administration* and [Chapter 127, "DBMS\\_RULE\\_ADM"](#) for more information about rules and rule sets

### Syntax

```
DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
  propagation_name      IN  VARCHAR2,
  rule_set_name         IN  VARCHAR2  DEFAULT NULL,
  remove_rule_set      IN  BOOLEAN   DEFAULT FALSE,
  negative_rule_set_name IN  VARCHAR2  DEFAULT NULL,
  remove_negative_rule_set IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 107–2 ALTER\_PROPAGATION Procedure Parameters**

Parameter	Description
propagation_name	The name of the propagation you are altering. You must specify an existing propagation name. Do not specify an owner.
rule_set_name	<p>The name of the positive rule set for the propagation. The positive rule set contains the rules that instruct the propagation to propagate messages.</p> <p>If you want to use a positive rule set for the propagation, then you must specify an existing rule set in the form <code>[ schema_name. ] rule_set_name</code>. For example, to specify a positive rule set in the <code>hr</code> schema named <code>prop_rules</code>, enter <code>hr.prop_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify <code>NULL</code> and the <code>remove_rule_set</code> parameter is set to <code>FALSE</code>, then the procedure retains any existing positive rule set. If you specify <code>NULL</code> and the <code>remove_rule_set</code> parameter is set to <code>TRUE</code>, then the procedure removes any existing positive rule set.</p>
remove_rule_set	<p>If <code>TRUE</code>, then the procedure removes the positive rule set for the specified propagation. If you remove a positive rule set for a propagation, and the propagation does not have a negative rule set, then the propagation propagates all messages.</p> <p>If you remove a positive rule set for a propagation, and a negative rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the negative rule set.</p> <p>If <code>FALSE</code>, then the procedure retains the positive rule set for the specified propagation.</p> <p>If the <code>rule_set_name</code> parameter is non-<code>NULL</code>, then this parameter should be set to <code>FALSE</code>.</p>

**Table 107-2 (Cont.) ALTER\_PROPAGATION Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>negative_rule_set_name</code>	<p>The name of the negative rule set for the propagation. The negative rule set contains the rules that instruct the propagation to discard messages.</p> <p>If you want to use a negative rule set for the propagation, then you must specify an existing rule set in the form <code>[schema_name.] rule_set_name</code>. For example, to specify a negative rule set in the <code>hr</code> schema named <code>neg_rules</code>, enter <code>hr.neg_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify <code>NULL</code> and the <code>remove_negative_rule_set</code> parameter is set to <code>FALSE</code>, then the procedure retains any existing negative rule set. If you specify <code>NULL</code> and the <code>remove_negative_rule_set</code> parameter is set to <code>TRUE</code>, then the procedure removes any existing negative rule set.</p> <p>If you specify both a positive and a negative rule set for a propagation, then the negative rule set is always evaluated first.</p>
<code>remove_negative_rule_set</code>	<p>If <code>TRUE</code>, then the procedure removes the negative rule set for the specified propagation. If you remove a negative rule set for a propagation, and the propagation does not have a positive rule set, then the propagation propagates all messages.</p> <p>If you remove a negative rule set for a propagation, and a positive rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the positive rule set.</p> <p>If <code>FALSE</code>, then the procedure retains the negative rule set for the specified propagation.</p> <p>If the <code>negative_rule_set_name</code> parameter is non-<code>NULL</code>, then this parameter should be set to <code>FALSE</code>.</p>

## CREATE\_PROPAGATION Procedure

This procedure creates a propagation and specifies the source queue, destination queue, and any rule set for the propagation. A propagation propagates messages in a local source queue to a destination queue. The destination queue might or might not be in the same database as the source queue.

### Syntax

```
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION (
  propagation_name      IN  VARCHAR2,
  source_queue          IN  VARCHAR2,
  destination_queue     IN  VARCHAR2,
  destination_dblink    IN  VARCHAR2  DEFAULT NULL,
  rule_set_name         IN  VARCHAR2  DEFAULT NULL,
  negative_rule_set_name IN  VARCHAR2  DEFAULT NULL,
  queue_to_queue        IN  BOOLEAN   DEFAULT NULL,
  original_propagation_name IN VARCHAR2  DEFAULT NULL,
  auto_merge_threshold  IN  NUMBER    DEFAULT NULL);
```

### Parameters

**Table 107-3 CREATE\_PROPAGATION Procedure Parameters**

Parameter	Description
<code>propagation_name</code>	The name of the propagation you are creating. A NULL setting is not allowed. Do not specify an owner.  <b>Note:</b> The <code>propagation_name</code> setting cannot be altered after the propagation is created.
<code>source_queue</code>	The name of the source queue, specified as [ <i>schema_name</i> .] <i>queue_name</i> . The current database must contain the source queue.  For example, to specify a source queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.
<code>destination_queue</code>	The name of the destination queue, specified as [ <i>schema_name</i> .] <i>queue_name</i> .  For example, to specify a destination queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.
<code>destination_dblink</code>	The name of the database link that will be used by the propagation. The database link is from the database that contains the source queue to the database that contains the destination queue.  If NULL, then the source queue and destination queue must be in the same database.  <b>Note:</b> Connection qualifiers are not allowed.



**Table 107-3 (Cont.) CREATE\_PROPAGATION Procedure Parameters**

Parameter	Description
rule_set_name	<p>The name of the positive rule set for the propagation. The positive rule set contains the rules that instruct the propagation to propagate messages.</p> <p>If you want to use a positive rule set for the propagation, then you must specify an existing rule set in the form <code>[ schema_name. ] rule_set_name</code>. For example, to specify a positive rule set in the <code>hr</code> schema named <code>prop_rules</code>, enter <code>hr.prop_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify <code>NULL</code>, and no negative rule set exists for the propagation, then the propagation propagates all messages in its queue.</p> <p>If you specify <code>NULL</code>, and a negative rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the negative rule set.</p>
negative_rule_set_name	<p>The name of the negative rule set for the propagation. The negative rule set contains the rules that instruct the propagation to discard messages.</p> <p>If you want to use a negative rule set for the propagation, then you must specify an existing rule set in the form <code>[ schema_name. ] rule_set_name</code>. For example, to specify a negative rule set in the <code>hr</code> schema named <code>neg_rules</code>, enter <code>hr.neg_rules</code>. If the schema is not specified, then the current user is the default.</p> <p>An error is returned if the specified rule set does not exist. You can create a rule set and add rules to it using the <code>DBMS_STREAMS_ADM</code> package or the <code>DBMS_RULE_ADM</code> package.</p> <p>If you specify <code>NULL</code>, and no positive rule set exists for the propagation, then the propagation propagates all messages in its queue.</p> <p>If you specify <code>NULL</code>, and a positive rule set exists for the propagation, then the propagation propagates all messages in its queue that are not discarded by the positive rule set.</p> <p>If you specify both a positive and a negative rule set for a propagation, then the negative rule set is always evaluated first.</p>
queue_to_queue	<p>If <code>TRUE</code> or <code>NULL</code>, then the propagation is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.</p> <p>If <code>FALSE</code>, then the propagation is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.</p> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about queue-to-queue propagations</p>

**Table 107–3 (Cont.) CREATE\_PROPAGATION Procedure Parameters**

Parameter	Description
<code>original_propagation_name</code>	<p>Specify the original propagation name if the propagation being created is part of a split and merge operation initiated by the <code>SPLIT_STREAMS</code> procedure in the <code>DBMS_STREAMS_ADM</code> package. The split operation clones the original propagation under a new name. The name of the original propagation is important when the cloned propagation is copied back to the original stream using the <code>MERGE_STREAMS</code> procedure in the <code>DBMS_STREAMS_ADM</code> package.</p> <p>Specify NULL if the propagation being created is not part of a split and merge operation.</p> <p><b>See Also:</b> <a href="#">SPLIT_STREAMS Procedure</a> on page 144-175 and <a href="#">MERGE_STREAMS Procedure</a> on page 144-130</p>
<code>auto_merge_theshold</code>	<p>Specify a positive number if both of the following conditions are met:</p> <ul style="list-style-type: none"> <li>■ The propagation being created is part of a split and merge operation initiated by the <code>SPLIT_STREAMS</code> procedure in the <code>DBMS_STREAMS_ADM</code> package.</li> <li>■ The stream will be merged back to the original stream automatically.</li> </ul> <p>Specify NULL if either of the following conditions are met:</p> <ul style="list-style-type: none"> <li>■ The propagation being created is not part of a split and merge operation.</li> <li>■ The propagation being created is part of a split and merge operation, but the stream being split off will not be merged back to the original stream automatically.</li> </ul> <p><b>See Also:</b> <a href="#">SPLIT_STREAMS Procedure</a> on page 144-175 and <a href="#">MERGE_STREAMS Procedure</a> on page 144-130</p>

## Usage Notes

This procedure starts propagation and might create a propagation job. If this procedure creates a propagation job, then it establishes a default schedule for the propagation job. Each propagation job is an Oracle Scheduler job. You can adjust the schedule of a propagation job using Oracle Scheduler.

The user who owns the source queue is the user who propagates messages. This user must have the necessary privileges to propagate messages.

**See Also:**

- [Chapter 127, "DBMS\\_RULE\\_ADM"](#)
- *Oracle Streams Concepts and Administration* for more information about propagations, the privileges required to propagate messages, propagation jobs, and propagation schedules

## DROP\_PROPAGATION Procedure

This procedure drops a propagation and deletes all messages for the destination queue in the source queue. This procedure also removes the schedule for propagation from the source queue to the destination queue.

### Syntax

```
DBMS_PROPAGATION_ADM.DROP_PROPAGATION(
  propagation_name      IN VARCHAR2,
  drop_unused_rule_sets IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 107–4** *DROP\_PROPAGATION Procedure Parameters*

Parameter	Description
propagation_name	The name of the propagation you are dropping. You must specify an existing propagation name. Do not specify an owner.
drop_unused_rule_sets	If TRUE, then the procedure drops any rule sets, positive and negative, used by the specified propagation if these rule sets are not used by any other Oracle Streams client, which includes capture processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set.  If FALSE, then the procedure does not drop the rule sets used by the specified propagation, and the rule sets retain their rules.

### Usage Notes

When you use this procedure to drop a propagation, information about rules created for the propagation using the DBMS\_STREAMS\_ADM package is removed from the data dictionary views for Oracle Streams rules. Information about such a rule is removed even if the rule is not in either rule set for the propagation.

**See Also:** *Oracle Streams Concepts and Administration* for more information about Oracle Streams data dictionary views

The following are the data dictionary views for Oracle Streams rules:

- ALL\_STREAMS\_GLOBAL\_RULES
- DBA\_STREAMS\_GLOBAL\_RULES
- ALL\_STREAMS\_MESSAGE\_RULES
- DBA\_STREAMS\_MESSAGE\_RULES
- ALL\_STREAMS\_SCHEMA\_RULES
- DBA\_STREAMS\_SCHEMA\_RULES
- ALL\_STREAMS\_TABLE\_RULES
- DBA\_STREAMS\_TABLE\_RULES

---

---

**Note:** When you drop a propagation, the propagation job used by the propagation is dropped automatically, if no other propagations are using the propagation job.

---

---

## START\_PROPAGATION Procedure

This procedure starts a propagation.

### Syntax

```
DBMS_PROPAGATION_ADM.START_PROPAGATION(  
    propagation_name IN VARCHAR2);
```

### Parameter

**Table 107-5** *START\_PROPAGATION Procedure Parameter*

Parameter	Description
propagation_name	The name of the propagation you are starting. You must specify an existing propagation name. Do not specify an owner.

### Usage Notes

The propagation status is persistently recorded. Hence, if the status is `ENABLED`, then the propagation is started upon database instance startup.

## STOP\_PROPAGATION Procedure

This procedure stops a propagation.

### Syntax

```
DBMS_PROPAGATION_ADM.STOP_PROPAGATION(  
    propagation_name IN VARCHAR2,  
    force             IN BOOLEAN DEFAULT FALSE);
```

### Parameter

**Table 107–6 STOP\_PROPAGATION Procedure Parameter**

Parameter	Description
propagation_name	The name of the propagation you are stopping. You must specify an existing propagation name. Do not specify an owner.
force	If <b>TRUE</b> , then the procedure stops the propagation and clears the statistics for the propagation.  If <b>FALSE</b> , then the procedure stops the propagation without clearing the statistics for the propagation.

### Usage Notes

The propagation status is persistently recorded. Hence, if the status is **DISABLED** or **ABORTED**, then the propagation is not started upon database instance startup.

The DBMS\_RANDOM package provides a built-in random number generator. DBMS\_RANDOM is not intended for cryptography.

This chapter contains the following topics:

- [Using DBMS\\_RANDOM](#)
  - Deprecated Subprograms
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_RANDOM Subprograms](#)

---

## Using DBMS\_RANDOM

- [Deprecated Subprograms](#)
- [Security Model](#)
- [Operational Notes](#)



## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

---

The following subprograms are deprecated with Oracle Database 11g:

- [INITIALIZE Procedure](#)
- [RANDOM Procedure](#)
- [TERMINATE Procedure](#)

## Security Model

This package should be installed as `SYS`. By default, the package is initialized with the current user name, current time down to the second, and the current session. Oracle recommends that users who need to execute this package should be given `EXECUTE` privilege explicitly and should not rely on `PUBLIC EXECUTE` privilege.

## Operational Notes

- `DBMS_RANDOM.RANDOM` produces integers in  $[-2^{31}, 2^{31})$ .
- `DBMS_RANDOM.VALUE` produces numbers in  $[0,1)$  with 38 digits of precision.

`DBMS_RANDOM` can be explicitly initialized, but does not need to be initialized before calling the random number generator. It will automatically initialize with the date, user ID, and process ID if no explicit initialization is performed.

If this package is seeded twice with the same seed, then accessed in the same way, it will produce the same results in both cases.

In some cases, such as when testing, you may want the sequence of random numbers to be the same on every run. In that case, you seed the generator with a constant value by calling one of the overloads of `DBMS_RANDOM.SEED`. To produce different output for every run, simply to omit the call to "Seed" and the system will choose a suitable seed for you.

---

## Summary of DBMS\_RANDOM Subprograms

**Table 108–1 DBMS\_RANDOM Package Subprograms**

Subprogram	Description
<a href="#">INITIALIZE Procedure</a> on page 108-7	Initializes the package with a seed value
<a href="#">NORMAL Function</a> on page 108-8	Returns random numbers in a normal distribution
<a href="#">RANDOM Procedure</a> on page 108-9	Generates a random number
<a href="#">SEED Procedures</a> on page 108-10	Resets the seed
<a href="#">STRING Function</a> on page 108-11	Gets a random string
<a href="#">TERMINATE Procedure</a> on page 108-12	Terminates package
<a href="#">VALUE Functions</a> on page 108-13	Gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision), while the overloaded function gets a random Oracle number <i>x</i> , where <i>x</i> is greater than or equal to <i>low</i> and less than <i>high</i>

## INITIALIZE Procedure

This procedure initializes the generator.

---

---

**Note:** This procedure is deprecated with Release 11gR1 and, although currently supported, it should not be used.

---

---

### Syntax

```
DBMS_RANDOM.INITIALIZE (  
    val IN BINARY_INTEGER);
```

### Pragmas

```
PRAGMA restrict_references (initialize, WNDS);
```

### Parameters

**Table 108–2 INITIALIZE Procedure Parameters**

Parameter	Description
val	Seed number used to generate a random number

### Usage Notes

This procedure is obsolete as it simply calls the [SEED Procedures](#) on page 108-10.

## NORMAL Function

This function returns random numbers in a standard normal distribution.

### Syntax

```
DBMS_RANDOM.NORMAL  
RETURN NUMBER;
```

### Pragmas

```
PRAGMA restrict_references (normal, WNDS);
```

### Return Values

**Table 108–3** *NORMAL Function Parameters*

Parameter	Description
number	Returns a random number

## RANDOM Procedure

This procedure generates a random number.

---

---

**Note:** This procedure is deprecated with Release 11gR1 and, although currently supported, it should not be used.

---

---

### Syntax

```
DBMS_RANDOM.RANDOM  
RETURN binary_integer;
```

### Pragmas

```
PRAGMA restrict_references (random, WNDS);
```

### Return Values

**Table 108–4** *RANDOM Procedure Parameters*

Parameter	Description
binary_integer	Returns a random integer greater or equal to $-\text{power}(2,31)$ and less than $\text{power}(2,31)$

## SEED Procedures

This procedure resets the seed.

### Syntax

```
DBMS_RANDOM.SEED (  
    val IN BINARY_INTEGER);
```

```
DBMS_RANDOM.SEED (  
    val IN VARCHAR2);
```

### Pragmas

```
PRAGMA restrict_references (seed, WNDS);
```

### Parameters

**Table 108–5 SEED Procedure Parameters**

Parameter	Description
val	Seed number or string used to generate a random number

### Usage Notes

The seed can be a string up to length 2000.



## STRING Function

This function gets a random string.

### Syntax

```
DBMS_RANDOM.STRING
  opt IN CHAR,
  len IN NUMBER)
RETURN VARCHAR2;
```

### Pragmas

```
PRAGMA restrict_references (string, WNDS);
```

### Parameters

**Table 108–6** *STRING Function Parameters*

Parameter	Description
opt	Specifies what the returning string looks like: <ul style="list-style-type: none"> <li>▪ 'u', 'U' - returning string in uppercase alpha characters</li> <li>▪ 'l', 'L' - returning string in lowercase alpha characters</li> <li>▪ 'a', 'A' - returning string in mixed case alpha characters</li> <li>▪ 'x', 'X' - returning string in uppercase alpha-numeric characters</li> <li>▪ 'p', 'P' - returning string in any printable characters.</li> </ul> Otherwise the returning string is in uppercase alpha characters.
len	Length of the returning string

### Return Values

**Table 108–7** *STRING Function Return Values*

Parameter	Description
VARCHAR2	Returns a VARCHAR2

## TERMINATE Procedure

When you are finished with the package, call the `TERMINATE` procedure.

---

---

**Note:** This procedure is deprecated with Release 11gR1 and, although currently supported, it should not be used.

---

---

### Syntax

```
DBMS_RANDOM.TERMINATE;
```

## VALUE Functions

The basic function gets a random number, greater than or equal to 0 and less than 1, with 38 digits to the right of the decimal (38-digit precision). Alternatively, you can get a random Oracle number *x*, where *x* is greater than or equal to *low* and less than *high*.

### Syntax

```
DBMS_RANDOM.VALUE
  RETURN NUMBER;
```

```
DBMS_RANDOM.VALUE (
  low IN NUMBER,
  high IN NUMBER)
  RETURN NUMBER;
```

### Parameters

**Table 108–8** VALUE Function Parameters

Parameter	Description
low	Lowest number in a range from which to generate a random number. The number generated may be equal to low
high	Highest number below which to generate a random number. The number generated will be less than high

### Return Values

**Table 108–9** VALUE Function Return Values

Parameter	Description
NUMBER	Returns an Oracle Number



---

---

## DBMS\_RECTIFIER\_DIFF

The DBMS\_RECTIFIER\_DIFF package provides an interface used to detect and resolve data inconsistencies between two replicated sites.

- [Documentation of DBMS\\_RECTIFIER\\_DIFF](#)

---

## Documentation of DBMS\_RECTIFIER\_DIFF

For a complete description of this package within the context of Replication, see DBMS\_RECTIFIER\_DIFF in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_REDEFINITION

The DBMS\_REDEFINITION package provides an interface to perform an online redefinition of tables.

**See Also:** *Oracle Database Administrator's Guide* for more information about online redefinition of tables

This chapter contains the following topics:

- [Using DBMS\\_REDEFINITION](#)
  - Overview
  - Constants
  - Operational Notes
- [Summary of DBMS\\_REDEFINITION Subprograms](#)

## Using DBMS\_REDEFINITION

- [Overview](#)
- [Constants](#)
- [Operational Notes](#)



## Overview

To achieve online redefinition, incrementally maintainable local materialized views are used. These logs keep track of the changes to the master tables and are used by the materialized views during refresh synchronization.

## Constants

The `DBMS_REDEFINITION` package uses the constants shown in [Table 110–1, "DBMS\\_REDEFINITION Constants"](#):

**Table 110–1 DBMS\_REDEFINITION Constants**

Constant	Type	Value	Description
<code>CONS_CONSTRAINT</code>	<code>PLS_INTEGER</code>	3	Used to specify that dependent object type is a constraint
<code>CONS_INDEX</code>	<code>PLS_INTEGER</code>	2	Used to specify that dependent object type is a index
<code>CONS_MVLOG</code>	<code>PLS_INTEGER</code>	10	Used to (un)register a materialized view log, as a dependent object of the table, through the <a href="#">REGISTER_DEPENDENT_OBJECT Procedure</a> and the <a href="#">UNREGISTER_DEPENDENT_OBJECT Procedure</a> .
<code>CONS_ORIG_PARAMS</code>	<code>PLS_INTEGER</code>	1	Used to specify that indexes should be cloned with their original storage parameters
<code>CONS_TRIGGER</code>	<code>PLS_INTEGER</code>	4	Used to specify that dependent object type is a trigger
<code>CONS_USE_PK</code>	<code>BINARY_INTEGER</code>	1	Used to indicate that the redefinition should be done using primary keys or pseudo-primary keys (unique keys with all component columns having not-NULL constraints)
<code>CONS_USE_ROWID</code>	<code>BINARY_INTEGER</code>	2	Used to indicate that the redefinition should be done using rowids

## Operational Notes

- `CONS_USE_PK` and `CONS_USE_ROWID` are constants used as input to the "options\_flag" parameter in both the [START\\_REDEF\\_TABLE Procedure](#) and [CAN\\_REDEF\\_TABLE Procedure](#). `CONS_USE_ROWID` is used to indicate that the redefinition should be done using rowids while `CONS_USE_PK` implies that the redefinition should be done using primary keys or pseudo-primary keys (which are unique keys with all component columns having NOT NULL constraints).
- `CONS_INDEX`, `CONS_MVLOG`, `CONS_TRIGGER` and `CONS_CONSTRAINT` are used to specify the type of the dependent object being (un)registered in [REGISTER\\_DEPENDENT\\_OBJECT Procedure](#) and [UNREGISTER\\_DEPENDENT\\_OBJECT Procedure](#) (parameter "dep\_type").  
`CONS_INDEX ==>` dependent object is of type INDEX  
`CONS_TRIGGER ==>` dependent object is of type TRIGGER  
`CONS_CONSTRAINT==>` dependent object type is of type CONSTRAINT  
`CONS_MVLOG ==>` dependent object is of type MATERIALIZED VIEW LOG
- `CONS_ORIG_PARAMS` as used as input to the "copy\_indexes" parameter in [COPY\\_TABLE\\_DEPENDENTS Procedure](#). Using this parameter implies that the indexes on the original table be copied onto the interim table using the same storage parameters as that of the original index.

## Rules and Limits

For information about various rules and limits that apply to implementation of this package, see the *Oracle Database Administrator's Guide*.

---

## Summary of DBMS\_REDEFINITION Subprograms

**Table 110–2 DBMS\_REDEFINITION Package Subprograms**

Subprogram	Description
<a href="#">ABORT_REDEF_TABLE Procedure</a> on page 110-8	Cleans up errors that occur during the redefinition process and removes all temporary objects created by the reorganization process
<a href="#">CAN_REDEF_TABLE Procedure</a> on page 110-9	Determines if a given table can be redefined online
<a href="#">COPY_TABLE_DEPENDENTS Procedure</a> on page 110-10	Copies the dependent objects of the original table onto the interim table
<a href="#">FINISH_REDEF_TABLE Procedure</a> on page 110-12	Completes the redefinition process.
<a href="#">REGISTER_DEPENDENT_OBJECT Procedure</a> on page 110-13	Registers a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table
<a href="#">START_REDEF_TABLE Procedure</a> on page 110-14	Initiates the redefinition process
<a href="#">SYNC_INTERIM_TABLE Procedure</a> on page 110-15	Keeps the interim table synchronized with the original table
<a href="#">UNREGISTER_DEPENDENT_OBJECT Procedure</a> on page 110-16	Unregisters a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table

## ABORT\_REDEF\_TABLE Procedure

This procedure cleans up errors that occur during the redefinition process. This procedure can also be used to terminate the redefinition process any time after the [START\\_REDEF\\_TABLE Procedure](#) has been called and before the [FINISH\\_REDEF\\_TABLE Procedure](#) is called. This process will remove the temporary objects that are created by the redefinition process such as materialized view logs.

### Syntax

```
DBMS_REDEFINITION.ABORT_REDEF_TABLE (  
    uname          IN VARCHAR2,  
    orig_table     IN VARCHAR2,  
    int_table      IN VARCHAR2,  
    part_name      IN  VARCHAR2 := NULL);
```

### Parameters

**Table 110–3** *ABORT\_REDEF\_TABLE Procedure Parameters*

Parameter	Description
uname	Schema name of the tables
orig_table	Name of the table to be redefined
int_table	Name of the interim table
part_name	Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

## CAN\_REDEF\_TABLE Procedure

This procedure determines if a given table can be redefined online. This is the first step of the online redefinition process. If the table is not a candidate for online redefinition, an error message is raised.

### Syntax

```
DBMS_REDEFINITION.CAN_REDEF_TABLE (
  uname          IN  VARCHAR2,
  tname          IN  VARCHAR2,
  options_flag   IN  PLS_INTEGER := 1,
  part_name      IN  VARCHAR2 := NULL);
```

### Parameters

**Table 110–4 CAN\_REDEF\_TABLE Procedure Parameters**

Parameter	Description
uname	Schema name of the table
tname	Name of the table to be re-organized
options_flag	Indicates the type of redefinition method to use. <ul style="list-style-type: none"> <li>■ If <code>dbms_redefinition.cons_use_pk</code>, the redefinition is done using primary keys or pseudo-primary keys (unique keys with all component columns having NOT NULL constraints). The default method of redefinition is using primary keys.</li> <li>■ If <code>dbms_redefinition.cons_use_rowid</code>, the redefinition is done using rowids.</li> </ul>
part_name	Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

### Exceptions

If the table is not a candidate for online redefinition, an error message is raised.

## COPY\_TABLE\_DEPENDENTS Procedure

This procedure clones the dependent objects of the table being redefined onto the interim table and registers the dependent objects. This procedure does not clone the already registered dependent objects.

This subprogram is used to clone the dependent objects like grants, triggers, constraints and privileges from the table being redefined to the interim table (which represents the post-redefinition table).

### Syntax

```
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS (
  uname           IN VARCHAR2,
  orig_table      IN VARCHAR2,
  int_table       IN VARCHAR2,
  copy_indexes    IN PLS_INTEGER := 1,
  copy_triggers   IN BOOLEAN := TRUE,
  copy_constraints IN BOOLEAN := TRUE,
  copy_privileges IN BOOLEAN := TRUE,
  ignore_errors   IN BOOLEAN := FALSE,
  num_errors      OUT PLS_INTEGER,
  copy_statistics IN BOOLEAN := FALSE,
  copy_mvlog      IN BOOLEAN := FALSE);
```

### Parameters

**Table 110–5 COPY\_TABLE\_DEPENDENTS Procedure Parameters**

Parameter	Description
uname	Schema name of the tables
orig_table	Name of the table being redefined
int_table	Name of the interim table
copy_indexes	Flag indicating whether to copy the indexes <ul style="list-style-type: none"> <li>■ 0 - do not copy any index</li> <li>■ dbms_redefinition.cons_orig_params - copy the indexes using the physical parameters of the source indexes</li> </ul>
copy_triggers	TRUE = clone triggers, FALSE = do nothing
copy_constraints	TRUE = clone constraints, FALSE = do nothing. If compatibility setting is 10.2 or higher, then clone CHECK and NOT NULL constraints
copy_privileges	TRUE = clone privileges, FALSE = do nothing
ignore_errors	TRUE = if an error occurs while cloning a particular dependent object, then skip that object and continue cloning other dependent objects. FALSE = that the cloning process should stop upon encountering an error.
num_errors	Number of errors that occurred while cloning dependent objects
copy_statistics	TRUE = copy statistics, FALSE = do nothing
copy_mvlog	TRUE = copy materialized view log, FALSE = do nothing



## Usage Notes

- The user must check the column `num_errors` before proceeding to ensure that no errors occurred during the cloning of the objects.
- In case of an error, the user should fix the cause of the error and call the [COPY\\_TABLE\\_DEPENDENTS Procedure](#) again to clone the dependent object. Alternatively the user can manually clone the dependent object and then register the manually cloned dependent object using the [REGISTER\\_DEPENDENT\\_OBJECT Procedure](#).
- All cloned referential constraints involving the interim tables will be created disabled (they will be automatically enabled after the redefinition) and all triggers on interim tables will not fire till the redefinition is completed. After the redefinition is complete, the cloned objects will be renamed to the corresponding pre-redefinition names of the objects (from which they were cloned from).
- It is the user's responsibility that the cloned dependent objects are unaffected by the redefinition. All the triggers will be cloned and it is the user's responsibility that the cloned triggers are unaffected by the redefinition.

## FINISH\_REDEF\_TABLE Procedure

This procedure completes the redefinition process. Before this step, you can create new indexes, triggers, grants, and constraints on the interim table. The referential constraints involving the interim table must be disabled. After completing this step, the original table is redefined with the attributes and data of the interim table. The original table is locked briefly during this procedure.

### Syntax

```
DBMS_REDEFINITION.FINISH_REDEF_TABLE (  
    uname          IN VARCHAR2,  
    orig_table     IN VARCHAR2,  
    int_table      IN VARCHAR2,  
    part_name      IN VARCHAR2 := NULL);
```

### Parameters

**Table 110–6** FINISH\_REDEF\_TABLE Procedure Parameters

Parameters	Description
uname	Schema name of the tables
orig_table	Name of the table to be redefined
int_table	Name of the interim table
part_name	Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

## REGISTER\_DEPENDENT\_OBJECT Procedure

This procedure registers a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table.

This can be used to have the same object on each table but with different attributes. For example: for an index, the storage and tablespace attributes could be different but the columns indexed remain the same

### Syntax

```
DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT (
  uname           IN VARCHAR2,
  orig_table      IN VARCHAR2,
  int_table       IN VARCHAR2,
  dep_type        IN PLS_INTEGER,
  dep_owner       IN VARCHAR2,
  dep_orig_name   IN VARCHAR2,
  dep_int_name    IN VARCHAR2);
```

### Parameters

**Table 110-7 REGISTER\_DEPENDENT\_OBJECT Procedure Parameters**

Parameters	Description
uname	Schema name of the tables
orig_table	Name of the table to be redefined
int_table	Name of the interim table
dep_type	Type of the dependent object (see <a href="#">Constants</a> on page 110-4 and <a href="#">Operational Notes</a> on page 110-5)
dep_owner	Owner of the dependent object
dep_orig_name	Name of the original dependent object
dep_int_name	Name of the interim dependent object

### Usage Notes

- Attempting to register an already registered object will raise an error.
- Registering a dependent object will automatically remove that object from DBA\_REDEFINITION\_ERRORS if an entry exists for that object.

## START\_REDEF\_TABLE Procedure

Prior to calling this procedure, you must manually create an empty interim table (in the same schema as the table to be redefined) with the desired attributes of the post-redefinition table, and then call this procedure to initiate the redefinition.

### Syntax

```
DBMS_REDEFINITION.START_REDEF_TABLE (
  uname          IN VARCHAR2,
  orig_table     IN VARCHAR2,
  int_table      IN VARCHAR2,
  col_mapping    IN VARCHAR2 := NULL,
  options_flag   IN BINARY_INTEGER := 1,
  orderby_cols  IN VARCHAR2 := NULL,
  part_name     IN VARCHAR2 := NULL);
```

### Parameters

**Table 110–8** START\_REDEF\_TABLE Procedure Parameters

Parameter	Description
uname	Schema name of the tables
orig_table	Name of the table to be redefined
int_table	Name of the interim table
col_mapping	Mapping information from the columns in the original table to the columns in the interim table. (This is similar to the column list on the SELECT clause of a query.) If NULL, all the columns in the original table are selected and have the same name after redefinition.
options_flag	Indicates the type of redefinition method to use: <ul style="list-style-type: none"> <li>■ If <code>dbms_redefinition.cons_use_pk</code>, the redefinition is done using primary keys or pseudo-primary keys (unique keys with all component columns having NOT NULL constraints). The default method of redefinition is using primary keys.</li> <li>■ If <code>dbms_redefinition.cons_use_rowid</code>, the redefinition is done using rowids.</li> </ul>
orderby_cols	This optional parameter accepts the list of columns (along with the optional keyword(s) ascending/descending) with which to order by the rows during the initial instantiation of the interim table (the order by is only done for the initial instantiation and not for subsequent synchronizations)
part_name	Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

## SYNC\_INTERIM\_TABLE Procedure

This procedure keeps the interim table synchronized with the original table.

### Syntax

```
DBMS_REDEFINITION.SYNC_INTERIM_TABLE (
  uname          IN VARCHAR2,
  orig_table     IN VARCHAR2,
  int_table      IN VARCHAR2,
  part_name      IN VARCHAR2 := NULL);
```

### Parameters

**Table 110–9 SYNC\_INTERIM\_TABLE Procedure Parameters**

Parameter	Description
uname	Schema name of the table
orig_table	Name of the table to be redefined
int_table	Name of the interim table
part_name	Name of the partition being redefined. If redefining only a single partition of a table, specify the partition name in this parameter. NULL implies the entire table is being redefined.

### Usage Notes

- This step is useful in minimizing the amount of synchronization needed to be done by the [FINISH\\_REDEF\\_TABLE Procedure](#) before completing the online redefinition.
- This procedure can be called between long running operations (such as CREATE INDEX) on the interim table to sync it up with the data in the original table and speed up subsequent operations.

## UNREGISTER\_DEPENDENT\_OBJECT Procedure

This procedure unregisters a dependent object (index, trigger, constraint or materialized view log) on the table being redefined and the corresponding dependent object on the interim table.

### Syntax

```
DBMS_REDEFINITION.UNREGISTER_DEPEPENDENT_OBJECT (  
    uname           IN VARCHAR2,  
    orig_table      IN VARCHAR2,  
    int_table       IN VARCHAR2,  
    dep_type        IN PLS_INTEGER,  
    dep_owner       IN VARCHAR2,  
    dep_orig_name   IN VARCHAR2,  
    dep_int_name    IN VARCHAR2);
```

### Parameters

**Table 110–10 UNREGISTER\_DEPENDENT\_OBJECT Procedure Parameters**

Parameters	Description
uname	Schema name of the tables
orig_table	Name of the table to be redefined
int_table	Name of the interim table
dep_type	Type of the dependent object
dep_owner	Owner of the dependent object
dep_orig_name	Name of the original dependent object
dep_int_name	Name of the interim dependent object

---

---

## DBMS\_REFRESH

The DBMS\_REFRESH package enables you to create groups of materialized views that can be refreshed together to a transactionally consistent point in time.

- [Documentation of DBMS\\_REFRESH](#)

---

## Documentation of DBMS\_REFRESH

For a complete description of this package within the context of Replication, see DBMS\_REFRESH in the *Oracle Database Advanced Replication Management API Reference*.



The DBMS\_REPAIR package contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes. You can address corruptions where possible and continue to use objects while you attempt to rebuild or repair them.

**See Also:** For detailed information about using the DBMS\_REPAIR package, see *Oracle Database Administrator's Guide*.

This chapter contains the following topics:

- [Using DBMS\\_REPAIR](#)
  - Overview
  - Security Model
  - Constants
  - Operating Notes
  - Exceptions
  - Examples
- [Summary of DBMS\\_REPAIR Subprograms](#)

## Using DBMS\_REPAIR

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Operating Notes](#)
- [Exceptions](#)
- [Examples](#)

## Overview

---

---

**Note:** The DBMS\_REPAIR package is intended for use by database administrators only. It is not intended for use by application developers.

---

---

## Security Model

The package is owned by SYS. Execution privilege is not granted to other users.

## Constants

The DBMS\_REPAIR package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS\_REPAIR.TABLE\_OBJECT.

Table 112-1 lists the parameters and the enumerated constants.

**Table 112-1 DBMS\_REPAIR Parameters with Enumerated Constants**

Parameter	Option	Type	Description
object_type	■ TABLE_OBJECT	BINARY_	-
	■ INDEX_OBJECT	INTEGER	
	■ CLUSTER_OBJECT		
action	■ CREATE_ACTION	BINARY_	-
	■ DROP_ACTION	INTEGER	
	■ PURGE_ACTION		
table_type	■ REPAIR_TABLE	BINARY_	-
	■ ORPHAN_TABLE	INTEGER	
flags	■ SKIP_FLAG	BINARY_	-
	■ NOSKIP_FLAG	INTEGER	
object_id	■ ALL_INDEX_ID := 0	BINARY_ INTEGER	Clean up all objects that qualify
wait_for_lock	■ LOCK_WAIT := 1	BINARY_	Specifies whether to try getting DML locks on underlying table [[sub]partition] object
	■ LOCK_NOWAIT := 0	INTEGER	

---

**Note:** The default table\_name will be REPAIR\_TABLE when table\_type is REPAIR\_TABLE, and will be ORPHAN\_KEY\_TABLE when table\_type is ORPHAN\_TABLE.

---

## Operating Notes

The procedure to create the `ORPHAN_KEYS_TABLE` is similar to the one used to create the `REPAIR_TABLE`.

```
CONNECT / AS SYSDBA;
EXEC DBMS_REPAIR.ADMIN_TABLES('ORPHAN_KEYS_TABLE', DBMS_REPAIR.ORPHAN_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);
EXEC DBMS_REPAIR.ADMIN_TABLES('REPAIR_TABLE', DBMS_REPAIR.REPAIR_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);

DESCRIBE ORPHAN_KEYS_TABLE;
DESCRIBE REPAIR_TABLE;
SELECT * FROM ORPHAN_KEYS_TABLE;
SELECT * FROM REPAIR_TABLE;
```

The DBA would create the repair and orphan keys tables once. Subsequent executions of the [CHECK\\_OBJECT Procedure](#) would add rows into the appropriate table indicating the types of errors found.

The name of the repair and orphan keys tables can be chosen by the user, with the following restriction: the name of the repair table must begin with the 'REPAIR\_' prefix, and the name of the orphan keys table must begin with the 'ORPHAN\_' prefix. The following code is also legal:

```
CONNECT / AS SYSDBA;
EXEC DBMS_REPAIR.ADMIN_TABLES('ORPHAN_FOOBAR', DBMS_REPAIR.ORPHAN_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);
EXEC DBMS_REPAIR.ADMIN_TABLES('REPAIR_ABCD', DBMS_REPAIR.REPAIR_TABLE,
                              DBMS_REPAIR.CREATE_ACTION);

DESCRIBE ORPHAN_FOOBAR;
DESCRIBE REPAIR_ABCD;
SELECT * FROM ORPHAN_FOOBAR;
SELECT * FROM REPAIR_ABCD;
```

When invoking the [CHECK\\_OBJECT Procedure](#) the name of the repair and orphan keys tables that were created should be specified correctly, especially if the default values were not used in the [ADMIN\\_TABLES Procedure](#) or `CREATE_ACTION`.

Other actions in the [ADMIN\\_TABLES Procedure](#) can be used to purge/delete the `REPAIR_TABLE` and the `ORPHAN_KEYS_TABLE`.

## Exceptions

**Table 112–2 DBMS\_REPAIR Exceptions**

Exception	Description	Action
942	Reported by DBMS_REPAIR.ADMIN_TABLES during a DROP_ACTION when the specified table doesn't exist.	-
955	Reported by DBMS_REPAIR.CREATE_ACTION when the specified table already exists.	-
24120	An invalid parameter was passed to the specified DBMS_REPAIR procedure.	Specify a valid parameter value or use the parameter's default.
24122	An incorrect block range was specified.	Specify correct values for the BLOCK_START and BLOCK_END parameters.
24123	An attempt was made to use the specified feature, but the feature is not yet implemented.	Do not attempt to use the feature.
24124	An invalid ACTION parameter was specified.	Specify CREATE_ACTION, PURGE_ACTION or DROP_ACTION for the ACTION parameter.
24125	An attempt was made to fix corrupt blocks on an object that has been dropped or truncated since DBMS_REPAIR.CHECK_OBJECT was run.	Use DBMS_REPAIR.ADMIN_TABLES to purge the repair table and run DBMS_REPAIR.CHECK_OBJECT to determine whether there are any corrupt blocks to be fixed.
24127	TABLESPACE parameter specified with an ACTION other than CREATE_ACTION.	Do not specify TABLESPACE when performing actions other than CREATE_ACTION.
24128	A partition name was specified for an object that is not partitioned.	Specify a partition name only if the object is partitioned.
24129	An attempt was made to pass a table name parameter without the specified prefix.	Pass a valid table name parameter.
24130	An attempt was made to specify a repair or orphan table that does not exist.	Specify a valid table name parameter.
24131	An attempt was made to specify a repair or orphan table that does not have a correct definition.	Specify a table name that refers to a properly created table.
24132	An attempt was made to specify a table name is greater than 30 characters long.	Specify a valid table name parameter.

## Examples

```
/* Fix the bitmap status for all the blocks in table mytab in schema sys */  
  
EXECUTE DBMS_REPAIR.SEGMENT_FIX_STATUS('SYS', 'MYTAB');  
  
/* Mark block number 45, filenumber 1 for table mytab in sys schema as FULL.*/  
  
EXECUTE DBMS_REPAIR.SEGMENT_FIX_STATUS('SYS', 'MYTAB', TABLE_OBJECT,1, 45, 1);
```



---

## Summary of DBMS\_REPAIR Subprograms

**Table 112-3 DBMS\_REPAIR Package Subprograms**

Subprogram	Description
<a href="#">ADMIN_TABLES Procedure</a> on page 112-10	Provides administrative functions for the DBMS_REPAIR package repair and orphan key tables, including create, purge, and drop functions
<a href="#">CHECK_OBJECT Procedure</a> on page 112-11	Detects and reports corruptions in a table or index
<a href="#">DUMP_ORPHAN_KEYS Procedure</a> on page 112-13	Reports on index entries that point to rows in corrupt data blocks
<a href="#">FIX_CORRUPT_BLOCKS Procedure</a> on page 112-14	Marks blocks software corrupt that have been previously detected as corrupt by CHECK_OBJECT
<a href="#">ONLINE_INDEX_CLEAN Function</a> on page 112-15	Performs a manual cleanup of failed or interrupted online index builds or rebuilds
<a href="#">REBUILD_FREELISTS Procedure</a> on page 112-16	Rebuilds an object's freelists
<a href="#">SEGMENT_FIX_STATUS Procedure</a> on page 112-17	Fixes the corrupted state of a bitmap entry
<a href="#">SKIP_CORRUPT_BLOCKS Procedure</a> on page 112-18	Sets whether to ignore blocks marked corrupt during table and index scans or to report ORA-1578 when blocks marked corrupt are encountered

## ADMIN\_TABLES Procedure

This procedure provides administrative functions for the DBMS\_REPAIR package repair and orphan key tables.

### Syntax

```
DBMS_REPAIR.ADMIN_TABLES (
  table_name IN VARCHAR2,
  table_type IN BINARY_INTEGER,
  action      IN BINARY_INTEGER,
  tablespace IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 112–4 ADMIN\_TABLES Procedure Parameters**

Parameter	Description
table_name	Name of the table to be processed. Defaults to ORPHAN_KEY_TABLE or REPAIR_TABLE based on the specified table_type. When specified, the table name must have the appropriate prefix: ORPHAN_ or REPAIR_.
table_type	Type of table; must be either ORPHAN_TABLE or REPAIR_TABLE.  See " <a href="#">Constants</a> " on page 112-5.
action	Indicates what administrative action to perform.  Must be either CREATE_ACTION, PURGE_ACTION, or DROP_ACTION. If the table already exists, and if CREATE_ACTION is specified, then an error is returned. PURGE_ACTION indicates to delete all rows in the table that are associated with non-existent objects. If the table does not exist, and if DROP_ACTION is specified, then an error is returned.  When CREATE_ACTION and DROP_ACTION are specified, an associated view named DBA_<table_name> is created and dropped respectively. The view is defined so that rows associated with non-existent objects are eliminated.  Created in the SYS schema.  See " <a href="#">Constants</a> " on page 112-5.
tablespace	Indicates the tablespace to use when creating a table.  By default, the SYS default tablespace is used. An error is returned if the tablespace is specified and if the action is not CREATE_ACTION.

## CHECK\_OBJECT Procedure

This procedure checks the specified objects and populates the repair table with information about corruptions and repair directives.

Validation consists of block checking all blocks in the object.

### Syntax

```
DBMS_REPAIR.CHECK_OBJECT (
  schema_name      IN VARCHAR2,
  object_name      IN VARCHAR2,
  partition_name   IN VARCHAR2      DEFAULT NULL,
  object_type      IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
  repair_table_name IN VARCHAR2      DEFAULT 'REPAIR_TABLE',
  flags           IN BINARY_INTEGER DEFAULT NULL,
  relative_fno    IN BINARY_INTEGER DEFAULT NULL,
  block_start     IN BINARY_INTEGER DEFAULT NULL,
  block_end       IN BINARY_INTEGER DEFAULT NULL,
  corrupt_count   OUT BINARY_INTEGER);
```

### Parameters

**Table 112-5 CHECK\_OBJECT Procedure Parameters**

Parameter	Description
schema_name	Schema name of the object to be checked.
object_name	Name of the table or index to be checked.
partition_name	Partition or subpartition name to be checked.  If this is a partitioned object, and if <code>partition_name</code> is not specified, then all partitions and subpartitions are checked. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are checked.
object_type	Type of the object to be processed. This must be either <code>TABLE_OBJECT</code> (default) or <code>INDEX_OBJECT</code> .  See " <a href="#">Constants</a> " on page 112-5.
repair_table_name	Name of the repair table to be populated.  The table must exist in the <code>SYS</code> schema. Use the <code>ADMIN_TABLES</code> Procedure to create a repair table. The default name is <code>REPAIR_TABLE</code> .
flags	Reserved for future use.
relative_fno	Relative file number: Used when specifying a block range.
block_start	First block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.
block_end	Last block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition. If only one of <code>block_start</code> or <code>block_end</code> is specified, then the other defaults to the first or last block in the file respectively.
corrupt_count	Number of corruptions reported.

## Usage Notes

You may optionally specify a DBA range, partition name, or subpartition name when you want to check a portion of an object.

## DUMP\_ORPHAN\_KEYS Procedure

This procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan table.

If the repair table is specified, then any corrupt blocks associated with the base table are handled in addition to all data blocks that are marked software corrupt. Otherwise, only blocks that are marked corrupt are handled.

This information may be useful for rebuilding lost rows in the table and for diagnostic purposes.

### Syntax

```
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
  schema_name      IN  VARCHAR2,
  object_name      IN  VARCHAR2,
  partition_name   IN  VARCHAR2      DEFAULT NULL,
  object_type      IN  BINARY_INTEGER DEFAULT INDEX_OBJECT,
  repair_table_name IN  VARCHAR2      DEFAULT 'REPAIR_TABLE',
  orphan_table_name IN  VARCHAR2      DEFAULT 'ORPHAN_KEYS_TABLE',
  flags            IN  BINARY_INTEGER DEFAULT NULL,
  key_count        OUT BINARY_INTEGER);
```

### Parameters

**Table 112–6 DUMP\_ORPHAN\_KEYS Procedure Parameters**

Parameter	Description
schema_name	Schema name.
object_name	Object name.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if <code>partition_name</code> is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. The default is <code>INDEX_OBJECT</code> . See " <a href="#">Constants</a> " on page 112-5.
repair_table_name	Name of the repair table that has information regarding corrupt blocks in the base table. The specified table must exist in the <code>SYS</code> schema. The <code>ADMIN_TABLES</code> Procedure is used to create the table.
orphan_table_name	Name of the orphan key table to populate with information regarding each index entry that refers to a row in a corrupt data block. The specified table must exist in the <code>SYS</code> schema. The <code>ADMIN_TABLES</code> Procedure is used to create the table.
flags	Reserved for future use.
key_count	Number of index entries processed.

## FIX\_CORRUPT\_BLOCKS Procedure

This procedure fixes the corrupt blocks in specified objects based on information in the repair table that was previously generated by the [CHECK\\_OBJECT Procedure](#).

Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is effected, the associated row in the repair table is updated with a fix timestamp.

### Syntax

```
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
  schema_name      IN  VARCHAR2,
  object_name      IN  VARCHAR2,
  partition_name   IN  VARCHAR2      DEFAULT NULL,
  object_type      IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
  repair_table_name IN  VARCHAR2      DEFAULT 'REPAIR_TABLE',
  flags           IN  BINARY_INTEGER DEFAULT NULL,
  fix_count       OUT BINARY_INTEGER);
```

### Parameters

**Table 112–7** *FIX\_CORRUPT\_BLOCKS Procedure Parameters*

Parameter	Description
schema_name	Schema name.
object_name	Name of the object with corrupt blocks to be fixed.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if <code>partition_name</code> is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either <code>TABLE_OBJECT</code> (default) or <code>INDEX_OBJECT</code> . See " <a href="#">Constants</a> " on page 112-5.
repair_table_name	Name of the repair table with the repair directives. Must exist in the <code>SYS</code> schema.
flags	Reserved for future use.
fix_count	Number of blocks fixed.

## ONLINE\_INDEX\_CLEAN Function

This function performs a manual cleanup of failed or interrupted online index builds or rebuilds. This action is also performed periodically by SMON, regardless of user-initiated cleanup.

This function returns TRUE if all indexes specified were cleaned up and FALSE if one or more indexes could not be cleaned up.

### Syntax

```
DBMS_REPAIR.ONLINE_INDEX_CLEAN (
    object_id      IN BINARY_INTEGER DEFAULT ALL_INDEX_ID,
    wait_for_lock  IN BINARY_INTEGER DEFAULT LOCK_WAIT)
RETURN BOOLEAN;
```

### Parameters

**Table 112–8 ONLINE\_INDEX\_CLEAN Function Parameters**

Parameter	Description
object_id	Object id of index to be cleaned up. The default cleans up all object ids that qualify.
wait_for_lock	This parameter specifies whether to try getting DML locks on underlying table [[sub]partition] object. The default retries up to an internal retry limit, after which the lock get will give up. If LOCK_NOWAIT is specified, then the lock get does not retry.

## REBUILD\_FREELISTS Procedure

This procedure rebuilds the freelists for the specified object. All free blocks are placed on the master freelist. All other freelists are zeroed.

If the object has multiple freelist groups, then the free blocks are distributed among all freelists, allocating to the different groups in round-robin fashion.

### Syntax

```
DBMS_REPAIR.REBUILD_FREELISTS (
  schema_name      IN VARCHAR2,
  object_name      IN  VARCHAR2,
  partition_name   IN VARCHAR2 DEFAULT NULL,
  object_type      IN BINARY_INTEGER DEFAULT TABLE_OBJECT);
```

### Parameters

**Table 112–9 REBUILD\_FREELISTS Procedure Parameters**

Parameter	Description
schema_name	Schema name.
object_name	Name of the object whose freelists are to be rebuilt.
partition_name	Partition or subpartition name whose freelists are to be rebuilt. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See " <a href="#">Constants</a> " on page 112-5.



## SEGMENT\_FIX\_STATUS Procedure

With this procedure you can fix the corrupted state of a bitmap entry. The procedure either recalculates the state based on the current contents of the corresponding block or sets the state to a specific value.

### Syntax

```
DBMS_REPAIR.SEGMENT_FIX_STATUS (
  segment_owner  IN VARCHAR2,
  segment_name   IN VARCHAR2,
  segment_type   IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
  file_number    IN BINARY_INTEGER DEFAULT NULL,
  block_number   IN BINARY_INTEGER DEFAULT NULL,
  status_value   IN BINARY_INTEGER DEFAULT NULL,
  partition_name IN VARCHAR2 DEFAULT NULL,);
```

### Parameters

**Table 112–10** *SEGMENT\_FIX\_STATUS Procedure Parameters*

Parameter	Description
schema_owner	Schema name of the segment.
segment_name	Segment name.
partition_name	Optional. Name of an individual partition. NULL for nonpartitioned objects. Default is NULL.
segment_type	Optional Type of the segment (for example, TABLE_OBJECT or INDEX_OBJECT). Default is NULL.
file_number	(optional) The tablespace-relative file number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.
block_number	(optional) The file-relative block number of the data block whose status has to be fixed. If omitted, all the blocks in the segment will be checked for state correctness and fixed.
status_value	(optional) The value to which the block status described by the file_number and block_number will be set. If omitted, the status will be set based on the current state of the block. This is almost always the case, but if there is a bug in the calculation algorithm, the value can be set manually. Status values: <ul style="list-style-type: none"> <li>■ 1 = block is full</li> <li>■ 2 = block is 0-25% free</li> <li>■ 3 = block is 25-50% free</li> <li>■ 4 = block is 50-75% free</li> <li>■ 5 = block is 75-100% free</li> </ul> The status for bitmap blocks, segment headers, and extent map blocks cannot be altered. The status for blocks in a fixed hash area cannot be altered. For index blocks, there are only two possible states: 1 = block is full and 3 = block has free space.

## SKIP\_CORRUPT\_BLOCKS Procedure

This procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object.

When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

---



---

**Note:** When Oracle performs an index range scan on a corrupt index after DBMS\_REPAIR.SKIP\_CORRUPT\_BLOCKS has been set for the base table, corrupt branch blocks and root blocks are not skipped. Only corrupt non-root leaf blocks are skipped.

---



---

### Syntax

```
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
    schema_name  IN VARCHAR2,
    object_name  IN VARCHAR2,
    object_type  IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
    flags        IN BINARY_INTEGER DEFAULT SKIP_FLAG);
```

### Parameters

**Table 112-11** SKIP\_CORRUPT\_BLOCKS Procedure Parameters

Parameter	Description
schema_name	Schema name of the object to be processed.
object_name	Name of the object.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or CLUSTER_OBJECT. See " <a href="#">Constants</a> " on page 112-5.
flags	If SKIP_FLAG is specified, then it turns on the skip of software corrupt blocks for the object during index and table scans. If NOSKIP_FLAG is specified, then scans that encounter software corrupt blocks return an ORA-1578. See " <a href="#">Constants</a> " on page 112-5.

The DBMS\_REPCAT package provides routines to administer and update the replication catalog and environment.

- [Documentation of DBMS\\_REPCAT](#)

---

## Documentation of DBMS\_REPCAT

For a complete description of this package within the context of Replication, see DBMS\_REPCAT in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_REPCAT\_ADMIN

The DBMS\_REPCAT\_ADMIN package enables you to create users with the privileges needed by the symmetric replication facility.

- [Documentation of DBMS\\_REPCAT\\_ADMIN](#)

---

## Documentation of DBMS\_REPCAT\_ADMIN

For a complete description of this package within the context of Replication, see DBMS\_REPCAT\_ADMIN in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_REPCAT\_INSTANTIATE

The DBMS\_REPCAT\_INSTANTIATE package instantiates deployment templates.

- [Documentation of DBMS\\_REPCAT\\_INSTANTIATE](#)

---

## Documentation of DBMS\_REPCAT\_INSTANTIATE

For a complete description of this package within the context of Replication, see DBMS\_REPCAT\_INSTANTIATE in the *Oracle Database Advanced Replication Management API Reference*.



---

---

## DBMS\_REPCAT\_RGT

The DBMS\_REPCAT\_RGT package controls the maintenance and definition of refresh group templates.

- [Documentation of DBMS\\_REPCAT\\_RGT](#)

---

## Documentation of DBMS\_REPCAT\_RGT

For a complete description of this package within the context of Replication, see DBMS\_REPCAT\_RGT in the *Oracle Database Advanced Replication Management API Reference*.

---

---

## DBMS\_REPUTIL

The DBMS\_REPUTIL package contains subprograms to generate shadow tables, triggers, and packages for table replication, as well as subprograms to generate wrappers for replication of standalone procedure invocations and packaged procedure invocations. This package is referenced only by the generated code.

- [Documentation of DBMS\\_REPUTIL](#)

---

## Documentation of DBMS\_REPUTIL

For a complete description of this package within the context of Replication, see DBMS\_REPUTIL in the *Oracle Database Advanced Replication Management API Reference*.

The DBMS\_RESCONFIG package provides an interface to operate on the resource configuration list, and to retrieve listener information for a resource.

**See Also:** *Oracle XML DB Developer's Guide* for more information about "Resource Configuration".

This chapter contains the following topics:

- [Using DBMS\\_RESCONFIG](#)
  - Overview
- [Summary of DBMS\\_RESCONFIG Subprograms](#)

## Using DBMS\_RESCONFIG

---

- [Overview](#)

## Overview

The DBMS\_RESCONFIG package contains functions and procedures to manage the resource configuration lists of individual resources and the repository.

---

## Summary of DBMS\_RESCONFIG Subprograms

This table lists the package subprograms in alphabetical order.

**Table 118–1 DBMS\_RESCONFIG Package Subprograms**

Subprogram	Description
<a href="#">ADDREPOSITORYRESCONFIG Procedure</a> on page 118-5	Inserts the resource configuration specified by absolute path at the given position of the repository's configuration list
<a href="#">ADDRESCONFIG Procedure</a> on page 118-6	Inserts the resource configuration specified by the absolute path at the given position in the target resource's configuration list
<a href="#">APPENDRESCONFIG Procedure</a> on page 118-7	Appends the resource configuration specified by rpath to the target resource's configuration list if it is not already included in the list
<a href="#">DELETEREPOSITORYRESCONFIG Procedure</a> on page 118-8	Removes the configuration at the given position in the repository's configuration list.
<a href="#">DELETERESCONFIG Procedures</a> on page 118-9	Removes the configuration at the given position in the target resource's configuration list. 1
<a href="#">GETLISTENERS Function</a> on page 118-10	Returns the list of listeners applicable for a given resource
<a href="#">GETREPOSITORYRESCONFIG Function</a> on page 118-11	Returns the resource configuration at the specified position of the repository's configuration list
<a href="#">GETREPOSITORYRESCONFIGPATHS Function</a> on page 118-12	Returns a list of resource configuration paths defined for the repository
<a href="#">GETRESCONFIG Function</a> on page 118-13	Returns the resource configuration at the specified position of the target resource's configuration list
<a href="#">GETRESCONFIGPATHS Function</a> on page 118-14	Returns a list of resource configuration paths defined in the target resource's configuration list
<a href="#">PATCHREPOSITORYRESCONFIGLIST Procedure</a> on page 118-15	Removes invalid references from the repository resource configuration list, and makes the repository available



## ADDREPOSITORYRESCONFIG Procedure

This procedure inserts the resource configuration specified by absolute path of the resource configuration at the specified position of the repository's configuration list. It shifts the element currently at that position (if any) and any subsequent elements to the right.

### Syntax

```
DBMS_RESCONFIG.ADDREPOSITORYRESCONFIG (
  rcpath      IN   VARCHAR2,
  pos         IN   PLS_INTEGER := NULL);
```

### Parameters

**Table 118–2 ADDREPOSITORYRESCONFIG Function Parameters**

Parameter	Description
rcpath	Absolute path of the resource configuration to be inserted. An exception is raised if <code>rcpath</code> already exists in the target's configuration list.
pos	Index at which the new configuration is to be inserted. If this parameter is not specified then the new configuration is appended to the end of the list. An exception is raised if the index is out of range ( <code>pos &lt; 0</code> or <code>pos &gt;</code> the size of the target resource's configuration list).

### Usage Notes

- An error is raised if the document referenced by `rcpath` is not based on `XDBResConfig.xsd` schema.
- Users must have `XDBADMIN` role and `READ` privilege on the resource configuration to be inserted; otherwise, an error is returned.

## ADDRESCONFIG Procedure

This procedure inserts the resource configuration specified by the absolute path of the resource configuration at the given position in the target resource's configuration list. It shifts the element currently at that position (if any) and any subsequent elements to the right.

### Syntax

```
DBMS_RESCONFIG.ADDRESCONFIG(
  respath   IN   VARCHAR2,
  rcpath    IN   VARCHAR2,
  pos       IN   PLS_INTEGER := NULL);
```

### Parameters

**Table 118–3 ADDRESCONFIG Function Parameters**

Parameter	Description
respath	Absolute path of the target resource
rcpath	Absolute path of the resource configuration to be inserted. An exception is raised if rcpath already exists in the target's configuration list.
pos	Index at which the new configuration is to be inserted. If this parameter is not specified then the new configuration is appended to the end of the list. An exception is raised if the index is out of range ( $pos < 0$ or $pos >$ the size of the target resource's configuration list).

### Usage Notes

- An error is raised if the document referenced by `rcpath` is not based on `XDBResConfig.xsd` schema.
- Users must have `WRITE-CONFIG` privilege on the target resource and read privilege on the resource configuration to be inserted; otherwise, an error is returned.

## APPENDRESCONFIG Procedure

This procedure appends the resource configuration specified by `rspath` to the target resource's configuration list if it is not already included in the list.

### Syntax

```
DBMS_RESCONFIG.ADDRESCONFIG(
  respath      IN   VARCHAR2,
  rspath       IN   VARCHAR2,
  appendOption IN   PLS_INTEGER);
```

### Parameters

**Table 118–4 ADDRESCONFIG Function Parameters**

Parameter	Description
<code>respath</code>	Absolute path of the target resource
<code>rspath</code>	Absolute path of the resource configuration to be appended at the end of the target's configuration list. If <code>rspath</code> already exists in the list then nothing is appended.
<code>appendOption</code>	Either <code>APPEND_RESOURCE</code> or <code>APPEND_RECURSIVE</code> . If <code>APPEND_RESOURCE</code> is specified then only the target resource is affected. If <code>APPEND_RECURSIVE</code> is specified then the target resource and all its descendents will be affected.

### Usage Notes

- An error is raised if the document referenced by `rspath` is not based on `XDBResConfig.xsd` schema.
- Users must have `WRITE-CONFIG` privilege on all affected resources and required read privilege on the resource configuration to be inserted; otherwise, an error is returned.

## DELETEREPOSITORYRESCONFIG Procedure

This procedure removes the configuration at the given position in the repository's configuration list. It shifts any subsequent elements to the left.

### Syntax

```
DBMS_RESCONFIG.DELETEREPOSITORYRESCONFIG(  
    pos          IN    PLS_INTEGER);
```

### Parameters

**Table 118–5 DELETEREPOSITORYRESCONFIG Function Parameters**

Parameter	Description
pos	The index of the configuration to be removed. An exception is raised if the index is out of range ( <code>pos &lt; 0</code> or <code>pos &gt;=</code> the size of the target resource's configuration list).

### Usage Notes

- Users must have `XDBADMIN` role to execute this.
- This statement is treated as if it is a DDL statement. This means the system will implicitly commit before and after this statement.

## DELETERESCONFIG Procedures

This procedure removes the configuration at the given position in the target resource's configuration list. It shifts any subsequent elements to the left. Users can use the overloaded for recursive deletion.

### Syntax

```
DBMS_RESCONFIG.DELETERESCONFIG(
  respath      IN  VARCHAR2,
  pos          IN  PLS_INTEGER);
```

```
DBMS_RESCONFIG.DELETERESCONFIG(
  respath      IN  VARCHAR2,
  rcpath       IN  VARCHAR2,
  deleteOption IN  PLS_INTEGER);
```

### Parameters

**Table 118–6 DELETERESCONFIG Procedure Parameters**

Parameter	Description
respath	Absolute path of the target resource
pos	The index of the configuration to be removed. An exception is raised if the index is out of range ( <code>pos &lt; 0</code> or <code>pos &gt;=</code> the size of the target resource's configuration list).
rcpath	Absolute path of the resource configuration to be deleted if found in list.
deleteOption	Either <code>DELETE_RESOURCE</code> or <code>DELETE_RECURSIVE</code> . If <code>DELETE_RESOURCE</code> is specified then only the configuration list of the target resource is affected. If <code>DELETE_RECURSIVE</code> is specified then the configuration list of the target resource and all its descendents will be affected.

### Usage Notes

Users must have `WRITE-CONFIG` privilege on the target resource to execute this.

## GETLISTENERS Function

This function returns the list of listeners applicable for a given resource.

The value returned by this function is an XML document containing the `<event-listeners>` element of the `XDBResconfig.xsd` schema. It contains all the listeners applicable to the target resource, including repository-level listeners. From the returned XML document users can use the `EXTRACT` operator to retrieve the listeners defined for a specific event.

### Syntax

```
DBMS_RESCONFIG.GETLISTENERS(  
    path    IN    VARCHAR2)  
RETURN XMLTYPE;
```

### Parameters

**Table 118–7** *GETLISTENERS Function Parameters*

Parameter	Description
path	Absolute path of the target resource

### Usage Notes

Users must have the required access privilege on all resource configurations referenced by the repository and the target resource; otherwise, an error is returned.

## GETREPOSITORYRESCONFIG Function

This function returns the resource configuration at the specified position of the repository's configuration list.

### Syntax

```
DBMS_RESCONFIG.GETREPOSITORYRESCONFIG(  
    pos    IN    PLS_INTEGER)  
RETURN XMLTYPE;
```

### Parameters

**Table 118–8** *GETREPOSITORYRESCONFIG Function Parameters*

Parameter	Description
pos	Index of element to return. An exception is raised if the index is out of range ( $pos < 0$ or $pos \geq$ the size of the repository's configuration list).

### Usage Notes

Users must have the required read privilege on the requested resource configuration; otherwise, an error is returned.

## GETREPOSITORYRESCONFIGPATHS Function

This function returns a list of resource configuration paths defined for the repository.

### Syntax

```
DBMS_RESCONFIG.GETREPOSITORYRESCONFIGPATHS  
RETURN XDB$STRING_LIST_T;
```

### Usage Notes

Users must be able to access all the referenced resource configurations; otherwise, an error is returned.



## GETRESCONFIG Function

This function returns the resource configuration at the specified position of the target resource's configuration list.

### Syntax

```
DBMS_RESCONFIG.GETRESCONFIG(
    respath IN VARCHAR2,
    pos IN PLS_INTEGER)
RETURN XMLTYPE;
```

### Parameters

**Table 118–9** *GETRESCONFIG Function Parameters*

Parameter	Description
respath	Absolute path of the target resource
pos	Index of element to return. An exception is raised if the index is out of range ( <code>pos &lt; 0</code> or <code>pos &gt;=</code> the size of the target resource's configuration list).

### Usage Notes

Users must have the required read privilege on the requested resource configuration; otherwise, an error is returned.

## GETRESCONFIGPATHS Function

This function returns a list of resource configuration paths defined in the target resource's configuration list.

### Syntax

```
DBMS_RESCONFIG.GETRESCONFIGPATHS (  
    respath IN VARCHAR2)  
RETURN XDB$STRING_LIST_T;
```

### Parameters

**Table 118–10** *GETRESCONFIGPATHS Function Parameters*

Parameter	Description
respath	Absolute path of the target resource

### Usage Notes

Users must be able to access all the referenced resource configurations; otherwise, an error is returned.

## PATCHREPOSITORYRESCONFIGLIST Procedure

Under normal circumstances, deletion of a resource configuration resource cannot be performed if it is part of the repository resource configuration list. If, for some reason, the deletion of a resource configuration resource that is part of the repository resource configuration list succeeds, then any repository operation results in a 'dangling reference' error. This procedure removes invalid references from the repository resource configuration list, and makes the repository available. This procedure must be run as SYS.

### Syntax

```
DBMS_RESCONFIG.PATCHREPOSITORYRESCONFIGLIST;
```



---

---

## DBMS\_RESOURCE\_MANAGER

The DBMS\_RESOURCE\_MANAGER package maintains plans, consumer groups, and plan directives. It also provides semantics so that you may group together changes to the plan schema.

**See Also:** For more information on using the Database Resource Manager, see *Oracle Database Administrator's Guide*.

This chapter contains the following topics:

- [Using DBMS\\_RESOURCE\\_MANAGER](#)
  - Deprecated Subprograms
  - Security Model
  - Constants
- [Summary of DBMS\\_RESOURCE\\_MANAGER Subprograms](#)

---

## Using DBMS\_RESOURCE\_MANAGER

- [Deprecated Subprograms](#)
- [Security Model](#)
- [Constants](#)

## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

---

The following subprograms are deprecated with Oracle Database 11g:

- [SET\\_INITIAL\\_CONSUMER\\_GROUP Procedure](#)

## Security Model

The invoker must have the `ADMINISTER_RESOURCE_MANAGER` system privilege to execute these procedures. The procedures to grant and revoke this privilege are in the package [Chapter 120, "DBMS\\_RESOURCE\\_MANAGER\\_PRIVS"](#).



## Constants

**Table 119–1 DBMS\_RESOURCE\_MANAGER Constants**

Constant	Type	Value	Description
CLIENT_MACHINE	VARCHAR2 (30)	CLIENT_MACHINE	Name of the computer from which the client is making the connection
CLIENT_OS_USER	VARCHAR2 (30)	CLIENT_OS_USER	Operating system user name of the client that is logging in
CLIENT_PROGRAM	VARCHAR2 (30)	CLIENT_PROGRAM	Name of the client program used to log in to the server
MODULE_NAME	VARCHAR2 (30)	MODULE_NAME	Module name in the currently running application as set by the <a href="#">SET_MODULE Procedure</a> in the <a href="#">DBMS_APPLICATION_INFO</a> package, or the equivalent OCI attribute setting
MODULE_NAME_ACTION	VARCHAR2 (30)	MODULE_NAME_ACTION	<p>A combination of the current module and the action being performed as set by either of the following procedures in the <a href="#">DBMS_APPLICATION_INFO</a> package, or their equivalent OCI attribute setting:</p> <ul style="list-style-type: none"> <li>▪ <a href="#">SET_MODULE Procedure</a></li> <li>▪ <a href="#">SET_ACTION Procedure</a></li> </ul> <p>The attribute is specified as the module name followed by a period (.), followed by the action name (<i>module_name.action_name</i>).</p>
ORACLE_FUNCTION	VARCHAR2 (30)	ORACLE_FUNCTION	Function the session is currently executing. Valid functions are the BACKUP, COPY, and DATALOAD. BACKUP is set for sessions that are doing backup operations using RMAN. COPY is set for sessions that are doing image copies using RMAN. DATALOAD is set for sessions that are loading data using datapump.
ORACLE_USER	VARCHAR2 (30)	ORACLE_USER	Oracle Database user name
SERVICE_MODULE	VARCHAR2 (30)	SERVICE_MODULE	Combination of service and module names in this form: <i>service_name.module_name</i>
SERVICE_MODULE_ACTION	VARCHAR2 (30)	SERVICE_MODULE_ACTION	Combination of service name, module name, and action name, in this form: <i>service_name.module_name.action_name</i>
SERVICE_NAME	VARCHAR2 (30)	SERVICE_NAME	Service name used by the client to establish a connection

**Table 119-1 (Cont.) DBMS\_RESOURCE\_MANAGER Constants**

<b>Constant</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
PERFORMANCE_ CLASS	VARCHAR2 (30)	PERFORMANCE_ CLASS	Oracle Database user name

## Summary of DBMS\_RESOURCE\_MANAGER Subprograms

**Table 119–2 DBMS\_RESOURCE\_MANAGER Package Subprograms**

Subprogram	Description
<a href="#">BEGIN_SQL_BLOCK Procedure</a> on page 119-9	Indicates the start of a block of SQL statements to be treated as a group by resource manager
<a href="#">CALIBRATE_IO Procedure</a> on page 119-10	Calibrates the I/O capabilities of storage
<a href="#">CLEAR_PENDING_AREA Procedure</a> on page 119-13	Clears the work area for the resource manager
<a href="#">CREATE_CATEGORY Procedure</a> on page 119-14	Creates a new resource consumer group category
<a href="#">CREATE_CONSUMER_GROUP Procedure</a> on page 119-15	Creates entries which define resource consumer groups
<a href="#">CREATE_PENDING_AREA Procedure</a> on page 119-16	Creates a work area for changes to resource manager objects
<a href="#">CREATE_PLAN Procedure</a> on page 119-18	Creates entries which define resource plans
<a href="#">CREATE_PLAN_DIRECTIVE Procedure</a> on page 119-19	Creates resource plan directives
<a href="#">CREATE_SIMPLE_PLAN Procedure</a> on page 119-23	Creates a single-level resource plan containing up to eight consumer groups in one step
<a href="#">DELETE_CATEGORY Procedure</a> on page 119-25	Deletes an existing resource consumer group category
<a href="#">DELETE_CONSUMER_GROUP Procedure</a> on page 119-26	Deletes entries which define resource consumer groups
<a href="#">DELETE_PLAN Procedure</a> on page 119-27	Deletes the specified plan as well as all the plan directives it refers to
<a href="#">DELETE_PLAN_CASCADE Procedure</a> on page 119-28	Deletes the specified plan as well as all its descendants (plan directives, subplans, consumer groups)
<a href="#">DELETE_PLAN_DIRECTIVE Procedure</a> on page 119-29	Deletes resource plan directives
<a href="#">END_SQL_BLOCK Procedure</a> on page 119-30	Indicates the end of a block of SQL statements that should be treated as a group by resource manager
<a href="#">SET_CONSUMER_GROUP_MAPPING Procedure</a> on page 119-31	Adds, deletes, or modifies entries for the login and run-time attribute mappings
<a href="#">SET_CONSUMER_GROUP_MAPPING_PRI Procedure</a> on page 119-32	Creates the session attribute mapping priority list
<a href="#">SET_INITIAL_CONSUMER_GROUP Procedure</a> on page 119-33	Assigns the initial resource consumer group for a user (Caution: Deprecated Subprogram)
<a href="#">SUBMIT_PENDING_AREA Procedure</a> on page 119-34	Submits pending changes for the resource manager

**Table 119–2 (Cont.) DBMS\_RESOURCE\_MANAGER Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SWITCH_CONSUMER_GROUP_FOR_SESS Procedure</a> on page 119-35	Changes the resource consumer group of a specific session
<a href="#">SWITCH_CONSUMER_GROUP_FOR_USER Procedure</a> on page 119-36	Changes the resource consumer group for all sessions with a given user name
<a href="#">SWITCH_PLAN Procedure</a> on page 119-37	Sets the current resource manager plan
<a href="#">UPDATE_CATEGORY Procedure</a> on page 119-38	Updates an existing resource consumer group category
<a href="#">UPDATE_CONSUMER_GROUP Procedure</a> on page 119-39	Updates entries which define resource consumer groups
<a href="#">UPDATE_PLAN Procedure</a> on page 119-40	Updates entries which define resource plans
<a href="#">UPDATE_PLAN_DIRECTIVE Procedure</a> on page 119-41	Updates resource plan directives
<a href="#">VALIDATE_PENDING_AREA Procedure</a> on page 119-45	Validates pending changes for the resource manager

## BEGIN\_SQL\_BLOCK Procedure

This procedure, to be used with parallel statement queuing, indicates the start of a block of SQL statements that should be treated as a group by resource manager.

---

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

---

### Syntax

```
DBMS_RESOURCE_MANAGER.BEGIN_SQL_BLOCK;
```

### Usage Notes

For more information, see "Parallel Statement Queuing" and "Managing Parallel Statement Queuing with Resource Manager" in *Oracle Database VLDB and Partitioning Guide*.

## CALIBRATE\_IO Procedure

This procedure calibrates the I/O capabilities of storage. Calibration status is available from the `V$IO_CALIBRATION_STATUS` view and results for a successful calibration run are located in `DBA_RSRC_IO_CALIBRATE` table.

### Syntax

```
DBMS_RESOURCE_MANAGER.CALIBRATE_IO (
    num_physical_disks    IN  PLS_INTEGER DEFAULT 1,
    max_latency           IN  PLS_INTEGER DEFAULT 20,
    max_iops              OUT PLS_INTEGER,
    max_mbps              OUT PLS_INTEGER,
    actual_latency        OUT PLS_INTEGER);
```

### Parameters

**Table 119–3 CALIBRATE\_IO Procedure Parameters**

Parameter	Description
<code>num_physical_disks</code>	Approximate number of physical disks in the database storage
<code>max_latency</code>	Maximum tolerable latency in milliseconds for database-block-sized IO requests
<code>max_iops</code>	Maximum number of I/O requests per second that can be sustained. The I/O requests are randomly-distributed, database-block-sized reads.
<code>max_mbps</code>	Maximum throughput of I/O that can be sustained, expressed in megabytes per second. The I/O requests are randomly-distributed, 1 megabyte reads.
<code>actual_latency</code>	Average latency of database-block-sized I/O requests at <code>max_iops</code> rate, expressed in milliseconds

### Usage Notes

- Only users with the `SYSDBA` privilege can run this procedure. Qualified users must also turn on `timed_statistics`, and ensure `asynch_io` is enabled for datafiles. This can be achieved by setting `filesystemio_options` to either `ASYNCH` or `SETALL`. One can also query the `asynch_io` status by means of the following SQL statement:
 

```
col name format a50
SELECT name, asynch_io FROM v$datafile f,v$iostat_file i
WHERE f.file#      = i.file_no
AND  filetype_name = 'Data File'
/
```
- Only one calibration can be run at a time. If another calibration is initiated at the same time, it will fail.
- For an Oracle Real Application Clusters (Oracle RAC) database, the workload is simultaneously generated from all instances.

**See Also:** *Oracle Database Performance Tuning Guide* for more information about calibration

## Examples

### Example of using I/O Calibration procedure

```

SET SERVEROUTPUT ON
DECLARE
    lat INTEGER;
    iops INTEGER;
    mbps INTEGER;
BEGIN
-- DBMS_RESOURCE_MANAGER.CALIBRATE_IO (<DISKS>, <MAX_LATENCY>, iops, mbps, lat);
    DBMS_RESOURCE_MANAGER.CALIBRATE_IO (2, 10, iops, mbps, lat);

    DBMS_OUTPUT.PUT_LINE ('max_iops = ' || iops);
    DBMS_OUTPUT.PUT_LINE ('latency = ' || lat);
    DBMS_OUTPUT.PUT_LINE ('max_mbps = ' || mbps);
end;
/

```

### View for I/O calibration results

```
SQL> desc V$IO_CALIBRATION_STATUS
```

Name	Null?	Type
STATUS		VARCHAR2(13)
CALIBRATION_TIME		TIMESTAMP(3)

```
SQL> desc gv$io_calibration_status
```

Name	Null?	Type
INST_ID		NUMBER
STATUS		VARCHAR2(13)
CALIBRATION_TIME		TIMESTAMP(3)

Column explanation:

-----

STATUS:

IN PROGRESS : Calibration in Progress (Results from previous calibration run displayed, if available)

READY : Results ready and available from earlier run

NOT AVAILABLE : Calibration results not available.

CALIBRATION\_TIME: End time of the last calibration run

### DBA table that stores I/O Calibration results

```
SQL> desc DBA_RSRC_IO_CALIBRATE
```

Name	Null?	Type
START_TIME		TIMESTAMP(6)
END_TIME		TIMESTAMP(6)
MAX_IOPS		NUMBER
MAX_MBPS		NUMBER
MAX_PMBPS		NUMBER
LATENCY		NUMBER
NUM_PHYSICAL_DISKS		NUMBER

comment on table DBA\_RSRC\_IO\_CALIBRATE is  
'Results of the most recent I/O calibration'

/

comment on column DBA\_RSRC\_IO\_CALIBRATE.START\_TIME is

```
'start time of the most recent I/O calibration'  
/  
comment on column DBA_RSRC_IO_CALIBRATE.END_TIME is  
'end time of the most recent I/O calibration'  
/  
comment on column DBA_RSRC_IO_CALIBRATE.MAX_IOPS is  
'maximum number of data-block read requests that can be sustained per second'  
/  
comment on column DBA_RSRC_IO_CALIBRATE.MAX_MBPS is  
'maximum megabytes per second of maximum-sized read requests that can be  
sustained'  
/  
comment on column DBA_RSRC_IO_CALIBRATE.MAX_PMBPS is  
'maximum megabytes per second of large I/O requests that  
can be sustained by a single process'  
/  
comment on column DBA_RSRC_IO_CALIBRATE.LATENCY is  
'latency for data-block read requests'  
/  
comment on column DBA_RSRC_IO_CALIBRATE.NUM_PHYSICAL_DISKS is  
'number of physical disks in the storage subsystem (as specified by user)'  
/
```



## **CLEAR\_PENDING\_AREA Procedure**

This procedure clears pending changes for the resource manager.

### **Syntax**

```
DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

## CREATE\_CATEGORY Procedure

This procedure creates a new consumer group category. The primary purpose of this attribute is to support Exadata I/O Resource Manager category plans. The view `DBA_RSRC_CATEGORIES` defines the currently defined categories. The `ADMINISTRATIVE`, `INTERACTIVE`, `BATCH`, `MAINTENANCE`, and `OTHER` categories are available.

### Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CATEGORY (  
    category    IN    VARCHAR2,  
    comment     IN    VARCHAR2);
```

### Parameters

**Table 119–4** *CREATE\_CATEGORY Procedure Parameters*

Parameter	Description
<code>category</code>	Name of consumer group category
<code>comment</code>	User's comment

## CREATE\_CONSUMER\_GROUP Procedure

This procedure creates entries which define resource consumer groups.

### Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
  consumer_group IN VARCHAR2,
  comment        IN VARCHAR2,
  cpu_mth        IN VARCHAR2 DEFAULT NULL,
  mgmt_mth       IN VARCHAR2 DEFAULT 'ROUND-ROBIN',
  category       IN VARCHAR2 DEFAULT 'OTHER');
```

### Parameters

**Table 119–5** CREATE\_CONSUMER\_GROUP Procedure Parameters

Parameter	Description
consumer_group	Name of the consumer group
comment	User's comment
cpu_mth	Name of CPU resource allocation method (deprecated)
mgmt_mth	Name of CPU resource allocation method
category	Describes the category of the consumer group. The primary purpose of this attribute is to support Exadata I/O Resource Manager category plans. The view <code>DBA_RSRC_CATEGORIES</code> defines the currently defined categories. Categories can be modified, using the <a href="#">CREATE_CATEGORY Procedure</a> , <a href="#">UPDATE_CATEGORY Procedure</a> , and <a href="#">DELETE_CATEGORY Procedure</a> .

## CREATE\_PENDING\_AREA Procedure

This procedure makes changes to resource manager objects.

All changes to the plan schema must be done within a pending area. The pending area can be thought of as a "scratch" area for plan schema changes. The administrator creates this pending area, makes changes as necessary, possibly validates these changes, and only when the submit is completed do these changes become active.

### Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

### Usage Notes

You may, at any time while the pending area is active, view the current plan schema with your changes by selecting from the appropriate user views.

At any time, you may clear the pending area if you want to stop the current changes. You may also call the `VALIDATE` procedure to confirm whether the changes you have made are valid. You do not have to perform your changes in a given order to maintain a consistent group of entries. These checks are also implicitly done when the pending area is submitted.

---

---

**Note:** Oracle allows "orphan" consumer groups (in other words, consumer groups that have no plan directives that refer to them). This is in anticipation that an administrator may want to create a consumer group that is not currently being used, but will be used in the future.

---

---

The following rules must be adhered to, and they are checked whenever the `validate` or `submit` procedures are executed:

- No plan schema may contain any loops.
- All plans and consumer groups referred to by plan directives must exist.
- All plans must have plan directives that refer to either plans or consumer groups.
- All percentages in any given level must not add up to greater than 100 for the emphasis resource allocation method.
- No plan may be deleted that is currently being used as a top plan by an active instance.
- The plan directive parameter, `parallel_degree_limit_p1`, may only appear in plan directives that refer to consumer groups (that is, not at subplans).
- There cannot be more than 28 plan directives coming from any given plan (that is, no plan can have more than 28 children).
- There cannot be more than 28 consumer groups in any active plan schema.
- Plans and consumer groups use the same namespace; therefore, no plan can have the same name as any consumer group.
- There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan schema. This ensures that a session not covered by the currently active plan is allocated resources as specified by the `OTHER_GROUPS` directive.

If any of the preceding rules are broken when checked by the `VALIDATE` or `SUBMIT` procedures, then an informative error message is returned. You may then make changes to fix one or more problems and reissue the `validate` or `submit` procedures.

## CREATE\_PLAN Procedure

This procedure creates entries which define resource plans.

### Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
  plan                IN  VARCHAR2,
  comment             IN  VARCHAR2,
  cpu_mth             IN  VARCHAR2 DEFAULT NULL, -- deprecated
  active_sess_pool_mth IN  VARCHAR2 DEFAULT 'ACTIVE_SESS_POOL_ABSOLUTE',
  parallel_degree_limit_mth IN  VARCHAR2 DEFAULT
    'PARALLEL_DEGREE_LIMIT_ABSOLUTE',
  queueing_mth       IN  VARCHAR2 DEFAULT 'FIFO_TIMEOUT',
  mgmt_mth           IN  VARCHAR2 DEFAULT 'EMPHASIS',
  sub_plan           IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 119–6 CREATE\_PLAN Procedure Parameters**

Parameter	Description
plan	Name of the resource plan
comment	User's comment
cpu_mth	Allocation method for CPU resources (deprecated)
active_sess_pool_mth	Active session pool resource allocation method. Limits the number of active sessions. All other sessions are inactive and wait in a queue to be activated. ACTIVE_SESS_POOL_ABSOLUTE is the default and only method available.
parallel_degree_limit_mth	Resource allocation method for specifying a limit on the degree of parallelism of any operation. PARALLEL_DEGREE_LIMIT_ABSOLUTE is the default and only method available.
queueing_mth	Queuing resource allocation method. Controls order in which queued inactive sessions will execute. FIFO_TIMEOUT is the default and only method available
mgmt_mth	Resource allocation method for specifying how much resources (for example, CPU or I/O) each consumer group or sub-plan gets <ul style="list-style-type: none"> <li>▪ EMPHASIS - for multilevel plans that use percentages to specify how I/O resources are distributed among consumer groups</li> <li>▪ RATIO - for single-level plans that use ratios to specify how I/O resources are distributed</li> </ul>
sub_plan	If TRUE, indicates that this plan is only intended for use as a sub-plan. Sub-plans are not required to have an OTHER_GROUPS directive. Default is FALSE.

### Usage Notes

If you want to use any default resource allocation method, then you do not need to specify it when creating or updating a plan.

## CREATE\_PLAN\_DIRECTIVE Procedure

This procedure creates resource plan directives.

---



---

**Note:** The functionality associated with the `parallel_target_percentage` and `parallel_queue_timeout` parameters is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
  plan                IN VARCHAR2,
  group_or_subplan    IN VARCHAR2,
  comment             IN VARCHAR2,
  cpu_p1              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p2              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p3              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p4              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p5              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p6              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p7              IN NUMBER   DEFAULT NULL, -- deprecated
  cpu_p8              IN NUMBER   DEFAULT NULL, -- deprecated
  active_sess_pool_p1 IN NUMBER   DEFAULT NULL,
  queueing_p1         IN NUMBER   DEFAULT NULL,
  parallel_degree_limit_p1 IN NUMBER DEFAULT NULL,
  switch_group        IN VARCHAR2 DEFAULT NULL,
  switch_time         IN NUMBER   DEFAULT NULL,
  switch_estimate     IN BOOLEAN  DEFAULT FALSE,
  max_est_exec_time   IN NUMBER   DEFAULT NULL,
  undo_pool           IN NUMBER   DEFAULT NULL,
  max_idle_time       IN NUMBER   DEFAULT NULL,
  max_idle_blocker_time IN NUMBER DEFAULT NULL,
  switch_time_in_call IN NUMBER   DEFAULT NULL, -- deprecated
  mgmt_p1             IN NUMBER   DEFAULT NULL,
  mgmt_p2             IN NUMBER   DEFAULT NULL,
  mgmt_p3             IN NUMBER   DEFAULT NULL,
  mgmt_p4             IN NUMBER   DEFAULT NULL,
  mgmt_p5             IN NUMBER   DEFAULT NULL,
  mgmt_p6             IN NUMBER   DEFAULT NULL,
  mgmt_p7             IN NUMBER   DEFAULT NULL,
  mgmt_p8             IN NUMBER   DEFAULT NULL,
  switch_io_megabytes IN NUMBER   DEFAULT NULL,
  switch_io_reqs      IN NUMBER   DEFAULT NULL,
  switch_for_call     IN BOOLEAN  DEFAULT NULL,
  max_utilization_limit IN NUMBER DEFAULT NULL,
  parallel_target_percentage IN NUMBER DEFAULT NULL,
  parallel_queue_timeout IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 119–7 CREATE\_PLAN\_DIRECTIVE Procedure Parameters**

Parameter	Description
<code>plan</code>	Name of the resource plan
<code>group_or_subplan</code>	Name of the consumer group or subplan
<code>comment</code>	Comment for the plan directive

**Table 119–7 (Cont.) CREATE\_PLAN\_DIRECTIVE Procedure Parameters**

Parameter	Description
cpu_p1	-- deprecated: use mgmt_p1 instead
cpu_p2	-- deprecated: use mgmt_p2 instead)
cpu_p3	-- deprecated: use mgmt_p3 instead)
cpu_p4	-- deprecated: use mgmt_p4 instead)
cpu_p5	-- deprecated: use mgmt_p5 instead)
cpu_p6	-- deprecated: use mgmt_p6 instead)
cpu_p7	-- deprecated: use mgmt_p7 instead)
cpu_p8	-- deprecated: use mgmt_p8 instead)
active_sess_pool_p1	Specifies maximum number of concurrently active sessions for a consumer group. Default is NULL, which means unlimited.
queueing_p1	Specified time (in seconds) after which a job in the inactive session queue (waiting for execution) will time out. Default is NULL, which means unlimited.
parallel_degree_limit_p1	Specifies a limit on the degree of parallelism for any operation. Default is NULL, which means unlimited.
switch_group	Specifies consumer group to switch to, once a switch condition is met. If the group name is 'CANCEL_SQL', then the current call is canceled when the switch condition is met. If the group name is 'KILL_SESSION', then the session is killed when the switch condition is met. Default is NULL.
switch_time	Specifies time (in CPU seconds) that a session can execute before an action is taken. Default is NULL, which means unlimited.
switch_estimate	If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is FALSE.
max_est_exec_time	Specifies the maximum execution time (in CPU seconds) allowed for a session. If the optimizer estimates that an operation will take longer than MAX_EST_EXEC_TIME, the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is NULL, which means unlimited.
undo_pool	Limits the size in kilobytes of the undo records corresponding to uncommitted transactions by this consumer group
max_idle_time	Indicates the maximum session idle time. Default is NULL, which means unlimited.
max_idle_blocker_time	Maximum amount of time in seconds that a session can be idle while blocking another session's acquisition of a resource
switch_time_in_call	Deprecated. If this parameter is specified, switch_time is set to switch_time_in_call (in seconds) and switch_for_call is effectively set to TRUE. It is better to use switch_time and switch_for_call.
mgmt_p1	Resource allocation value for level 1 (replaces cpu_p1): <ul style="list-style-type: none"> <li>■ EMPHASIS - specifies the resource percentage at the first level</li> <li>■ RATIO - specifies the weight of resource usage</li> </ul>



**Table 119-7 (Cont.) CREATE\_PLAN\_DIRECTIVE Procedure Parameters**

Parameter	Description
mgmt_p2	Resource allocation value for level 2 (replaces <code>cpu_p2</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the second level</li> <li>▪ RATIO - non-applicable</li> </ul>
mgmt_p3	Resource allocation value for level 3 (replaces <code>cpu_p3</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the third level</li> <li>▪ RATIO - non-applicable</li> </ul>
mgmt_p4	Resource allocation value for level 4 (replaces <code>cpu_p4</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the fourth level</li> <li>▪ RATIO - non-applicable</li> </ul>
mgmt_p5	Resource allocation value for level 5 (replaces <code>cpu_p5</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the fifth level</li> <li>▪ RATIO - non-applicable</li> </ul>
mgmt_p6	Resource allocation value for level 6 (replaces <code>cpu_p6</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the sixth level</li> <li>▪ RATIO - non-applicable</li> </ul>
mgmt_p7	Resource allocation value for level 7 (replaces <code>cpu_p7</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the seventh level</li> <li>▪ RATIO - non-applicable</li> </ul>
mgmt_p8	Resource allocation value for level 8 (replaces <code>cpu_p8</code> ) <ul style="list-style-type: none"> <li>▪ EMPHASIS - specifies the resource percentage at the eighth level</li> <li>▪ RATIO - non-applicable</li> </ul>
switch_io_megabytes	Specifies the amount of I/O (in MB) that a session can issue before an action is taken. Default is <code>NULL</code> , which means unlimited.
switch_io_reqs	Specifies the number of I/O requests that a session can issue before an action is taken. Default is <code>NULL</code> , which means unlimited.
switch_for_call	Specifies that if an action is taken because of the <code>switch_time</code> , <code>switch_io_megabytes</code> , or <code>switch_io_reqs</code> parameters, the consumer group is restored to its original consumer group at the end of the top call. Default is <code>FALSE</code> , which means that the original consumer group is not restored at the end of the top call.
max_utilization_limit	Specifies the maximum percentage of CPU that this Consumer Group or Sub-Plan can utilize. Valid values are 0% to 100%. <code>NULL</code> implies that there is no limit, or equivalently 100%. You can specify this attribute and leave <code>mgmt_p1</code> through <code>mgmt_p8</code> <code>NULL</code> .

**Table 119–7 (Cont.) CREATE\_PLAN\_DIRECTIVE Procedure Parameters**

Parameter	Description
<code>parallel_target_percentage</code>	<p>Specifies the maximum percentage of the target number of parallel servers in an Oracle RAC environment that a consumer group can use. Any additional parallel statements that are launched from this consumer group will be queued. The default is <code>NULL</code>, which means that the limit is 100% of the target number. Valid values for queuing are in the range of 0 to 100 (%). For updates to the plan directive, the value of -1 will reset the value to <code>NULL</code>.</p> <p>If a consumer group does not have any parallel statements running within an Oracle RAC database, the first parallel statement is allowed to exceed this limit.</p> <p>The target number of parallel servers in an Oracle RAC environment is the sum of the parameter <code>parallel_server_target</code> across all instances.</p>
<code>parallel_queue_timeout</code>	<p>Specifies the time (in seconds) that a query may remain in its Consumer Group's parallel statement queue before it is removed and terminated with an error (ORA-07454).</p>

### Usage Notes

- All parameters default to `NULL`. However, for the `EMPHASIS` CPU resource allocation method, this case would severely limit resources to all the users.
- For `max_idle_time` and `max_idle_blocker_time`, PMON will check these limits once a minute. If it finds a session that has exceeded one of the limits, it will forcibly kill the session and clean up all its state.
- The parameter `switch_time_in_call` is mostly useful for three-tier applications where the mid-tier server is implementing session pooling. By using `switch_time_in_call`, the resource usage of one client will not affect a future client that happens to be executed on the same session.

## CREATE\_SIMPLE\_PLAN Procedure

This procedure creates a single-level resource plan containing up to eight consumer groups in one step. You do not need to create a pending area manually before creating a resource plan, or use the CREATE\_CONSUMER\_GROUP and CREATE\_RESOURCE\_PLAN\_DIRECTIVES procedures separately.

### Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN (
  simple_plan      IN  VARCHAR2  DEFAULT NULL,
  consumer_group1  IN  VARCHAR2  DEFAULT NULL,
  group1_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group2  IN  VARCHAR2  DEFAULT NULL,
  group2_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group3  IN  VARCHAR2  DEFAULT NULL,
  group3_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group4  IN  VARCHAR2  DEFAULT NULL,
  group4_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group5  IN  VARCHAR2  DEFAULT NULL,
  group5_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group6  IN  VARCHAR2  DEFAULT NULL,
  group6_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group7  IN  VARCHAR2  DEFAULT NULL,
  group7_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  consumer_group8  IN  VARCHAR2  DEFAULT NULL,
  group8_cpu       IN  NUMBER     DEFAULT NULL,    -- deprecated
  group1_percent   IN  NUMBER     DEFAULT NULL,
  group2_percent   IN  NUMBER     DEFAULT NULL,
  group3_percent   IN  NUMBER     DEFAULT NULL,
  group4_percent   IN  NUMBER     DEFAULT NULL,
  group5_percent   IN  NUMBER     DEFAULT NULL,
  group6_percent   IN  NUMBER     DEFAULT NULL,
  group7_percent   IN  NUMBER     DEFAULT NULL,
  group8_percent   IN  NUMBER     DEFAULT NULL);
```

### Parameters

**Table 119–8 CREATE\_SIMPLE\_PLAN Procedure Parameters**

Parameter	Description
simple_plan	Name of the resource plan
consumer_group1	Name of the consumer group
group1_cpu	Percentage for group (deprecated)
consumer_group2	Name of the consumer group
group2_cpu	Percentage for group (deprecated)
consumer_group3	Name of the consumer group
group3_cpu	Percentage for group (deprecated)
consumer_group4	Name of the consumer group
group4_cpu	Percentage for group (deprecated)
consumer_group5	Name of the consumer group
group5_cpu	Percentage for group (deprecated)

**Table 119–8 (Cont.) CREATE\_SIMPLE\_PLAN Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
consumer_group6	Name of the consumer group
group6_cpu	Percentage for group (deprecated)
consumer_group7	Name of the consumer group
group7_cpu	Percentage for group (deprecated)
consumer_group8	Name of the consumer group
group8_cpu	Percentage for group (deprecated)
group1_percent	Percentage of resources allocated for this consumer group
group2_percent	Percentage of resources allocated for this consumer group
group3_percent	Percentage of resources allocated for this consumer group
group4_percent	Percentage of resources allocated for this consumer group
group5_percent	Percentage of resources allocated for this consumer group
group6_percent	Percentage of resources allocated for this consumer group
group7_percent	Percentage of resources allocated for this consumer group
group8_percent	Percentage of resources allocated for this consumer group

## DELETE\_CATEGORY Procedure

This procedure deletes an existing resource consumer group category.

### Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CATEGORY (  
    category      IN    VARCHAR2,  
    new_comment   IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 119–9** *DELETE\_CATEGORY Procedure Parameters*

Parameter	Description
category	Name of consumer group category

## DELETE\_CONSUMER\_GROUP Procedure

This procedure deletes entries which define resource consumer groups.

### Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP (  
    consumer_group IN VARCHAR2);
```

### Parameters

**Table 119–10** *DELETE\_CONSUMER\_GROUP Procedure Parameters*

Parameters	Description
consumer_group	Name of the consumer group to be deleted

## DELETE\_PLAN Procedure

This procedure deletes the specified plan as well as all the plan directives to which it refers.

### Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN (  
    plan IN VARCHAR2);
```

### Parameters

**Table 119–11** *DELETE\_PLAN Procedure Parameters*

Parameter	Description
plan	Name of the resource plan to delete

## DELETE\_PLAN\_CASCADE Procedure

This procedure deletes the specified plan and all of its descendants (plan directives, subplans, consumer groups). Mandatory objects and directives are not deleted.

### Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_CASCADE (  
    plan IN VARCHAR2);
```

### Parameters

**Table 119–12** *DELETE\_PLAN\_CASCADE Procedure Parameters*

Parameters	Description
plan	Name of the plan

### Usage Notes

If `DELETE_PLAN_CASCADE` encounters any error, then it rolls back the operation, and nothing is deleted.



## DELETE\_PLAN\_DIRECTIVE Procedure

This procedure deletes resource plan directives.

### Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE (  
    plan           IN VARCHAR2,  
    group_or_subplan IN VARCHAR2);
```

### Parameters

**Table 119–13** *DELETE\_PLAN\_DIRECTIVE Procedure Parameters*

Parameter	Description
plan	Name of the resource plan
group_or_subplan	Name of the group or subplan

## END\_SQL\_BLOCK Procedure

This procedure, to be used with parallel statement queuing, indicates the end of a block of SQL statements that should be treated as a group by resource manager.

---

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

---

### Syntax

```
DBMS_RESOURCE_MANAGER.END_SQL_BLOCK;
```

### Usage Notes

For more information, see "Parallel Statement Queuing" and "Managing Parallel Statement Queuing with Resource Manager" in *Oracle Database VLDB and Partitioning Guide*.

## SET\_CONSUMER\_GROUP\_MAPPING Procedure

This procedure adds, deletes, or modifies entries that map sessions to consumer groups, based on the session's login and runtime attributes.

### Syntax

```
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING (
  attribute      IN VARCHAR2,
  value         IN VARCHAR2,
  consumer_group IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 119-14 SET\_CONSUMER\_GROUP\_MAPPING Procedure Parameters**

Parameters	Description
attribute	Mapping attribute to add or modify. It can be one of the <a href="#">Constants</a> listed.
value	Attribute value to match. This includes both absolute mapping and regular expressions.
consumer_group	Name of the mapped consumer group, or NULL to delete a mapping

### Usage Notes

- If no mapping exists for the given attribute and value, a mapping to the given consumer group will be created. If a mapping already exists for the given attribute and value, the mapped consumer group will be updated to the one given. If the consumer\_group argument is NULL, then any mapping from the given attribute and value will be deleted.
- The subprogram supports simple regex expressions for the value parameter. It implements the same semantics as the SQL 'LIKE' operator. Specifically, it uses '%' as a multicharacter wildcard and '\_' as a single character wildcard. The '\' character can be used to escape the wildcards. Note that wildcards can only be used if the attribute is one of the following:
  - CLIENT\_OS\_USER
  - CLIENT\_PROGRAM
  - CLIENT\_MACHINE
  - MODULE\_NAME
  - MODULE\_NAME\_ACTION
  - SERVICE\_MODULE
  - SERVICE\_MODULE\_ACTION

## SET\_CONSUMER\_GROUP\_MAPPING\_PRI Procedure

Multiple attributes of a session can be used to map the session to a consumer group. This procedure prioritizes the attribute mappings.

### Syntax

```
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI (
  explicit          IN NUMBER,
  oracle_user      IN NUMBER,
  service_name     IN NUMBER,
  client_os_user   IN NUMBER,
  client_program   IN NUMBER,
  client_machine   IN NUMBER,
  module_name      IN NUMBER,
  module_name_action IN NUMBER,
  service_module   IN NUMBER,
  service_module_action IN NUMBER);
```

### Parameters

**Table 119–15 SET\_CONSUMER\_GROUP\_MAPPING\_PRI Procedure Parameters**

Parameters	Description
explicit	Priority of the explicit mapping
oracle_user	Priority of the Oracle user name mapping
service_name	Priority of the client service name mapping
client_os_user	Priority of the client operating system user name mapping
client_program	Priority of the client program mapping
client_machine	Priority of the client machine mapping
module_name	Priority of the application module name mapping
module_name_action	Priority of the application module name and action mapping
service_module	Priority of the service name and application module name mapping
module_name_action	Priority of the service name, application module name, and application action mapping

### Usage Notes

- This procedure requires that you include the pseudo-attribute `explicit` as an argument. It must be set to 1. It indicates that explicit consumer group switches have the highest priority. You explicitly switch consumer groups with these package procedures:
  - `DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP`
  - `DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS`
  - `DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER`
- Each priority value must be a unique integer from 1 to 10. Together, they establish an ordering where 1 is the highest priority and 10 is the lowest.

## SET\_INITIAL\_CONSUMER\_GROUP Procedure

---

**Note:** This procedure is deprecated in Release 11gR1. While the procedure remains available in the package, Initial Consumer Group is set by the session-to-consumer group mapping rules.

---

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. This procedure sets the initial resource consumer group for a user.

### Syntax

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
    user          IN  VARCHAR2,
    consumer_group IN  VARCHAR2);
```

### Parameters

**Table 119–16** SET\_INITIAL\_CONSUMER\_GROUP Procedure Parameters

Parameters	Description
user	Name of the user
consumer_group	User's initial consumer group

### Usage Notes

- The ADMINISTER\_RESOURCE\_MANAGER or the ALTER USER system privilege are required to be able to execute this procedure. The user, or PUBLIC, must be directly granted switch privilege to a consumer group before it can be set to be the user's initial consumer group. Switch privilege for the initial consumer group cannot come from a role granted to that user.

---

**Note:** These semantics are similar to those for ALTER USER DEFAULT ROLE.

---

- If the initial consumer group for a user has never been set, then the user's initial consumer group is automatically the consumer group: DEFAULT\_CONSUMER\_GROUP.
- DEFAULT\_CONSUMER\_GROUP has switch privileges granted to PUBLIC; therefore, all users are automatically granted switch privilege for this consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group now have DEFAULT\_CONSUMER\_GROUP as their initial consumer group. All currently active sessions belonging to a deleted consumer group are switched to DEFAULT\_CONSUMER\_GROUP.

## SUBMIT\_PENDING\_AREA Procedure

This procedure submits pending changes for the resource manager. It clears the pending area after validating and committing the changes (if valid).

---

---

**Note:** A call to `SUBMIT_PENDING_AREA` may fail even if `VALIDATE_PENDING_AREA` succeeds. This may happen if a plan being deleted is loaded by an instance after a call to `VALIDATE_PENDING_AREA`, but before a call to `SUBMIT_PENDING_AREA`.

---

---

### Syntax

```
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

## SWITCH\_CONSUMER\_GROUP\_FOR\_SESS Procedure

This procedure changes the resource consumer group of a specific session. It also changes the consumer group of any (PQ) slave sessions that are related to the top user session.

### Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS (  
    session_id      IN NUMBER,  
    session_serial  IN NUMBER,  
    consumer_group  IN VARCHAR2);
```

### Parameters

**Table 119–17 SWITCH\_CONSUMER\_GROUP\_FOR\_SESS Procedure Parameters**

Parameter	Description
session_id	SID column from the view V\$SESSION
session_serial	SERIAL# column from view V\$SESSION.
consumer_group	Name of the consumer group to which to switch

## SWITCH\_CONSUMER\_GROUP\_FOR\_USER Procedure

This procedure changes the resource consumer group for all sessions with a given user ID. It also changes the consumer group of any (PQ) slave sessions that are related to the top user session.

### Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER (  
    user          IN VARCHAR2,  
    consumer_group IN VARCHAR2);
```

### Parameters

**Table 119–18 SWITCH\_CONSUMER\_GROUP\_FOR\_USER Procedure Parameters**

Parameter	Description
user	Name of the user
consumer_group	Name of the consumer group to which to switch

### Usage Notes

- The [SWITCH\\_CONSUMER\\_GROUP\\_FOR\\_SESS Procedure](#) and the SWITCH\_CONSUMER\_GROUP\_FOR\_USER procedures let you raise or lower the allocation of CPU resources of certain sessions or users. This provides a functionality similar to the nice command on UNIX.
- These procedures cause the session to be moved into the newly specified consumer group immediately.



## SWITCH\_PLAN Procedure

This procedure sets the current resource manager plan.

### Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_PLAN(
  plan_name          IN  VARCHAR2,
  sid                IN  VARCHAR2 DEFAULT '*',
  allow_scheduler_plan_switches IN  BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 119–19 SWITCH\_PLAN Procedure Parameters**

Parameter	Description
plan_name	Name of the plan to which to switch. Passing in an empty string (") for the plan_name, disables the resource manager
sid	The sid parameter is relevant only in an Oracle Real Application Clusters environment. This parameter lets you change the plan for a particular instance. Specify the sid of the instance where you want to change the plan. Or specify '*' if you want Oracle to change the plan for all instances.
allow_scheduler_plan_switches	FALSE - disables automated plan switches by the job scheduler at window boundaries. To reenab automated plan switches, switch_plan must be called again by the administrator with allow_scheduler_plan_switches set to TRUE. By default automated plan switches by the job scheduler are enabled.

## UPDATE\_CATEGORY Procedure

This procedure updates an existing resource consumer group category.

### Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_CATEGORY (  
    category      IN    VARCHAR2,  
    new_comment   IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 119–20 UPDATE\_CATEGORY Procedure Parameters**

Parameter	Description
category	Name of consumer group category
new_comment	User's comment

## UPDATE\_CONSUMER\_GROUP Procedure

This procedure updates entries which define resource consumer groups.

### Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP (
  consumer_group IN VARCHAR2,
  new_comment    IN VARCHAR2 DEFAULT NULL,
  new_cpu_mth    IN VARCHAR2 DEFAULT NULL,
  new_mgmt_mth   IN VARCHAR2 DEFAULT NULL,
  new_category   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 119–21 UPDATE\_CONSUMER\_GROUP Procedure Parameter**

Parameter	Description
consumer_group	Name of consumer group
new_comment	New user's comment
new_cpu_mth	Name of new method for CPU resource allocation (deprecated)
new_mgmt_mth	Name of new method for CPU resource allocation
new_category	New consumer group category

### Usage Notes

If the parameters to the UPDATE\_CONSUMER\_GROUP procedure are not specified, then they remain unchanged in the data dictionary.

## UPDATE\_PLAN Procedure

This procedure updates entries which define resource plans.

### Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN (
  plan                IN VARCHAR2,
  new_comment         IN VARCHAR2 DEFAULT NULL,
  new_cpu_mth        IN VARCHAR2 DEFAULT NULL, -- deprecated
  new_active_sess_pool_mth IN VARCHAR2 DEFAULT NULL,
  new_parallel_degree_limit_mth IN VARCHAR2 DEFAULT NULL,
  new_queueing_mth   IN VARCHAR2 DEFAULT NULL,
  new_mgmt_mth       IN VARCHAR2 DEFAULT NULL,
  new_sub_plan        IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 119–22 UPDATE\_PLAN Procedure Parameters**

Parameter	Description
plan	Name of resource plan
new_comment	New user's comment
new_cpu_mth	Name of new allocation method for CPU resources (deprecated)
new_active_sess_pool_mth	Name of new method for maximum active sessions
new_parallel_degree_limit_mth	Name of new method for degree of parallelism
new_queueing_mth	Specifies type of queuing policy to use with active session pool feature
new_mgmt_mth	Resource allocation method for specifying how much resources (for example, CPU or I/O) each consumer group or sub-plan gets <ul style="list-style-type: none"> <li>▪ <b>EMPHASIS</b> - for multilevel plans that use percentages to specify how I/O resources are distributed among consumer groups.</li> <li>▪ <b>RATIO</b> - for single-level plans that use ratios to specify how I/O resources are distributed.</li> </ul>
new_sub_plan	New setting for whether the plan is only intended for use as a sub-plan

### Usage Notes

- If the parameters to [UPDATE\\_PLAN Procedure](#) are not specified, then they remain unchanged in the data dictionary.
- If you want to use any default resource allocation method, then you do not need to specify it when creating or updating a plan.

## UPDATE\_PLAN\_DIRECTIVE Procedure

This procedure updates resource plan directives.

---

**Note:** The functionality associated with the `new_parallel_target_percentage` and `new_parallel_queue_timeout` parameters is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (
  plan                               IN VARCHAR2,
  group_or_subplan                   IN VARCHAR2,
  new_comment                         IN VARCHAR2 DEFAULT NULL,
  new_cpu_p1                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p2                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p3                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p4                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p5                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p6                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p7                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_cpu_p8                         IN NUMBER   DEFAULT NULL, -- deprecated
  new_active_sess_pool_p1            IN NUMBER   DEFAULT NULL,
  new_queueing_p1                    IN NUMBER   DEFAULT NULL,
  new_parallel_degree_limit_p1       IN NUMBER   DEFAULT NULL,
  new_switch_group                   IN VARCHAR2 DEFAULT NULL,
  new_switch_time                    IN NUMBER   DEFAULT NULL,
  new_switch_estimate                IN BOOLEAN  DEFAULT FALSE,
  new_max_est_exec_time              IN NUMBER   DEFAULT NULL,
  new_undo_pool                      IN NUMBER   DEFAULT NULL,
  new_max_idle_time                  IN NUMBER   DEFAULT NULL,
  new_max_idle_blocker_time          IN NUMBER   DEFAULT NULL,
  switch_time_in_call                IN NUMBER   DEFAULT NULL, -- deprecated
  new_mgmt_p1                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p2                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p3                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p4                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p5                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p6                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p7                        IN NUMBER   DEFAULT NULL,
  new_mgmt_p8                        IN NUMBER   DEFAULT NULL,
  new_switch_io_megabytes            IN NUMBER   DEFAULT NULL,
  new_switch_io_reqs                 IN NUMBER   DEFAULT NULL,
  new_switch_for_call                IN BOOLEAN  DEFAULT NULL,
  new_max_utilization_limit          IN NUMBER   DEFAULT NULL,
  new_parallel_target_percentage     IN NUMBER   DEFAULT NULL,
  new_parallel_queue_timeout         IN NUMBER   DEFAULT NULL);
```

### Parameters

**Table 119–23** UPDATE\_PLAN\_DIRECTIVE Procedure Parameters

Parameter	Description
<code>plan</code>	Name of the resource plan
<code>group_or_subplan</code>	Name of the consumer group or subplan

**Table 119–23 (Cont.) UPDATE\_PLAN\_DIRECTIVE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>new_comment</code>	Comment for the plan directive
<code>new_cpu_p1</code>	First parameter for the CPU resources allocation method ((deprecated - use <code>new_mgmt_p1</code> instead)
<code>new_cpu_p2</code>	Parameter for the CPU resources allocation method ((deprecated - use <code>new_mgmt_p2</code> instead)
<code>new_cpu_p3</code>	Parameter for the CPU resources allocation method (deprecated - use <code>new_mgmt_p3</code> instead)
<code>new_cpu_p4</code>	Parameter for the CPU resources allocation method (deprecated- use <code>new_mgmt_p4</code> instead)
<code>new_cpu_p5</code>	Parameter for the CPU resources allocation method (deprecated - use <code>new_mgmt_p5</code> instead)
<code>new_cpu_p6</code>	Parameter for the CPU resources allocation method (deprecated- use <code>new_mgmt_p6</code> instead)
<code>new_cpu_p7</code>	Parameter for the CPU resources allocation method (deprecated- use <code>new_mgmt_p7</code> instead)
<code>new_cpu_p8</code>	Parameter for the CPU resources allocation method (deprecated- use <code>new_mgmt_p8</code> instead)
<code>new_active_sess_pool_p1</code>	Specifies maximum number of concurrently active sessions for a consumer group. Default is <code>NULL</code> , which means unlimited.
<code>new_queueing_p1</code>	Specified time (in seconds) after which a job in the inactive session queue (waiting for execution) will time out. Default is <code>NULL</code> , which means unlimited.
<code>new_parallel_degree_limit_p1</code>	Specifies a limit on the degree of parallelism for any operation. Default is <code>NULL</code> , which means unlimited.
<code>new_switch_group</code>	Specifies consumer group to which this session is switched if other switch criteria are met. Default is <code>NULL</code> . If the group name is 'CANCEL_SQL', the current call will be canceled when other switch criteria are met. If the group name is 'KILL_SESSION', the session will be killed when other switch criteria are met.
<code>new_switch_time</code>	Specifies time (in CPU seconds) that a session can execute before an action is taken. Default is <code>NULL</code> , which means unlimited.
<code>new_switch_estimate</code>	If <code>TRUE</code> , tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation before beginning its execution. Default is <code>FALSE</code> .
<code>new_max_est_exec_time</code>	Specifies the maximum execution time (in CPU seconds) allowed for a session. If the optimizer estimates that an operation will take longer than <code>MAX_EST_EXEC_TIME</code> , the operation is not started and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is <code>NULL</code> , which means unlimited.
<code>new_undo_pool</code>	Limits the size in kilobytes of the undo records corresponding to uncommitted transactions by this consumer group
<code>new_max_idle_time</code>	Indicates the maximum session idle time. Default is <code>NULL</code> , which means unlimited.
<code>new_max_idle_blocker_time</code>	Maximum amount of time in seconds that a session can be idle while blocking another session's acquisition of a resource

**Table 119–23 (Cont.) UPDATE\_PLAN\_DIRECTIVE Procedure Parameters**

Parameter	Description
<code>new_switch_time_in_call</code>	Deprecated. If this parameter is specified, <code>new_switch_time</code> will be effectively set to <code>new_switch_time_in_call</code> and <code>new_switch_for_call</code> will be effectively set to <code>TRUE</code> .
<code>new_mgmt_p1</code>	Resource allocation value for level 1 (replaces <code>new_cpu_p1</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the first level</li> <li>■ <b>RATIO</b> - specifies the weight of resource usage</li> </ul>
<code>new_mgmt_p2</code>	Resource allocation value for level 2 (replaces <code>new_cpu_p2</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the second level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_mgmt_p3</code>	Resource allocation value for level 3 (replaces <code>new_cpu_p3</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the third level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_mgmt_p4</code>	Resource allocation value for level 4 (replaces <code>new_cpu_p4</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the fourth level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_mgmt_p5</code>	Resource allocation value for level 5 (replaces <code>new_cpu_p5</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the fifth level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_mgmt_p6</code>	Resource allocation value for level 6 (replaces <code>new_cpu_p6</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the sixth level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_mgmt_p7</code>	Resource allocation value for level 7 (replaces <code>new_cpu_p7</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the seventh level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_mgmt_p8</code>	Resource allocation value for level 8 (replaces <code>new_cpu_p8</code> ): <ul style="list-style-type: none"> <li>■ <b>EMPHASIS</b> - specifies the resource percentage at the eighth level</li> <li>■ <b>RATIO</b> - non-applicable</li> </ul>
<code>new_switch_io_megabytes</code>	Specifies the amount of I/O (in MB) that a session can issue before an action is taken. Default is <code>NULL</code> , which means unlimited.
<code>new_switch_io_reqs</code>	Specifies the number of I/O requests that a session can issue before an action is taken. Default is <code>NULL</code> , which means unlimited.
<code>new_switch_for_call</code>	Specifies that if an action is taken because of the <code>new_switch_time</code> , <code>new_switch_io_megabytes</code> , or <code>new_switch_io_reqs</code> parameters, the consumer group is restored to its original consumer group at the end of the top call. Default is <code>FALSE</code> , which means that the original consumer group is not restored at the end of the top call.

**Table 119–23 (Cont.) UPDATE\_PLAN\_DIRECTIVE Procedure Parameters**

Parameter	Description
<code>new_max_utilization_limit</code>	Specifies the maximum percentage of CPU that this Consumer Group or Sub-Plan can utilize. Valid values are 0% to 100%. To unset the limit, use -1.
<code>new_parallel_target_percentage</code>	<p>Specifies the maximum percentage of the target number of parallel servers in an Oracle RAC environment a consumer group can use. Any additional parallel statements that are launched from this consumer group will be queued. The default is NULL, which means that the limit is 100% of the target number. Valid values for queuing are in the range of 0 to 100 (%). For updates to the plan directive, the value of -1 will reset the value to NULL.</p> <p>If a consumer group does not have any parallel statements running within an Oracle RAC database, the first parallel statement is allowed to exceed this limit.</p> <p>The target number of parallel servers in an Oracle RAC environment is the sum of the parameter <code>parallel_server_target</code> across all instances.</p>
<code>new_parallel_queue_timeout</code>	Specifies the time (in seconds) that a query may remain in its Consumer Group's parallel statement queue before it is removed and terminated with an error (ORA- 07454).

## Usage Notes

- If the parameters for `UPDATE_PLAN_DIRECTIVE` are left unspecified, then they remain unchanged in the data dictionary.
- For `new_max_idle_time` and `new_max_idle_blocker_time`, PMON will check these limits once a minute. If it finds a session that has exceeded one of the limits, it will forcibly kill the session and clean up all its state.
- The parameter `new_switch_time_in_call` is mostly useful for three-tier applications where the mid-tier server is implementing session pooling. By turning on `new_switch_time_in_call`, the resource usage of one client will not affect the consumer group of a future client that happens to be executed on the same session.
- To clear (zero or nullify) any numeric parameter in a resource plan directive, set it to -1 using the `UPDATE_PLAN_DIRECTIVE` Procedure.



## **VALIDATE\_PENDING\_AREA Procedure**

This procedure validates pending changes for the resource manager.

### **Syntax**

```
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```



---

---

## DBMS\_RESOURCE\_MANAGER\_PRIVS

The DBMS\_RESOURCE\_MANAGER\_PRIVS package maintains privileges associated with the Resource Manager.

**See Also:** For more information on using the Database Resource Manager, see *Oracle Database Administrator's Guide*.

This chapter contains the following topics:

- [Summary of DBMS\\_RESOURCE\\_MANAGER\\_PRIVS Subprograms](#)

---

## Summary of DBMS\_RESOURCE\_MANAGER\_PRIVS Subprograms

**Table 120–1** *DBMS\_RESOURCE\_MANAGER\_PRIVS Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">GRANT_SWITCH_CONSUMER_GROUP</a> Procedure on page 120-3	Grants the privilege to switch to resource consumer groups
<a href="#">GRANT_SYSTEM_PRIVILEGE</a> Procedure on page 120-4	Performs a grant of a system privilege
<a href="#">REVOKE_SWITCH_CONSUMER_GROUP</a> Procedure on page 120-5	Revokes the privilege to switch to resource consumer groups.
<a href="#">REVOKE_SYSTEM_PRIVILEGE</a> Procedure on page 120-6	Performs a revoke of a system privilege

## GRANT\_SWITCH\_CONSUMER\_GROUP Procedure

This procedure grants the privilege to switch to a resource consumer group.

### Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
  grantee_name  IN VARCHAR2,
  consumer_group IN VARCHAR2,
  grant_option  IN BOOLEAN);
```

### Parameters

**Table 120–2 GRANT\_SWITCH\_CONSUMER\_GROUP Procedure Parameters**

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
consumer_group	Name of consumer group.
grant_option	TRUE if grantee should be allowed to grant access, FALSE otherwise.

### Usage Notes

If you grant permission to switch to a particular consumer group to a user, then that user can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to a role, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to PUBLIC, then any user can switch to that consumer group.

If the `grant_option` parameter is TRUE, then users granted switch privilege for the consumer group may also grant switch privileges for that consumer group to others.

In order to set the initial consumer group of a user, you must grant the switch privilege for that group to the user.

**See Also:** [Chapter 119, "DBMS\\_RESOURCE\\_MANAGER"](#)

### Examples

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
  'scott', 'mail_maintenance_group', true);
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.set_consumer_group_mapping(
  dbms_resource_manager.oracle_user, 'scott','mail_maintenance_group');
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

## GRANT\_SYSTEM\_PRIVILEGE Procedure

This procedure performs a grant of a system privilege to a user or role.

### Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
    grantee_name    IN VARCHAR2,
    privilege_name  IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER',
    admin_option    IN BOOLEAN);
```

### Parameters

**Table 120–3 GRANT\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
privilege_name	Name of the privilege to be granted.
admin_option	TRUE if the grant is with <code>admin_option</code> , FALSE otherwise.

### Usage Notes

Currently, Oracle provides only one system privilege for the Resource Manager: `ADMINISTER_RESOURCE_MANAGER`. Database administrators have this system privilege with the `ADMIN` option. The grantee and the revokee can either be a user or a role. Users that have been granted the system privilege with the `ADMIN` option can also grant this privilege to others.

### Examples

The following call grants this privilege to a user called `scott` without the `ADMIN` option:

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
    grantee_name => 'scott',
    privilege_name => 'ADMINISTER_RESOURCE_MANAGER',
    admin_option => FALSE);
END;
/
```

## REVOKE\_SWITCH\_CONSUMER\_GROUP Procedure

This procedure revokes the privilege to switch to a resource consumer group.

### Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
    revokee_name  IN VARCHAR2,
    consumer_group IN VARCHAR2);
```

### Parameters

**Table 120–4 REVOKE\_SWITCH\_CONSUMER\_GROUP Procedure Parameter**

Parameter	Description
revokee_name	Name of user/role from which to revoke access.
consumer_group	Name of consumer group.

### Usage Notes

If you revoke a user's switch privilege for a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail.

If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT\_CONSUMER\_GROUP consumer group when logging in.

If you revoke the switch privilege for a consumer group from a role, then any users who only had switch privilege for the consumer group through that role will not be able to switch to that consumer group.

If you revoke the switch privilege for a consumer group from PUBLIC, then any users who could previously only use the consumer group through PUBLIC will not be able to switch to that consumer group.

### Examples

The following example revokes the privileges to switch to mail\_maintenance\_group from Scott:

```
BEGIN
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
    'scott', 'mail_maintenance_group');
END;
/
```

## REVOKE\_SYSTEM\_PRIVILEGE Procedure

This procedure performs a revoke of a system privilege from a user or role.

### Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE (  
    revokee_name    IN VARCHAR2,  
    privilege_name  IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER');
```

### Parameters

**Table 120–5 REVOKE\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
revokee_name	Name of the user or role from whom privilege is to be revoked.
privilege_name	Name of the privilege to be revoked.

### Examples

The following call revokes the ADMINISTER\_RESOURCE\_MANAGER from user scott:

```
BEGIN  
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE ('scott');  
END;  
/
```



---

---

## DBMS\_RESULT\_CACHE

The `DBMS_RESULT_CACHE` package provides an interface to allow the DBA to administer that part of the shared pool that is used by the SQL result cache and the PL/SQL function result cache. Both these caches use the same infrastructure. Therefore, for example, `DBMS_RESULT_CACHE.BYPASS` determines whether both caches are bypassed or both caches are used, and `DBMS_RESULT_CACHE.FLUSH` flushes both all the cached results for SQL queries and all the cached results for PL/SQL functions.

**See Also:**

- *Oracle Database PL/SQL Language Reference* for more information about "Using the Cross-Session PL/SQL Function Result Cache"
- *Oracle Database Performance Tuning Guide* for more information about "Result Cache Concepts"

This chapter contains the following topics:

- [Using DBMS\\_RESULT\\_CACHE](#)
  - Constants
- [Summary of DBMS\\_RESULT\\_CACHE Subprograms](#)

## Using DBMS\_RESULT\_CACHE

---

- [Constants](#)

## Constants

**Table 121-1 DBMS\_RESULT\_CACHE Constants**

<b>Constant</b>	<b>Definition</b>
STATUS_BYPS	CONSTANT VARCHAR(10) := 'BYPASS';
STATUS_CORR	CONSTANT VARCHAR(10) := 'CORRUPT';
STATUS_DISA	CONSTANT VARCHAR(10) := 'DISABLED';
STATUS_ENAB	CONSTANT VARCHAR(10) := 'ENABLED';
STATUS_SYNC	CONSTANT VARCHAR(10) := 'SYNC';

## Summary of DBMS\_RESULT\_CACHE Subprograms

**Table 121–2 DBMS\_RESULT\_CACHE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">BYPASS Procedure</a> on page 121-5	Sets the bypass mode for the Result Cache
<a href="#">FLUSH Function &amp; Procedure</a> on page 121-7	Attempts to remove all the objects from the Result Cache, and depending on the arguments retains or releases the memory and retains or clears the statistics
<a href="#">INVALIDATE Functions &amp; Procedures</a> on page 121-8	Invalidates all the result-set objects that dependent upon the specified dependency object
<a href="#">INVALIDATE_OBJECT Functions &amp; Procedures</a> on page 121-9	Invalidates the specified result-set object(s)
<a href="#">MEMORY_REPORT Procedure</a> on page 121-10	Produces the memory usage report for the Result Cache
<a href="#">STATUS Function</a> on page 121-11	Checks the status of the Result Cache

## BYPASS Procedure

This procedure sets the bypass mode for the Result Cache:

- When bypass mode is turned on, it implies that cached results are no longer used and that no new results are saved in the cache.
- When bypass mode is turned off, the cache resumes normal operation.

### Syntax

```
DBMS_RESULT_CACHE.BYPASS (
    bypass_mode    IN    BOOLEAN,
    session        IN    BOOLEAN);
```

### Parameters

**Table 121–3** *BYPASS Procedure Parameters*

Parameter	Description
bypass_mode	<ul style="list-style-type: none"> <li>■ TRUE =&gt; Result Cache usage is bypassed</li> <li>■ FALSE =&gt; Result Cache usage is turned on</li> </ul>
session	<ul style="list-style-type: none"> <li>■ TRUE =&gt; Applies to current session</li> <li>■ FALSE (default) =&gt; Applies to all sessions</li> </ul>

### Usage Notes

This operation is database instance specific.

### Examples

This operation can be used when there is a need to hot patch PL/SQL code in a running system. If a code-patch is applied to a PL/SQL module on which a result cached function directly or transitively depends, then the cached results associated with the result cache function are not automatically flushed (if the instance is not restarted/bounced). This must be manually achieved.

To ensure correctness during the patching process follow these steps:

1. Place the result cache in bypass mode, and flush existing result.

```
BEGIN
    DBMS_RESULT_CACHE.BYPASS(TRUE);
    DBMS_RESULT_CACHE.FLUSH;
END;
/
```

This step must be performed on each instance if in a Oracle Real Application Clusters environment.

2. Apply the PL/SQL code patches.
3. Resume use of the result cache, by turning off the cache bypass mode.

```
BEGIN
    DBMS_RESULT_CACHE.BYPASS(FALSE);
END;
/
```

This step must be performed on each instance if in a Oracle Real Application Clusters environment.

## FLUSH Function & Procedure

This function and procedure attempts to remove all the objects from the Result Cache, and depending on the arguments retains or releases the memory and retains or clears the statistics.

### Syntax

```
DBMS_RESULT_CACHE.FLUSH (
  retainMem IN BOOLEAN DEFAULT FALSE,
  retainSta IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBMS_RESULT_CACHE.FLUSH (
  retainMem IN BOOLEAN DEFAULT FALSE,
  retainSta IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 121–4 FLUSH Function & Procedure Parameters**

Parameter	Description
retainMem	<ul style="list-style-type: none"> <li>▪ TRUE =&gt; retains the free memory in the cache</li> <li>▪ FALSE (default) =&gt; releases the free memory to the system</li> </ul>
retainSta	<ul style="list-style-type: none"> <li>▪ TRUE =&gt; retains the existing cache statistics</li> <li>▪ FALSE (default) =&gt; clears the existing cache statistics</li> </ul>

### Return Values

TRUE if successful in removing all the objects.

## INVALIDATE Functions & Procedures

This function and procedure invalidates all the result-set objects that dependent upon the specified dependency object.

### Syntax

```
DBMS_RESULT_CACHE.INVALIDATE (  
  owner      IN VARCHAR2,  
  name       IN VARCHAR2)  
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE (  
  owner      IN VARCHAR2,  
  name       IN VARCHAR2);
```

```
DBMS_RESULT_CACHE.INVALIDATE (  
  object_id  IN BINARY_INTEGER)  
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE (  
  object_id  IN BINARY_INTEGER);
```

### Parameters

**Table 121–5** *INVALIDATE Function & Procedure Parameters*

Parameter	Description
owner	Schema name
name	Object name
object_id	Dictionary object number

### Return Values

The number of objects invalidated.



## INVALIDATE\_OBJECT Functions & Procedures

This function and procedure invalidates the specified result-set object(s).

### Syntax

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT (
  id          IN BINARY_INTEGER)
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT (
  id          IN BINARY_INTEGER);
```

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT (
  cache_id    IN VARCHAR2)
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT (
  cache_id    IN VARCHAR2);
```

### Parameters

**Table 121–6** *INVALIDATE Function & Procedure Parameters*

Parameter	Description
id	Address of the cache object in the Result Cache
cache_id	Cache-id

### Return Values

The number of objects invalidated.

## MEMORY\_REPORT Procedure

This procedure produces the memory usage report for the Result Cache.

### Syntax

```
DBMS_RESULT_CACHE.MEMORY_REPORT (  
    detailed IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 121–7 MEMORY\_REPORT Procedure Parameters**

Parameter	Description
detailed	<ul style="list-style-type: none"><li>TRUE =&gt; produces a more detailed report</li><li>FALSE (default) =&gt; produces the standard report</li></ul>

### Usage Notes

Invoking this procedure from SQL\*Plus requires that the serveroutput be turned on.

## STATUS Function

This function checks the status of the Result Cache.

### Syntax

```
DBMS_RESULT_CACHE.STATUS  
RETURN VARCHAR2;
```

### Return Values

One of the following values:

- STATUS\_DISA - Cache is not available
- STATUS\_ENAB - Cache is available
- STATUS\_BYPS: Cache has been made temporarilyunavailable.
- STATUS\_SYNC - Cache is available, but synchronizing with Oracle RAC nodes



---

---

## DBMS\_RESUMABLE

With the `DBMS_RESUMABLE` package, you can suspend large operations that run out of space or reach space limits after executing for a long time, fix the problem, and make the statement resume execution. In this way you can write applications without worrying about running into space-related errors.

This chapter contains the following topics:

- [Using DBMS\\_RESUMABLE](#)
  - Operational Notes
- [Summary of DBMS\\_RESUMABLE Subprograms](#)

## Using DBMS\_RESUMABLE

- [Operational Notes](#)

## Operational Notes

When you suspend a statement, you should log the suspension in the alert log. You should also register a procedure to be executed when the statement is suspended. Using a view, you can monitor the progress of the statement and indicate whether the statement is currently executing or suspended.

Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held during a statement suspend and resume. When the error condition disappears, the suspended statement automatically resumes execution. A resumable space allocation can be suspended and resumed multiple times during execution.

A suspension timeout interval is associated with resumable space allocations. A resumable space allocation that is suspended for the timeout interval (the default is two hours) wakes up and returns an exception to the user. A suspended statement may be forced to throw an exception using the `DBMS_RESUMABLE.ABORT()` procedure.

## Summary of DBMS\_RESUMABLE Subprograms

**Table 122–1 DBMS\_RESUMABLE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ABORT Procedure</a> on page 122-5	Aborts a suspended resumable space allocation
<a href="#">GET_SESSION_TIMEOUT Function</a> on page 122-6	Returns the current timeout value of the resumable space allocations for a session with <code>session_id</code>
<a href="#">GET_TIMEOUT Function</a> on page 122-7	Returns the current timeout value of resumable space allocations for the current session
<a href="#">SET_SESSION_TIMEOUT Procedure</a> on page 122-8	Sets the timeout of resumable space allocations for a session with <code>session_id</code>
<a href="#">SET_TIMEOUT Procedure</a> on page 122-9	Sets the timeout of resumable space allocations for the current session
<a href="#">SPACE_ERROR_INFO Function</a> on page 122-10	Looks for space-related errors in the error stack, otherwise returning <code>FALSE</code>



## ABORT Procedure

This procedure aborts a suspended resumable space allocation. The parameter `session_id` is the session ID in which the statement is executed. For a parallel DML/DDL, `session_id` is any session ID that participates in the parallel DML/DDL. This operation is guaranteed to succeed. The procedure can be called either inside or outside of the `AFTER SUSPEND` trigger.

### Syntax

```
DBMS_RESUMABLE.ABORT (  
    session_id IN NUMBER);
```

### Parameters

**Table 122–2** *ABORT Procedure Parameters*

Parameter	Description
<code>session_id</code>	The session identifier of the resumable space allocation.

### Usage Notes

To call an `ABORT` procedure, you must be the owner of the session with `session_id`, have `ALTER SYSTEM` privileges, or be a DBA.

## GET\_SESSION\_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for a session with `session_id`.

### Syntax

```
DBMS_RESUMABLE.GET_SESSION_TIMEOUT (  
    session_id IN NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 122–3** *GET\_SESSION\_TIMEOUT Function Parameters*

Parameter	Description
<code>session_id</code>	The session identifier of the resumable space allocation.

### Return Values

**Table 122–4** *GET\_SESSION\_TIMEOUT Function Return Values*

Return Value	Description
NUMBER	The current timeout value of resumable space allocations for a session with <code>session_id</code> . The timeout is returned in seconds.

### Usage Notes

If `session_id` does not exist, the `GET_SESSION_TIMEOUT` function returns -1.

## GET\_TIMEOUT Function

This function returns the current timeout value of resumable space allocations for the current session.

### Syntax

```
DBMS_RESUMABLE.GET_TIMEOUT  
RETURN NUMBER;
```

### Return Values

**Table 122-5** *GET\_TIMEOUT Function Return Values*

Return Value	Description
NUMBER	The current timeout value of resumable space allocations for the current session. The returned value is in seconds.

### Usage Notes

If the current session is not resumable enabled, the GET\_TIMEOUT function returns -1.

## SET\_SESSION\_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for a session with `session_id`. The new timeout setting applies to the session immediately. If `session_id` does not exist, no operation occurs.

### Syntax

```
DBMS_RESUMABLE.SET_SESSION_TIMEOUT (  
    session_id IN NUMBER,  
    timeout    IN NUMBER);
```

### Parameters

**Table 122–6** *SET\_SESSION\_TIMEOUT Procedure Parameters*

Parameter	Description
<code>session_id</code>	The session identifier of the resumable space allocation.
<code>timeout</code>	The timeout of the resumable space allocation.

## SET\_TIMEOUT Procedure

This procedure sets the timeout of resumable space allocations for the current session. The new timeout setting applies to the session immediately.

### Syntax

```
DBMS_RESUMABLE.SET_TIMEOUT (  
    timeout IN NUMBER);
```

### Parameters

**Table 122–7** *SET\_TIMEOUT Procedure Parameters*

Parameter	Description
timeout	The timeout of the resumable space allocation.

## SPACE\_ERROR\_INFO Function

This function looks for space-related errors in the error stack. If it cannot find a space related error, it will return `FALSE`. Otherwise, `TRUE` is returned and information about the particular object that causes the space error is returned.

### Syntax

```
DBMS_RESUMABLE.SPACE_ERROR_INFO
  error_type      OUT VARCHAR2,
  object_type     OUT VARCHAR2,
  object_owner    OUT VARCHAR2,
  table_space_name OUT VARCHAR2,
  object_name     OUT VARCHAR2,
  sub_object_name OUT VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

**Table 122–8** SPACE\_ERROR\_INFO Function Parameters

Parameter	Description
error_type	The space error type. It will be one of the following: <ul style="list-style-type: none"> <li>▪ NO MORE SPACE</li> <li>▪ MAX EXTENTS REACHED</li> <li>▪ SPACE QUOTA EXCEEDED</li> </ul>
object_type	The object type. It will be one of the following: <ul style="list-style-type: none"> <li>▪ TABLE</li> <li>▪ INDEX</li> <li>▪ CLUSTER</li> <li>▪ TABLE SPACE</li> <li>▪ ROLLBACK SEGMENT</li> <li>▪ UNDO SEGMENT</li> <li>▪ LOB SEGMENT</li> <li>▪ TEMP SEGMENT</li> <li>▪ INDEX PARTITION</li> <li>▪ TABLE PARTITION</li> <li>▪ LOB PARTITION</li> <li>▪ TABLE SUBPARTITION</li> <li>▪ INDEX SUBPARTITION</li> <li>▪ LOB SUBPARTITION</li> </ul> <p>The type can also be <code>NULL</code> if it does not apply.</p>
object_owner	The owner of the object. <code>NULL</code> if it cannot be determined.
table_space_name	The table space where the object resides. <code>NULL</code> if it cannot be determined.
object_name	The name of rollback segment, temp segment, table, index, or cluster.
sub_object_name	The partition name or sub-partition name of <code>LOB</code> , <code>TABLE</code> , or <code>INDEX</code> . <code>NULL</code> if it cannot be determined.







---

---

**Note:** This functionality is deprecated with Oracle Database Release 11.2 and obsoleted with Release 12.1. For details regarding obsolescence, see My Oracle Support Note ID 1244535.1

---

---

The DBMS\_RLMGR package contains various procedures to create and manage rules and rule sessions by the Rules Manager.

**See Also:** *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information.

This chapter contains the following topic:

- [Using DBMS\\_RLMGR](#)
- [Summary of Rules Manager Subprograms](#)

---

## Using DBMS\_RLMGR

This section contains topics that relate to using the Rules Manager DBMS\_RLMGR package.

- [Security Model](#)

## Security Model

The Oracle Database installation runs the `catrul.sql` script to load the DBMS\_RLMGR package and create the required Rules Manager schema objects in the EXFSYS Schema.

DBMS\_RLMGR is an EXFSYS-owned package compiled with `AUTHID CURRENT_USER`. Any DBMS\_RLMGR subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

A user must be granted `CONNECT` and `RESOURCE` roles, `EXECUTE` privilege on `DBMS_LOCK`, and `CREATE VIEW` privilege to use this package.

For successful creation of a rule class, you must have sufficient privileges to create views, object types, tables, packages, and procedures.

The owner of the rule class always has privileges to drop a rule class, process rules in a rule class, add rules and delete rules from a rules class. Only the owner of the rule class can drop a rule class and this privilege cannot be granted to another user. Rule class privileges cannot be revoked from the owner of the rule class.

A user who is not the owner of the rule class must be granted appropriate types of privileges to perform certain tasks. The types of privileges that can be granted are:

- **PROCESS RULES:** A user with `PROCESS RULES` privilege on a rule class can process the rules in the rule class using the `PROCESS_RULES` procedure or the `ADD_EVENT` procedure. Also, the user with this privilege can select from the corresponding rule class results view.
- **ADD RULE:** A user with `ADD RULE` privilege on a rule class can add rules to a rule class. Alternatively, the owner of the rule class can grant the `INSERT` privilege on one rule class table to other users.
- **DELETE RULE:** A user with `DELETE RULE` privilege on a rule class can delete rules from a rule class. Alternatively, the owner of the rule class can grant the `DELETE` privilege on one rule class table to other users.
- **ALL:** Granting the `ALL` privilege on a rule class is equivalent to granting all the above privileges on the rule class to the user.

A user must have the `EXECUTE` privilege on the primitive event types associated with a rule class before that user can make use of the corresponding rule class results view.

The owner of the rule class can add the rules using `SQL INSERT` statement on the rule class table (that shares the same name as the rule class). Note that the owner of the rule class can also grant direct DML privileges on the rule class table to other users. When you use the schema extended name for the rule class, the user must have the `ADD RULE` privilege on the rule class to add a rule to the rule class.

The owner of the rule class can use an `SQL DELETE` statement on one rule class table to delete a rule. When you use the schema extended name for the rule class, the user must have the `DELETE RULE` privilege on the rule class.

When the schema extended name is used for the rule class, the user must have `PROCESS RULES` privilege on the rule class.

A user must have `EXECUTE` privilege on the `CTX_DDL` package for successful synchronization of the text indexes using the `DBMS_RLMGR.SYNC_TEXT_INDEXES` procedure.

The `USER_RLMGR_PRIVILEGES` view lists privileges of the current user for the rule classes.

## Summary of Rules Manager Subprograms

Table 123–1 describes the subprograms in the DBMS\_RLMGR package.

All the values and names passed to the procedures defined in the DBMS\_RLMGR package are case insensitive unless otherwise mentioned. To preserve the case, enclose the values with double quotation marks.

**Table 123–1 DBMS\_RLMGR Package Subprograms**

Subprogram	Description
ADD_ELEMENTARY_ATTRIBUTE Procedures	Adds the specified attribute to the event structure and the Expression Filter attribute set
ADD_EVENT Procedure	Adds an event to a rule class in an active session
ADD_FUNCTIONS Procedure	Adds a Function, a Type, or a Package to the approved list of functions with an event structure and to the Expression Filter attribute set
ADD_RULE Procedure	Adds a rule to the rule class
CONDITION_REF Function	Retrieves the primitive rule condition reference from a rule condition for composite events
CONSUME_EVENT Function	Consumes an event using its identifiers and prepares the corresponding rule for action execution
CONSUME_PRIM_EVENTS Function	Consumes one or more primitive events with all or none semantics
CREATE_CONDITIONS_TABLE Procedure	Creates a repository for the primitive rule conditions that can be shared by multiple rules from the same or different rule classes
CREATE_EVENT_STRUCT Procedure	Creates an event structure
CREATE_EXPFIL_INDEXES Procedure	Creates expression filter indexes for the rule class if the default indexes have been dropped
CREATE_INTERFACE Procedure	Creates a rule class interface package to directly operate on the rule class
CREATE_RULE_CLASS Procedure	Creates a rule class
DELETE_RULE Procedure	Deletes a rule from a rule class
DROP_CONDITIONS_TABLE Procedure	Drops the conditions table
DROP_EVENT_STRUCT Procedure	Drops an event structure
DROP_EXPFIL_INDEXES Procedure	Drops Expression Filter indexes for the rule conditions
DROP_INTERFACE Procedure	Drops the rule class interface package
DROP_RULE_CLASS Procedure	Drops a rule class
EXTEND_EVENT_STRUCT Procedure	Adds an attribute to the primitive event structure
GET_AGGREGATE_VALUE Function	Retrieves the aggregate value computed for a collection event
GRANT_PRIVILEGE Procedure	Grants a privilege on a rule class to another user
PROCESS_RULES Procedure	Process the rules for a given event

**Table 123-1 (Cont.) DBMS\_RLMGR Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">PURGE_EVENTS Procedure</a>	Resets the rule class by removing all the events associated with the rule class and purging any state information pertaining to rules matching some events
<a href="#">RESET_SESSION Procedure</a>	Starts a new rule session within a database session
<a href="#">REVOKE_PRIVILEGE Procedure</a>	Revokes a privilege on a rule class from a user
<a href="#">SYNC_TEXT_INDEXES Procedure</a>	Synchronizes the indexes defined to process the predicates involving the CONTAINS operator in rule conditions

## ADD\_ELEMENTARY\_ATTRIBUTE Procedures

This procedure adds the specified attribute to an event structure, which is also the Expression Filter attribute set. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Adds the specified elementary attribute to the attribute set:

```
DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    event_struct IN VARCHAR2,
    attr_name    IN VARCHAR2,
    attr_type    IN VARCHAR2,
    attr_defv1   IN VARCHAR2 default NULL);
```

Identifies the elementary attributes that are table aliases and adds them to the event structure:

```
DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    event_struct IN VARCHAR2,
    attr_name    IN VARCHAR2,
    tab_alias    IN rlm$table_alias);
```

Allows addition of text attributes to the attribute set:

```
DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    event_struct IN VARCHAR2,
    attr_name    IN VARCHAR2,
    attr_type    IN VARCHAR2,
    text_pref    IN EXP$TEXT);
```

### Parameters

**Table 123–2 ADD\_ELEMENTARY\_ATTRIBUTE Procedure Parameters**

Parameter	Description
event_struct	Name of the event structure or attribute set to which this attribute is added
attr_name	Name of the elementary attribute to be added. No two attributes in a set can have the same name.
attr_type	Datatype of the attribute. This argument accepts any standard SQL datatype or the name of an object type that is accessible to the current user.
tab_alias	The type that identifies the database table to which the attribute is aliased
attr_defv1	Default value for the elementary attribute
text_pref	Text preferences such as LEXER and WORDLIST specification

### Usage Notes

- This procedure adds an elementary attribute to an event structure. The event structure is internally managed as the Expression Filter attribute set. If the event structure was originally created from an existing object type, then additional attributes cannot be added.

Elementary attributes cannot be added to an attribute set that is already assigned to a column storing expressions, which is equivalent to an event structure that is used for a rule class.

- One or more, or all elementary attributes in an attribute set can be table aliases. If an elementary attribute is a table alias, then the value assigned to the elementary attribute is a ROWID from the corresponding table. An attribute set with one or more table alias attributes cannot be created from an existing object type. For more information about table aliases, see *Oracle Database Rules Manager and Expression Filter Developer's Guide*.
- You cannot add elementary attributes to an attribute set that is already assigned to a column storing expressions.
- See the section on defining attribute sets in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about adding elementary attributes.
- Related views: USER\_EXPFIL\_ATTRIBUTE\_SETS and USER\_EXPFIL\_ATTRIBUTES.
- This procedure with a text preference bound to the text\_pref argument creates a text attribute in the attribute set. The data type for such an attribute should be a VARCHAR2 or a CLOB. The preferences specified for a text attribute are used to process the predicates involving CONTAINS operator on the attributes. The valid preferences are those that are valid in the PARAMETERS clause of CTXRULE index creation. See *Oracle Text Application Developer's Guide* for the syntax.

## Examples

The following command adds two elementary attributes to an attribute set:

```
BEGIN
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    EVENT_STRUCT => 'HRAttrSet',
    ATTR_NAME => 'HRREP',
    attr_type => 'VARCHAR2');
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    EVENT_STRUCT => 'HRAttrSet',
    ATTR_NAME => 'DEPT',
    TAB_ALIAS => RLM$TABLE_ALIAS('DEPT'));
END;
```

The following commands create an attribute set with each hotel reservation including some additional information, described as the Add1Info attribute of CLOB data type. Rule conditions specified for this event structure can include text predicates on this attribute.

```
BEGIN
  DBMS_RLMGR.CREATE_EVENT_STRUCT (EVENT_STRUCT => 'AddFlight');
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    EVENT_STRUCT => 'AddHotel',
    ATTR_NAME => 'CustId',
    ATTR_TYPE => 'NUMBER');
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    EVENT_STRUCT => 'AddHotel',
    ATTR_NAME => 'Type',
    ATTR_TYPE => 'VARCHAR2(20)');
  . . .
  DBMS_RLMGR.ADD_ELEMENTARY_ATTRIBUTE (
    EVENT_STRUCT => 'AddHotel',
```

```
ATTR_NAME => 'AddlInfo',  
ATTR_TYPE => 'CLOB',  
TEXT_PREF => EXF$TEXT('LEXER hotelreserv_lexer');  
  
END;
```



## ADD\_EVENT Procedure

This procedure adds a primitive event to a rule class in an active rule session. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Adds a string representation of the primitive event instance to a rule class:

```
DBMS_RLMGR.ADD_EVENT (
    rule_class      IN VARCHAR2,
    event_inst      IN VARCHAR2,
    event_type      IN VARCHAR2 default null);
```

Adds an AnyData representation of the primitive event instance to a rule class:

```
DBMS_RLMGR.ADD_EVENT (
    rule_class      IN VARCHAR2,
    event_inst      IN sys.AnyData);
```

### Parameters

**Table 123–3 ADD\_EVENT Procedure Parameters**

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
event_inst	String or AnyData representation of the event instance being added to the rule class
event_type	Type of event instance assigned to the event_inst argument when the string representation of the event instance is used for a rule class configured for composite events

### Usage Notes

- This procedure is used to add a primitive or a simple event to a rule class within an active rule session. By default, a rule session is the same as the database session. Optionally, multiple (sequential) rule sessions can be started within a database session by using the `RESET_SESSION` or `PROCESS_RULES` procedures.
- When the rule class is configured for simple events (consisting of only one primitive event structure), the `event_type` argument for the `ADD_EVENT` procedure can be ignored. Also, when the AnyData format of the event instance is passed, the event type information is embedded in the AnyData instance. In all other cases, the name of the primitive event structure being added to the rule class should be assigned to the `event_type` argument.
- For a valid event instance, the `ADD_EVENT` procedure processes the rules in the rule class and captures the results in the rule class results view (configured at the time of rule class creation). These results are preserved until the end of the rule session.
- When schema extended name is used for the rule class, you should have `PROCESS_RULES` privilege on the rule class. See the [GRANT\\_PRIVILEGE Procedure](#) for additional information. The value specified for the `event_type` argument is always resolved in the rule class owner's schema and should not use schema extended names. When a composite event structure is configured with a table alias

primitive event type, the name of the corresponding table should be assigned to the `event_type` argument.

## Examples

The following commands add two events to the `CompTravelPromo` rule class that is configured for two types of primitive events (`AddFlight` and `AddRentalCar`).

```
BEGIN
  DBMS_RLMGR.ADD_EVENT(rule_class => 'CompTravelPromo',
                      event_inst =>
                        AddFlight.getVarchar(987, 'Abcair', 'Boston',
                                             'Orlando', '01-APR-2003', '08-APR-2003'),
                      event_type => 'AddFlight');

  DBMS_RLMGR.ADD_EVENT(rule_class => 'Scott.CompTravelPromo',
                      event_inst =>
                        AnyData.convertObject(
                          AddRentalCar(987, 'Luxury', '03-APR-2003',
                                       '08-APR-2003', NULL)));

END;
/
```

## ADD\_FUNCTIONS Procedure

This procedure adds a user-defined function, package, or type representing a set of functions to the event structure, which is also the Expression Filter attribute set.

### Syntax

```
DBMS_RLMGR.ADD_FUNCTIONS (
    event_struct IN VARCHAR2,
    funcs_name  IN VARCHAR2);
```

### Parameters

**Table 123–4 ADD\_FUNCTIONS Procedure Parameters**

Parameter	Description
event_struct	Name of the event structure to which the functions are added
funcs_name	Name of a function, package, or type (representing a function set) or its synonyms

### Usage Notes

- By default, an attribute set implicitly allows references to all Oracle Database-supplied SQL functions for use in the rule conditions. If the expression set refers to a user-defined function, the expression set must be explicitly added to the attribute set.
- The ADD\_FUNCTIONS procedure adds a user-defined function or a package (or type) representing a set of functions to the attribute set. Any new or modified expressions are validated using this list.
- You can specify the function or the package name with a schema extension. If you specify a function name without a schema extension, only such references in the rule condition are considered valid. You can restrict the conditional expression to use a synonym to a function or a package by adding the corresponding synonym to the attribute set. This preserves the portability of the expression set to other schemas.
- See the section on defining attribute sets in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information about adding functions to an attribute set.
- Related views: USER\_EXPFIL\_ATTRIBUTE\_SETS and USER\_EXPFIL\_ASET\_FUNCTIONS

### Examples

The following command adds two functions to the attribute set:

```
BEGIN
    DBMS_RLMGR.ADD_FUNCTIONS (attr_set => 'Car4Sale',
                             funcs_name => 'HorsePower');
    DBMS_RLMGR.ADD_FUNCTIONS (attr_set => 'Car4Sale',
                             funcs_name => 'Scott.CrashTestRating');
END;
/
```

## ADD\_RULE Procedure

This procedure adds new rules to a rule class.

### Syntax

```
DBMS_RLMGR.ADD_RULE (
    rule_class      IN  VARCHAR2,
    rule_id         IN  VARCHAR2,
    rule_cond       IN  VARCHAR2,
    actprf_nml     IN  VARCHAR2 DEFAULT NULL,
    actprf_vall    IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 123–5 ADD\_RULE Procedure Parameters**

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
rule_id	Unique identifier for the rule within the rule class
rule_cond	The condition for the rule. The condition uses the variables defined in the rule class's event structure.
actprf_nml	The list of action preference names for which values will be assigned through the actprf_vall argument
actprf_vall	The list of action preference values for the names list assigned to the actprf_nml argument

### Usage Notes

- This procedure is used to add new rules to the rule class. The rule condition passed to the ADD\_RULE procedure is validated using the event structure associated with the rule class. The action preferences names list is a subset of action preference categories configured during rule class creation.
- When schema extended name is used for the rule class, you should have ADD RULE privilege on the rule class. See the [GRANT\\_PRIVILEGE Procedure](#) for more information.
- Alternately, the owner of the rule class can add the rules using SQL INSERT statement on the rule class table (that shares the same name as the rule class). Note that the owner of the rule class can also grant direct DML privileges on the rule class table to other users.

---

**Note:** The AUTOCOMMIT property of the rule class is ignored if the new rules are added using the SQL INSERT statement instead of the ADD\_RULE procedure.

---

- See the [CREATE\\_RULE\\_CLASS Procedure](#) procedure for the structure of the rule class table.

### Examples

The following command adds a rule to the rule class.

```

BEGIN
DBMS_RLMGR.ADD_RULE (
    rule_class => 'CompTravelPromo',
    rule_id => 'AB_AV_FL',
    rule_cond =>
        '<condition>
            <and join="Flt.CustId = Car.CustId">
                <object name="Flt">
                    Airline='Abcair' and ToCity='Orlando'
                </object>
                <object name="Car">
                    CarType = 'Luxury'
                </object>
            </and>
        </condition>',
    actprf_nml => 'PromoType, OfferedBy',
    actprf_vall => ''RentalCar', 'Acar'');
END;

```

With proper privileges, the following SQL INSERT statement can be used to add the rule to the rule class.

```

INSERT INTO CompTravelPromo (rlm$ruleid, rlm$rulecond, PromoType, OfferedBy)
VALUES ('AB_AV_FL',
    '<condition>
        <and join="Flt.CustId = Car.CustId">
            <object name="Flt">
                Airline='Abcair' and ToCity='Orlando'
            </object>
            <object name="Car">
                CarType = 'Luxury'
            </object>
        </and>
    </condition>',
    'RentalCar', 'Acar');

```

## CONDITION\_REF Function

This function retrieves the primitive rule condition reference from a rule condition for composite events.

### Syntax

```
DBMS_RLMGR.CONDITION_REF (
    rule_cond IN  VARCHAR2,
    eventnm   IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 123–6** *CONDITION\_REF Function Parameters*

Parameter	Description
rule_cond	Rule condition in XML format
eventnm	Name of the event for which the reference should be retrieved

### Usage Notes

- For a rule condition in XML format, with a root `<condition>` element, this function retrieves the reference to a shared conditional expression on a particular primitive event.
- Use this function in a query operating on the rule class table to find all the references to a given primitive rule condition. To speed-up such queries, one or more functional indexes are defined on the `rlm$rulecond` column of the rule class table using this function signature. In order to make use of the index for a lookup query, the value assigned to the `eventnm` argument should be case sensitive.

### Examples

The following command joins the rule class table with the primitive conditions table to identify all the rule conditions that have references to the shareable primitive conditions (the query uses a functional index defined on the `rlm$rulecond` column). This query identifies all the rule conditions that refer to any shared conditions stored in the `FlightConditions` table.

```
select ctp.rlm$ruleid from CompTravelPromo ctp, FlightConditions fc
where dbms_rlmgr.condition_ref(ctp.rlm$rulecond, 'FLT') = fc.rlm$condid;
```

## CONSUME\_EVENT Function

This function consumes an event and prepares the corresponding rule for action execution. This is required only when the action (or rule execution) is carried by the user's application and not in the callback.

### Syntax

```
DBMS_RLMGR.CONSUME_EVENT (
    rule_class      IN VARCHAR2,
    event_ident     IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 123–7 CONSUME\_EVENT Function Parameters**

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
event_ident	Event identifier obtained from the corresponding rule class results view (or arguments of the action callback procedure in the case of rule class configured for RULE based consumption policy)

### Returns

The function returns:

- 1 -- If the event is successfully consumed.
- 0 -- If the event is expired (owing to duration policy) or consumed by another session prior to this call.

### Usage Notes

- When an EXCLUSIVE consumption policy is set for the events in a rule class, an event must be deleted from the system immediately after the rule it matched is executed (action is executed). When the rule action is carried in the rule class callback procedure by calling the PROCESS\_RULES procedure, the rule manager automatically handles the consumption of the events. However, when you request the results from matching events with rules in a rule class results view using the ADD\_EVENT procedure, you should take appropriate action to indicate the exact rule-event combination that is to be used for rule execution. The CONSUME\_EVENT function performs the required housekeeping services when the unique identifier for the event used in a rule execution is passed in.
- Because there could be a time lag between fetching the rule class matching results and the execution of the user initiated action, the application must execute the action only if the CONSUME\_EVENT call succeeds in consuming the event. This avoids any race condition with parallel sessions trying to consume the same events. When the event is successfully consumed, this call returns 1. In all other cases, it returns 0. A return value of 0 implies that the event is already consumed by another session and hence it is not available for this session.
- The CONSUME\_EVENT function deletes the events configured with EXCLUSIVE consumption policy and does nothing for events configured for 4 consumption policy.

- Unlike the `EXCLUSIVE` and `SHARED` consumption policies, which are determined at the rule class level, you use a `RULE` consumption policy to determine the consumption of an event on a rule by rule basis. That is a subset of the rules in a rule class may be configured such that when they are matched, the event is deleted from the system. At the same time the other set of rules could leave the event in the system even after executing the corresponding action. In this scenario, the action callback procedure implemented by the application developer can call `CONSUME_EVENT` function (with appropriate arguments) to conditionally consume the event for certain rules. Also see the use of [CONSUME\\_PRIM\\_EVENTS Function](#) for rule classes configured for `RULE` consumption policy

## Examples

The following commands identify an event that is used for a rule execution and consumes it using its identifier.

```
var eventid VARCHAR(40);
var evtcnsmd NUMBER;

BEGIN
  SELECT rlm$eventid INTO :eventid FROM MatchingPromos WHERE rownum < 2;

  -- carry the required action for a rule matched by the above event --
  :evtcnsmd := DBMS_RLMGR.CONSUME_EVENT(rule_class => 'TravelPromotion',
                                       event_ident => :eventid);
END;
```



## CONSUME\_PRIM\_EVENTS Function

This function consumes a set of primitive events with all or nothing semantics in the case of a rule class configured with `RULE` based consumption policy.

### Syntax

```
DBMS_RLMGR.CONSUME_PRIM_EVENTS (
    rule_class      IN VARCHAR2,
    event_idents   IN RLM$EVENTIDS)
RETURN NUMBER;
```

### Parameters

**Table 123–8 CONSUME\_PRIM\_EVENTS Function Parameters**

Parameter	Description
<code>rule_class</code>	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
<code>event_ident</code>	Event identifiers obtained from the corresponding rule class results view or the arguments of the action callback procedure

### Returns

The function returns:

- 1 -- If all the events, the identifiers for which are passed in, are successfully consumed.
- 0 -- If one or more primitive event could not be consumed.

### Usage Notes

- When you configure the rule class for `RULE` based consumption policy, it uses the `CONSUME_PRIM_EVENTS` function to consume one or more primitive events that constitute a composite event. This operation succeeds only when all the events passed in are still valid and are available for consumption. Any user initiated action must be implemented after checking the return value of the `CONSUME_PRIM_EVENTS` call.

### Examples

The following commands show the body of the action callback procedure for a rule class configured for `RULE` consumption policy. This demonstrates the use of `CONSUME_PRIM_EVENTS` function to consume the events before executing the action for the matched rules.

```
create or replace procedure PromoAction (
    Flt      AddFlight,
    Flt_EvtId ROWID,    --- rowid for the flight primitive event
    Car      AddRentalCar,
    Car_EvtId ROWID,
    rlm$rule TravelPromotions%ROWTYPE) is
    evtcnsmd NUMBER;
BEGIN
    evtcnsmd := DBMS_RLMGR.CONSUME_PRIM_EVENTS(
        rule_class => 'TravelPromotions',
        event_idents => RLM$EVENTIDS(Flt_EvtId, Car_EvtId));
```

```
    if (evtcnsmd = 1) then
        -- consume operation was successful; perform the action ---
        OfferPromotion (Flt.CustId, rlm$rule.PromoType, rlm$rule.OfferedBy);
    end if;
END;
/
```

## CREATE\_CONDITIONS\_TABLE Procedure

This procedure creates a conditions table, which is a repository for the primitive rule conditions that can be shared by multiple rules from the same or different rule classes. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Creates a conditions table to store shareable primitive conditions defined for a primitive event.

```
DBMS_RLMGR.CREATE_CONDITIONS_TABLE (
    cond_table     IN  VARCHAR2,
    pevent_struct  IN  VARCHAR2,
    stg_clause     IN  VARCHAR2 DEFAULT NULL);
```

Creates a conditions table to store shareable primitive conditions defined for a relational table identified through table aliases.

```
DBMS_RLMGR.CREATE_CONDITIONS_TABLE (
    cond_table     IN  VARCHAR2,
    tab_alias      IN  rlm$table_alias,
    stg_clause     IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 123–9 CREATE\_CONDITIONS\_TABLE Procedure Parameters**

Parameter	Description
pevent_struct	Primitive event structure for which the shareable primitive rule conditions are defined
cond_table	Name of the table storing the primitive rule conditions
stg_clause	Storage clause for the conditions table
tab_alias	Type that identifies the database table for which the shareable primitive rule conditions are defined

### Usage Notes

- This procedure creates a relational table to store the primitive rule conditions that can be shared by multiple rules. It creates the table with the user specified name and it has a VARCHAR2 column to store the unique identifier for each primitive rule condition (`rlm$condid`), an expression data type column to store the conditional expressions (`rlm$condition`), and a VARCHAR2 column to store the descriptions for the primitive rule conditions in plain text (`rlm$conddesc`).
- Once it creates the table, the primitive rule condition can be added or modified using standard DML operations on the conditions table. The conditions table is configured to validate the primitive rule conditions (in the `rlm$condition` column) using the primitive event structure specified for the `pevent_struct` argument.
- A rule class configured with a primitive event structure can include some rule conditions that refer to rows in the conditions table using corresponding identifiers.

## Examples

The following command creates a conditions table that can store shareable primitive rule conditions for the AddRentalCar event structure:

```
BEGIN
  DBMS_RLMGR.CREATE_CONDITIONS_TABLE (
    cond_table    => 'FlightConditions',
    pevent_struct => 'AddFlight',
    stg_clause    => 'TABLESPACE TBS_1');
END;
/
```

## CREATE\_EVENT\_STRUCT Procedure

This procedure creates an event structure.

### Syntax

```
DBMS_RLMGR.CREATE_EVENT_STRUCT (
    event_struct IN VARCHAR2);
```

### Parameters

**Table 123–10** CREATE\_EVENT\_STRUCT Procedure Parameter

Parameter	Description
event_struct	Name of the event structure to be created in the current schema

### Usage Notes

- This procedure creates a dummy event structure in the current schema. One or more attributes can be added to this event structure using the ADD\_ELEMENTARY\_ATTRIBUTE procedure.

### Examples

The following command creates the event structure.

```
BEGIN
    DBMS_RLMGR.CREATE_EVENT_STRUCT(event_struct => 'AddFlight');
END;
```

## CREATE\_EXPFIL\_INDEXES Procedure

This procedure creates expression filter indexes for the rule class if the default indexes have been dropped. If a representative set of rules is stored in the rule class table, the indexes can be tuned for these expressions by collecting statistics.

### Syntax

```
DBMS_RLMGR.CREATE_EXPFIL_INDEXES (
    rule_class IN VARCHAR2,
    coll_stats IN VARCHAR2 default 'NO');
```

### Parameters

**Table 123–11** CREATE\_EXPFIL\_INDEXES Procedure Parameter

Parameter	Description
rule_class	Name of the rule class
coll_stats	To collect expression statistics for building the indexes

### Usage Notes

- Expression filter indexes are used to identify the rule conditions in a rule class for appropriate events. The default indexes created at the time of rule class creation assume that all types of predicates (equality, inequality, and so forth) involving scalar attributes in an event structure are equally likely. The performance of a rule class can be improved by tuning the expression filter indexes for a specific workload. This is achieved either by collecting statistics on a representative workload or by identifying the most common predicate constructs with some domain knowledge.
- The default expression filter indexes created for the rule class can be dropped using the `DBMS_RLMGR.DROP_EXPFIL_INDEXES` procedure. Once the indexes are dropped, they can be recreated using the `DBMS_RLMGR.CREATE_EXPFIL_INDEXES` procedure. When the `coll_stats` argument of the `CREATE_EXPFIL_INDEXES` procedure is set to `YES`, rule condition statistics are collected for the most common predicate constructs and the indexes are created using these statistics. Alternately, a domain expert can manually set the index parameters by identifying the most common and discriminating predicate constructs and then create the indexes with these parameters. Note that the index parameters can be set for each of the primitive event structures associated with the rule class. The index parameters can be assigned to the event structure (which is also the Expression Filter attribute set) using the `DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS` procedure. When the `coll_stats` argument of the `CREATE_EXPFIL_INDEXES` procedure is set to `NO`, the expression filter indexes created for the rule class make use of the default index parameters associated with each primitive event structure. (See the chapter on indexing expressions in *Oracle Database Rules Manager and Expression Filter Developer's Guide* for additional information on tuning the Expression Filter indexes for better performance).
- Related view: `USER_EXPFIL_DEF_INDEX_PARAMS`

## Examples

The following commands collect the statistics for the rules defined in the `CompTravelPromo` rule class and create the expression filter indexes that are based on the most common predicates in the set.

```
BEGIN
DBMS_RLMGR.CREATE_EXPFIL_INDEXES (rule_class => 'CompTravelPromo',
                                coll_stats => 'yes');

END;
/
```

This is an Expression Filter tuning example where the domain knowledge is used to assign specific index parameters. The following commands associate specific index parameters to the `AddFlight` event structure such that the expression filter index created for corresponding expressions are optimized accordingly. The subsequent `CREATE_EXPFIL_INDEXES` step makes use of these index parameters.

```
BEGIN
  DBMS_EXPFIL.DEFAULT_INDEX_PARAMETERS('AddFlight',
    exf$attribute_list (
      exf$attribute (attr_name => 'Airline',
                    attr_oper => exf$indexoper('='),
                    attr_indexed => 'TRUE'),
      exf$attribute (attr_name => 'ToCity',
                    attr_oper => exf$indexoper('='),
                    attr_indexed => 'TRUE'),
      exf$attribute (attr_name => 'Depart',
                    attr_oper => exf$indexoper('=', '<', '>', '>=', '<='),
                    attr_indexed => 'FALSE')
    )
  );
  -- create the indexes after assigning the index parameters --
  DBMS_RLMGR.CREATE_EXPFIL_INDEXES (rule_class => 'CompTravelPromo');

END;
/
```

## CREATE\_INTERFACE Procedure

This procedure creates a rule class interface package that can be used to directly operate on the rule class for efficiency and ease of use.

### Syntax

```
DBMS_RLMGR.CREATE_INTERFACE (
    rule_class IN VARCHAR2,
    interface_nm IN VARCHAR2);
```

### Parameters

**Table 123–12 CREATE\_INTERFACE Procedure Parameter**

Parameter	Description
rule_class	Name of the rule class for which the interface package is created
interface_nm	Name of the PL/SQL package that acts as the interface to the rule application

### Usage Notes

- The common set of DBMS\_RLMGR procedures used for runtime operations such as processing the rules for some events, consuming the events and resetting the session make use of the rule class name passed in as one of the arguments and associate them to the corresponding operations on the rule class. You can the overhead involved in this step by creating a rule class interface package that is used to directly operate on the rule class.
- The rule class interface package is a PL/SQL package that has procedures or functions to process rules (PROCESS\_RULES), add event (ADD\_EVENT), consume events (CONSUME\_EVENT, CONSUME\_PRIM\_EVENTS) and reset rule session (RESET\_SESSION). The operational characteristics of these procedures and functions are the same as those of DBMS\_RLMGR procedures and functions with matching names with two exceptions. Since the rule class interface package is created for a specific rule class, the rule class name is implicit and it need not be passed in as an argument to the procedures and functions of the rule class interface package. Additionally, the rule class interface package has separate PROCESS\_RULES and ADD\_EVENT procedures to accept each primitive event type configured with the rule class. This is in contrast to the same procedures in the DBMS\_RLMGR package, which are generalized to accept the event instances only as a VARCHAR or an AnyData instance.

### Examples

The following commands create the rule class interface package for the CompTravelPromo rule class.

```
BEGIN
    DBMS_RLMGR.CREATE_INTERFACE (rule_class => 'CompTravelPromo',
                                interface_nm => 'TravelPromoRules');
END;
```

The following commands make use of the interface created in previous step to process the rules for an instance of AddFlight event.

```
BEGIN
```



```
TravelPromoRules.process_rules (event_inst =>
    AddFlight(987, 'Abcair', 'Boston', 'Orlando',
              '01-APR-2009', '08-APR-2009');
END;
```

## CREATE\_RULE\_CLASS Procedure

This procedure creates a rule class.

### Syntax

```
DBMS_RLMGR.CREATE_RULE_CLASS (
  rule_class      IN  VARCHAR2,
  event_struct    IN  VARCHAR2,
  action_cbk     IN  VARCHAR2,
  actprf_spec    IN  VARCHAR2  default null,
  rslt_viewnm    IN  VARCHAR2  default null,
  rlcls_prop     IN  VARCHAR2  default <simple/>);
```

### Parameters

**Table 123–13** CREATE\_RULE\_CLASS Procedure Parameters

Parameter	Description
rule_class	Name of the rule class to be created in the current schema
event_struct	Name of the object type or an Expression Filter attribute set in the current schema that represents the event structure for the rule class
action_cbk	Name of the action callback procedure to be created for the rule class
actprf_spec	Specification (name and SQL datatype pairs) for the action preferences associated with the rule class
rslt_viewnm	Name of rule class results view that lists the matching events and rules within a session. A view with this name is created in the current schema.
rlcls_prop	XML document for setting the rule class properties. By default, the rule class created is for simple events (non-composite).

### Usage Notes

- For successful creation of a rule class, you must have sufficient privileges to create views, object types, tables, packages, and procedures.
- This command creates the rule class and its dependent objects in the user's schema. For this operation to succeed the name specified for the event structure must refer to an existing object type or an Expression Filter attribute set in the user's schema. When an object type is used for an event structure, the CREATE\_RULE\_CLASS procedure implicitly creates an attribute set for the object type. In the case of a rule class configured for composite events, the previous procedure also creates attribute sets for the object types that are directly embedded in the event structure's object type (or the attribute set). A maximum of 32 embedded objects (and or or table aliases) can be specified with an event structure that is used for a composite rule class. The types of dependent objects created with this procedure and their structure depend on the properties of the rule class and its event structure. The minimum set of dependent objects created for a rule class is as follows:
  - Rule class table – A rule class table that shares the name of the rule class is created in the user's schema to store the rule definitions (rule identifiers, rule conditions, rule descriptions, and action preferences). This table implicitly has four columns, rlm\$ruleid, rlm\$rulecond, rlm\$enabled, and rlm\$ruledesc to store the rule identifiers, rule conditions, rule states, and rule descriptions respectively. In addition to these four columns, the rule class

table has few columns according to the action preference specification for the rule class. For example, if a `TravelPromotion` rule class uses 'PromoType VARCHAR(20), OfferedBy VARCHAR(20)' as its action preference specification (assigned to `actpref_spec` argument), the rule class table is created with the following structure.

```
TABLE TravelPromotion (
    rlm$ruleid VARCHAR(100),      -- rule identifier column --
    PromoType VARCHAR(20),       -- action preference 1 --
    OfferedBy VARCHAR(20),       -- action preference 2 --
    rlm$rulecond VARCHAR(4000),  -- rule condition --
    rlm$ruledesc VARCHAR(1000),  -- rule description --
    rlm$enabled CHAR(1));       -- rule status --
```

The rule class table structure varies from one rule class to another based on the exact list of action preference categories specified for the rule class.

- Action Callback Procedure – You create the skeleton for the action callback procedure with the given name in the user's schema and it is associated with the rule class. During rule evaluation, the callback procedure is called for each matching rule and event. You must implement the body of the action callback procedure to perform the appropriate action for each rule. The exact action for a rule can be determined based on the event that matched the rule and rule definition along with its action preferences. This information is passed to the action callback procedure through its arguments. Hence, the argument list for the action callback procedure depends on the event structure associated with the rule class and the rule class itself.

In the case of a rule class configured for simple events (`<simple/>` assigned to the properties of the rule class), the event that matches a rule is passed through a `rlm$event` argument that is declared to be of the same type as the event structure. Additionally, the rule definitions are passed to the action callback procedure using an `rlm$rule` argument that is declared as ROWTYPE of the corresponding rule class table. For example, the structure of the `PromoAction` action callback procedure created for a `TravelPromotion` rule class configured for a simple (non-composite) `AddFlight` event structure is as follows:

```
PROCEDURE PromoAction (rlm$event AddFlight,
    rlm$rule TravlePromotion%ROWTYPE);
```

In the case of a rule class created for composite events (`<composite/>` assigned to the properties of the rule class), the action callback procedure is created to pass each primitive event as a separate argument. For example, the `CompPromoAction` action callback procedure created for a rule class `CompTravelPromo` configured for a composite event with `AddFlight` and `AddRentalCar` primitive events are shown as follows:

```
-- composite event structure --
TYPE TSCompEvent (Flt AddFlight,
    Car AddRentalCar);
-- corresponding action callback procedure --
PROCEDURE PromoAction (Flt AddFlight,
    Car AddRentalCar,
    rlm$rule CompTravelPromo%ROWTYPE)
```

The action callback procedure includes additional arguments when the rule class is configured for the `RULE` consumption policy or when the rule class is enabled for one or more collection events. The arguments in these cases

include the identifiers for the events (ROWID data type) in addition to the event instances. You can use these event identifiers to further operate on the matched rules. For example, in the case of the rule class configured for rule consumption, the event identifiers are used to consume the events with `DBMS_RLMGR.CONSUME_PRIM_EVENTS` function. In the case of rule class enabled for collection events, the same identifiers for the collection events can be used to fetch specific aggregate values with the `DBMS_RLMGR.GET_AGGREGATE_VALUE` function.

- Rule class results view – A view to display the results from matching some events with rules is created in the same schema as the rule class. By default, this view is created with a system-generated name. Optionally, the rule class creator can specify a name for this view with the `rlst_viewnm` argument of the `CREATE_RULE_CLASS` procedure. When the events are added to the rule manager within a rule session using the `ADD_EVENT` procedure, the list of matching events and rules are displayed in the rule class results view.

The structure of the view defined for the rule class results depends on the event structure and the action preferences configured with the rule class. Minimally, the view has three columns to display the system generated event identifier (`rlm$eventid`), the identifier of the rule it matches (`rlm$ruleid`), and the rule condition (`rlm$rulecond`). Additionally, it has columns to display the event information and the rule action preferences.

In the case of a rule class configured for simple events, the event information is displayed as `rlm$event` that is declared to be of the event structure type. So, a `MatchingPromos` view created for the `TravelPromotion` rule class configured for a simple `AddFlight` event structure is as follows:

```
VIEW MatchingPromos (
    rlm$eventid ROWID,
    rlm$event AddFlight,
    rlm$ruleid VARCHAR(100),
    PromoType VARCHAR(30), -- action preference 1 --
    OffredBy VARCHAR(30), -- action preference 2 --
    rlm$rulecond VARCHAR(4000),
    rlm$ruledesc VARCHAR(1000)
);
```

In the case of a rule class configured for composite events, the primitive events matching a rule are displayed separately using corresponding columns. For the above `CompTravelPromo` rule class, a `MatchingCompPromos` view is created with the following structure.

```
VIEW MatchingCompPromos (
    rlm$eventid ROWID,
    Flt AddFlight,
    Car AddRentalCar,
    rlm$ruleid VARCHAR(100),
    PromoType VARCHAR(30), -- action preference 1 --
    OffredBy VARCHAR(30), -- action preference 2 --
    rlm$rulecond VARCHAR(4000),
    rlm$ruledesc VARCHAR(1000)
);
```

The values from the `rlm$eventid` column are used to enforce rule class consumption policies when the corresponding rule is executed. See the [CONSUME\\_EVENT Function](#) for more information.

## Examples

The following commands create a rule class for simple events (of AddFlight type).

```
CREATE or REPLACE TYPE AddFlight AS OBJECT (
    CustId NUMBER,
    Airline VARCHAR(20),
    FromCity VARCHAR(30),
    ToCity VARCHAR(30),
    Depart DATE,
    Return DATE);
BEGIN
    DBMS_RLMGR.CREATE_RULE_CLASS (
        rule_class => 'TravelPromotion', -- rule class name --
        event_struct => 'AddFlight', -- event struct name --
        action_cbk => 'PromoAction', -- callback proc name --
        rslt_viewnm => 'MatchingPromos', -- results view --
        actprf_spec => 'PromoType VARCHAR(20),
                        OfferedBy VARCHAR(20)');
END;
```

The following commands create a rule class for composite events consisting of two primitive events (AddFlight and AddRentalCar).

```
CREATE or REPLACE TYPE TSCompEvent (Flt AddFlight,
                                    Car AddRentalCar);
BEGIN
    DBMS_RLMGR.CREATE_RULE_CLASS (
        rule_class => 'CompTravelPromo', -- rule class name --
        event_struct => 'TSCompEvent', -- event struct name --
        action_cbk => 'CompPromoAction', -- callback proc name --
        rslt_viewnm => 'MatchingCompPromos', -- results view --
        actprf_spec => 'PromoType VARCHAR(20),
                        OfferedBy VARCHAR(20)',
        properties => '<composite/>');
END;
```

## DELETE\_RULE Procedure

This procedure deletes a rule from a rule class.

### Syntax

```
DBMS_RLMGR.DELETE_RULE (
    rule_class    IN    VARCHAR2,
    rule_id       IN    VARCHAR2);
```

### Parameters

**Table 123–14** *DELETE\_RULE Procedure Parameters*

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
rule_id	Identifier for the rule to be deleted

### Usage Notes

- Use this procedure to delete a rule from the rule class. The identifier for the rule to be deleted can be obtained by querying the rule class table (that shares the same name as the rule class). Alternately, the owner of the rule class can use a SQL `DELETE` statement on one rule class table to delete a rule.
- When you use the schema extended name for the rule class, you must have the `DELETE RULE` privilege on the rule class. See the [GRANT\\_PRIVILEGE Procedure](#) for more information.

---

**Note:** `AUTOCOMMIT` property of the rule class is ignored if the rules are deleted with the SQL `DELETE` statement instead of the `DELETE_RULE` procedure.

---

- See the [CREATE\\_RULE\\_CLASS Procedure](#) for the structure of the rule class table.

### Examples

The following command deletes a rule from the rule class.

```
BEGIN
    DBMS_RLMGR.DELETE_RULE (
        rule_class => 'CompTravelPromo',
        rule_id    => 'AB_AV_FL');
END;
```

Alternately, you can issue the following SQL `DELETE` statement to delete the above rule from the rule class.

```
DELETE FROM CompTravelPromo WHERE rlm$ruleid = 'AB_AV_FL';
```

## DROP\_CONDITIONS\_TABLE Procedure

This procedure drops the conditions table.

### Syntax

```
DBMS_RLMGR.DROP_CONDITIONS_TABLE (
    cond_table IN VARCHAR2);
```

### Parameters

**Table 123–15** *DROP\_CONDITIONS\_TABLE Procedure Parameters*

Parameter	Description
cond_table	Name of conditions table in the user schema

### Usage Notes

- This procedure drops the table that stores the shareable conditional expressions. If one of the conditional expressions in this table is used to form a rule condition in a rule class, the drop operation fails with an appropriate error.

### Examples

The following command drops the conditions table:

```
BEGIN
    DBMS_RLMGR.DROP_CONDITIONS_TABLE (cond_table => 'FlightConditions');
END;
/
```

## DROP\_EVENT\_STRUCT Procedure

This procedure drops an event structure.

### Syntax

```
DBMS_RLMGR.DROP_EVENT_STRUCT (  
    event_struct IN VARCHAR2);
```

### Parameters

**Table 123–16** *DROP\_EVENT\_STRUCT Procedure Parameter*

Parameter	Description
event_struct	Name of event structure in the current schema

### Usage Notes

- This procedure drops the event structure from the current schema. This drops all the dependent objects created to manage the event structure.

### Examples

The following command drops the event structure.

```
BEGIN  
    DBMS_RLMGR.DROP_EVENT_STRUCT(event_struct => 'AddFlight');  
END;
```



## DROP\_EXPFIL\_INDEXES Procedure

This procedure drops the expression filter indexes created for a rule class.

### Syntax

```
DBMS_RLMGR.DROP_EXPFIL_INDEXES (
  rule_class IN VARCHAR2);
```

### Parameters

**Table 123–17** *DROP\_EXPFIL\_INDEXES Procedure Parameter*

Parameter	Description
rule_class	Name of the rule class

### Usage Notes

This procedure drops all the expression filter indexes associated with a rule class. You can recreate the indexes using the `DBMS_RLMGR.CREATE_EXPFIL_INDEXES` call.

### Examples

The following command drops the expression filter indexes created for the `CompTravelPromo` rule class.

```
BEGIN
DBMS_RLMGR.DROP_EXPFIL_INDEXES (rule_class => 'CompTravelPromo');
END;
/
```

## DROP\_INTERFACE Procedure

This procedure drops the rule class interface package created for a rules application.

### Syntax

```
DBMS_RLMGR.DROP_INTERFACE (  
    interface_nm IN VARCHAR2);
```

### Parameters

**Table 123–18** *DROP\_INTERFACE Procedure Parameter*

Parameter	Description
interface_nm	Name of the PL/SQL package that acts as the interface to the rule application

### Usage Notes

This procedure drops the rule class interface package created with the `DBMS_RLMGR.CREATE_INTERFACE` call.

### Examples

The following command drops the rule class interface package `TravelPromoRules`.

```
BEGIN  
    DBMS_RLMGR.DROP_INTERFACE (interface_nm => 'TravelPromoRules'  
    END;
```

## DROP\_RULE\_CLASS Procedure

This procedure drops a rule class.

### Syntax

```
DBMS_RLMGR.DROP_RULE_CLASS (
    rule_class IN VARCHAR2);
```

### Parameters

**Table 123–19** *DROP\_RULE\_CLASS Procedure Parameter*

Parameter	Description
rule_class	Name of rule class in the current schema

### Usage Notes

- This procedure drops the rule class from the current schema. This drops all the dependent objects created to manage the rule class. Because an event structure in a user's schema can be shared across multiple rule classes, the event structure is not dropped with this command. You must use the `DROP_EVENT_STRUCTURE` procedure for the composite event as well as the individual primitive events to cleanup unused event structures.

### Examples

The following command drops the rule class.

```
BEGIN
    DBMS_RLMGR.DROP_RULE_CLASS(rule_class => 'CompTravelPromo');
END;
```

## EXTEND\_EVENT\_STRUCT Procedure

This is used to extend the primitive event structure used by one or more rule classes by adding a new attribute.

### Syntax

```
DBMS_RLMGR.EXTEND_EVENT_STRUCT (
    event_struct    IN   VARCHAR2,
    attr_name       IN   VARCHAR2,
    attr_type       IN   VARCHAR2,
    attr_defvl     IN   VARCHAR2 default NULL);
```

### Parameters

**Table 123–20** *EXTEND\_EVENT\_STRUCT Procedure Parameter*

Parameter	Description
event_struct	Name of the event structure to which this attribute is added
attr_name	Name of the elementary attribute to be added. No two attributes in a set can have the same name.
attr_type	Data type of the attribute. This argument accepts any standard SQL data type or the name of an object type that is accessible to the current user.
attr_defvl	Default value for the elementary attribute

### Usage Notes

- This procedure extends a primitive event structure already associated with a rule class to include a new attribute. You can use this procedure mostly to migrate a fully developed rules application to use extended event structures and you should not use it in the place of the `ADD_ELEMENTARY_ATTRIBUTE` call. Unlike the `ADD_ELEMENTARY_ATTRIBUTE` call, which builds an event structure one attribute at a time, the `EXTEND_EVENT_STRUCT` call evolves the object type associated with the event structure to include the new attributes and performs some maintenance operations on the dependent objects.
- The usage of the `EXTEND_EVENT_STRUCT` call is similar to that of the `ADD_ELEMENTARY_ATTRIBUTE` call with the same set of arguments. Table alias attributes and attributes of text and spatial data types cannot be added to the event structure using the `EXTEND_EVENT_STRUCT` call.

### Examples

The following commands add an attribute to the `AddRentalCar` event structure that is used by the `CompTravelPromo` rule class.

```
BEGIN
    DBMS_RLMGR.EXTEND_EVENT_STRUCT (
        event_struct => 'AddRentalCar',
        attr_name     => 'PrefMemberId',
        attr_type     => 'VARCHAR2(30)');
END;
```

## GET\_AGGREGATE\_VALUE Function

This function retrieves the aggregate value computed for a collection event.

### Syntax

```
DBMS_RLMGR.GRANT_PRIVILEGE (
  rule_class      IN VARCHAR2,
  event_ident     IN VARCHAR2,
  aggr_func       IN VARCHAR2) RETURN VARCHAR2;
```

### Parameters

**Table 123–21** GET\_AGGREGATE\_VALUE Function Parameters

Parameter	Description
rule_class	Name of the rule class for the collection event
event_ident	System-generated identifier for the collection event
aggr_func	Signature for the aggregate value to be retrieved

### Usage Notes

- When a rule condition with `collection` construct matches a set of events, an instance representing the collection event and a system-generated identifier for the collection event are passed into the action callback procedure. This event identifier can be used to fetch any aggregate values that are computed as part of the collection event evaluation.
- Within a collection construct in a rule condition, the aggregate functions can be included in the `HAVING` clause or in the `COMPUTE` clause. For example, the following rule condition computes three aggregate values for `sum(amount)`, `count(*)`, and `max(amount)`. At the time of action execution, these values can be obtained using the identifier for the collection event that represents all the instances of `BankTransaction` with the same `subjectId` (the attribute on which the events are grouped).

```
<condition>
  <collection name="bank" groupby="subjectId"
    having="sum(amount) > 10000"
    compute="max(amount), count(*)" />
</condition>
```

- The signature for the aggregate function is bound to the `aggr_func` argument of the `GET_AGGREGATE_VALUE` function to fetch the specific aggregate value. If the value is a `NUMBER` or a `DATE` data type, it returns the equivalent `VARCHAR` representation. It returns a `NULL` value if an attempt was made to fetch an aggregate value that is never computed as part of the collection event.

### Examples

The following example shows a sample implementation of the action callback procedure that prints the computed aggregate values as part of action execution. In this particular case, the `BankTransaction` primitive event is enabled for collections.

```
CREATE OR REPLACE PROCEDURE LAWENFORCEMENTCBK (
  bank            banktransaction,
  bankcollid      rowid,
```

```
transport          transportation,
fldrpt            fieldreport,
rlm$rule          LawEnforcementRC%ROWTYPE) IS
aggrval           VARCHAR(30);
begin
  dbms_output.put_line('Mathing Rule : '||rlm$rule.rlm$ruleid||chr(10));

  if (bank is not null) then
    dbms_output.put_line('-->Bank Transactions by ('||bank.subjectId||')'||chr(10));

    aggrval := dbms_rlmgr.get_aggregate_value(rule_class =>'LawEnforcementRC',
                                              event_ident => bankcollid,
                                              aggr_func  => 'sum(amount)');

    if (aggrval is not null) then
      dbms_output.put_line('---> Sum of the amounts is : '||aggrval||chr(10));
    end if;
    . . .
  end if;
end;
```

## GRANT\_PRIVILEGE Procedure

This procedure grants privileges on a rule class to another user.

### Syntax

```
DBMS_RLMGR.GRANT_PRIVILEGE (
  rule_class      IN  VARCHAR2,
  priv_type       IN  VARCHAR2,
  to_user         IN  VARCHAR2);
```

### Parameters

**Table 123–22 GRANT\_PRIVILEGE Procedure Parameters**

Parameter	Description
rule_class	Name of the rule class in the current schema
priv_type	Type of rule class privilege to be granted
to_user	User to whom the privilege is to be granted

### Usage Notes

- This procedure grants appropriate privileges to a user who is not the owner of the rule class. The types of privileges that can be granted to a user are:
  - PROCESS RULES: A user with PROCESS RULES privilege on a rule class can process the rules in the rule class using the PROCESS\_RULES procedure or the ADD\_EVENT procedure. Also, the user with this privilege can select from the corresponding rule class results view.
  - ADD RULE: A user with ADD RULE privilege on a rule class can add rules to a rule class. Alternatively, the owner of the rule class can grant the INSERT privilege on one rule class table to other users.
  - DELETE RULE: A user with DELETE RULE privilege on a rule class can delete rules from a rule class. Alternatively, the owner of the rule class can grant the DELETE privilege on one rule class table to other users.
  - ALL: Granting the ALL privilege on a rule class is equivalent to granting all the above privileges on the rule class to the user.
- The owner of the rule class always has privileges to drop a rule class, process rules in a rule class, add rules and delete rules from a rules class. Only the owner of the rule class can drop a rule class and this privilege cannot be granted to another user.
- You must have the EXECUTE privilege on the primitive event types associated with a rule class before you make use of the corresponding rule class results view.

### Examples

The following command grants PROCESS RULES privilege on TravelPromo rule class to the user SCOTT.

```
BEGIN
  DBMS_RLMGR.GRANT_PRIVILEGE(rule_class => 'TravelPromo',
                              priv_type => 'PROCESS RULES',
                              to_user => 'SCOTT');
END;
```

## PROCESS\_RULES Procedure

This procedure processes the rules for a given event. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Processes the rules for a string representation of the event instance being added to the rule class:

```
DBMS_RLMGR.PROCESS_RULES (
    rule_class    IN VARCHAR2,
    event_inst    IN VARCHAR2,
    event_type    IN VARCHAR2 default null);
```

Processes the rules for an AnyData representation of the event instance being added to the rule class:

```
DBMS_RLMGR.PROCESS_RULES (
    rule_class    IN VARCHAR2,
    event_inst    IN sys.AnyData);
```

### Parameters

**Table 123–23** *PROCESS\_RULES Procedure Parameters*

Parameter	Description
rule_class	Name of the rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.
event_inst	String or AnyData representation of the event instance being added to the rule class
event_type	Type of event instance assigned to the event_inst argument when the string representation of the event instance is used for a rule class configured for composite events

### Usage Notes

- This procedure is used to process the rules in a rule class for an event instance assigned to the event\_inst argument.
- In the case of a rule class configured for simple events (non-composite), the event instance is an instantiation of the corresponding event structure. The rules are evaluated (conclusively) for this event and the corresponding action callback procedure is called for each matching rule. If the event does not match any rule, no further action is performed. If the event matches two or more rules, the ordering clause configured for the rule class is used to order them accordingly to invoke the action callback procedure. If the rule class is configured for EXCLUSIVE consumption policy, once the first rule in this order is executed (and the corresponding action callback procedure is called), the rest of the rules that matched the event are ignored.
- In the case of a rule class configured for composite events, the event instance assigned to the event\_inst argument is an instantiation of one of the primitive type within the composite event. When the instance is represented as a string, the corresponding type name should be assigned to the event\_type argument. The PROCESS\_RULES call on a rule class configured for composite events performs



various actions depending on the state of the rule class and the kind of rules in the rule class. Note the following.

- The rules operating only on the primitive event passed in are evaluated conclusively and the action callback procedure is called for the matching rules, as described in previous paragraph.
- In the case of a rule operating on more than one primitive event, the event instance passed through `PROCESS_RULES` procedure could match only a part of the rule.
  - \* If there are other primitive event instances that matches the rest of the rule, the current event instance is combined with the other instances to form a complete composite event that matches a rule in the rule class. So, the event instance assigned to the `event_inst` argument of the `PROCESS_RULES` procedure could be combined with various other primitive events (previously processed) to evaluate one or more rules conclusively. The action callback procedure for the rule class is called for each such combination of primitive events (composite event) and the rule. The ordering clause for the rule class and the consumption policy for the primitive events in taken into account while invoking the action callback procedure.
  - \* If there is no other primitive event that matches the rest of the rule, the current event instance and its (incremental) evaluation results are recorded in the database. These results are preserved until either the event is consumed or deleted from the system owing to the duration policy used for the rule class.

## Examples

The following command processes the rules in the `TravelPromotion` rule class for the given events.

```
BEGIN
  DBMS_RLMGR.PROCESS_RULES (
    rule_class => 'TravelPromotion',
    event_inst =>
      AddFlight.getVarchar(987, 'Abcair', 'Boston', 'Orlando',
                          '01-APR-2003', '08-APR-2003'));
END;
```

The following commands process the rules in the `CompTravelPromo` rule class for the two primitive events shown.

```
BEGIN
  DBMS_RLMGR.PROCESS_RULES(
    rule_class => 'CompTravelPromo',
    event_inst =>
      AddFlight.getVarchar(987, 'Abcair', 'Boston', 'Orlando',
                          '01-APR-2003', '08-APR-2003'),
    event_type => 'AddFlight');
  DBMS_RLMGR.PROCESS_RULES(
    rule_class => 'Scott.CompTravelPromo',
    event_inst =>
      AnyData.convertObject(AddRentalCar(987, 'Luxury', '03-APR-2003',
                                          '08-APR-2003', NULL));
END;
```

## PURGE\_EVENTS Procedure

This procedure resets the incremental state maintained by the rule class by removing all the events associated with the rule class and purging any state information pertaining to rules matching some events.

### Syntax

```
DBMS_RLMGR.PURGE_EVENTS (  
    rule_class IN VARCHAR2);
```

### Parameters

**Table 123–24 PURGE\_EVENTS Procedure Parameters**

Parameter	Description
rule_class	Name of rule class in the current schema

### Usage Notes

- Use this procedure while developing rules applications using Rules Manager. You can test the rules defined in the rule class with hypothetical events and then remove all these events by issuing this procedure call. This call cleans up all the events in the events repository and purges any partial state information associated with the matching rules.

### Examples

The following command removes the events associated with the `CompTravelPromo` rule class:

```
BEGIN  
    DBMS_RLMGR.PURGE_EVENTS (rule_class => 'CompTravelPromo');  
END;  
/
```

## RESET\_SESSION Procedure

This procedure starts a new session and thus discards the results in the rule class results view.

### Syntax

```
DBMS_RLMGR.RESET_SESSION (
    rule_class IN VARCHAR2);
```

### Parameters

**Table 123–25** *RESET\_SESSION Procedure Parameter*

Parameter	Description
rule_class	Name of rule class. A schema extended rule class name can be used to refer to a rule class that does not belong to the current schema.

### Usage Notes

- When you use the `ADD_EVENT` procedure to add events to the rule class, the results from matching rules with events are recorded in the rule class results view. By default, these results are reset at the end of the database session. Alternately, you can use the [RESET\\_SESSION Procedure](#) to reset and start a new rule session within a database session.
- This procedure is only applicable while using [ADD\\_EVENT Procedure](#) to evaluate the rules.

### Examples

The following command resets a rule class session.

```
BEGIN
    DBMS_RLMGR.RESET_SESSION(
        rule_class => 'CompTravelPromo');
END;
```

## REVOKE\_PRIVILEGE Procedure

This procedure revokes privileges on a rule class from another user.

### Syntax

```
DBMS_RLMGR.REVOKE_PRIVILEGE (
    rule_class    IN  VARCHAR2,
    priv_type     IN  VARCHAR2,
    from_user     IN  VARCHAR2);
```

### Parameters

**Table 123–26 REVOKE\_PRIVILEGE Procedure Parameters**

Parameter	Description
rule_class	Name of the rule class in the current schema
priv_type	Type of rule class privilege to be revoked
from_user	User from whom the privilege is to be revoked

### Usage Notes

- This procedure revokes appropriate privileges from a user. The types of privileges that can be revoked are the same as the types listed in the description of the [GRANT\\_PRIVILEGE Procedure](#). Rule class privileges cannot be revoked from the owner of the rule class.

### Examples

The following command revokes PROCESS RULES privilege on TravelPromo rule class from the user SCOTT.

```
BEGIN
    DBMS_RLMGR.REVOKE_PRIVILEGE(rule_class => 'TravelPromo',
                                priv_type  => 'PROCESS RULES',
                                from_user  => 'SCOTT');
END;
```

## SYNC\_TEXT\_INDEXES Procedure

This procedure synchronizes the indexes defined to process the predicates involving the CONTAINS operator in rule conditions.

### Syntax

```
DBMS_RLMGR.SYNC_TEXT_INDEXES (
    rule_class IN VARCHAR2);
```

### Parameters

**Table 123–27 SYNC\_TEXT\_INDEXES Procedure Parameters**

Parameter	Description
rule_class	Name of the rule class in the current schema

### Usage Notes

- When a rule class is configured for events with one or more text attributes, the text predicates in the corresponding rule conditions are processed using CTXRULE indexes. Unlike other types of indexes (bitmap for scalar and XML predicates or spatial for spatial predicates) used to process other types of predicates in the rule conditions, the CTXRULE indexes are not transactional in nature. That is, if the text predicates in a rule condition are modified in a database transaction, the new predicates are not automatically reflected in the corresponding CTXRULE index. This could result in inconsistent results while matching events with the rule conditions. All the CTXRULE indexes associated with a rule class can be synchronized with the latest rule conditions using this procedure.

You must have EXECUTE privilege on the CTX\_DDL package for successful synchronization of the text indexes.

### Examples

The following command synchronizes any text indexes associated CompTravelPromo rule class:

```
BEGIN
    DBMS_RLMGR.SYNC_TEXT_INDEXES (rule_class => 'CompTravelPromo');
END;
/
```



The DBMS\_RLS package contains the fine-grained access control administrative interface, which is used to implement Virtual Private Database (VPD). DBMS\_RLS is available with the Enterprise Edition only.

**See Also:** *Oracle Database Security Guide* for usage information on DBMS\_RLS .

This chapter contains the following topics:

- [Using DBMS\\_RLS](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_RLS Subprograms](#)

---

## Using DBMS\_RLS

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)



## Overview

The functionality to support fine-grained access control is based on dynamic predicates, where security rules are not embedded in views, but are acquired at the statement parse time, when the base table or view is referenced in a DML statement.

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```
DBMS_RLS.ADD_POLICY (
    'hr', 'employees', 'emp_policy', 'hr', 'emp_sec', 'select');
```

Whenever the EMPLOYEES table, under the HR schema, is referenced in a query or subquery (SELECT), the server calls the EMP\_SEC function (under the HR schema). This function returns a predicate specific to the current user for the EMP\_POLICY policy. The policy function may generate the predicates based on the session environment variables available during the function call. These variables usually appear in the form of application contexts. The policy can specify any combination of security-relevant columns and of these statement types: INDEX, SELECT, INSERT, UPDATE, or DELETE.

The server then produces a transient view with the text:

```
SELECT * FROM hr.employees WHERE P1
```

Here, P1 (for example, where SAL > 10000, or even a subquery) is the predicate returned from the EMP\_SEC function. The server treats the EMPLOYEES table as a view and does the view expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

If the predicate contains subqueries, then the owner (definer) of the policy function is used to resolve objects within the subqueries and checks security for those objects. In other words, users who have access privilege to the policy-protected objects do not need to know anything about the policy. They do not need to be granted object privileges for any underlying security policy. Furthermore, the users do not require EXECUTE privilege on the policy function, because the server makes the call with the function definer's right.

---

**Note:** The transient view can preserve the updatability of the parent object because it is derived from a single table or view with predicate only; that is, no JOIN, ORDER BY, GROUP BY, and so on.

---

DBMS\_RLS also provides the interface to drop or enable security policies. For example, you can drop or enable the EMP\_POLICY with the following PL/SQL statements:

```
DBMS_RLS.DROP_POLICY('hr', 'employees', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('hr', 'employees', 'emp_policy', TRUE);
```

## Security Model

A security check is performed when the transient view is created with a subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view's definer for security check and object lookup.

## Operational Notes

The DBMS\_RLS procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS\_RLS procedures are part of the DDL transaction.

For example, you may create a trigger for CREATE TABLE. Inside the trigger, you may add a column through ALTER TABLE, and you can add a policy through DBMS\_RLS. All these operations are in the same transaction as CREATE TABLE, even though each one is a DDL statement. The CREATE TABLE succeeds only if the trigger is completed successfully.

Views of current cursors and corresponding predicates are available from v\$vpd\_policies.

A synonym can reference only a view or a table.

---

## Summary of DBMS\_RLS Subprograms

**Table 124–1 DBMS\_RLS Package Subprograms**

Subprogram	Description
<a href="#">ADD_GROUPED_POLICY Procedure</a> on page 124-7	Adds a policy associated with a policy group
<a href="#">ADD_POLICY Procedure</a> on page 124-9	Adds a fine-grained access control policy to a table, view, or synonym
<a href="#">ADD_POLICY_CONTEXT Procedure</a> on page 124-13	Adds the context for the active application
<a href="#">CREATE_POLICY_GROUP Procedure</a> on page 124-14	Creates a policy group
<a href="#">DELETE_POLICY_GROUP Procedure</a> on page 124-15	Deletes a policy group
<a href="#">DISABLE_GROUPED_POLICY Procedure</a> on page 124-16	Disables a row-level group security policy
<a href="#">DROP_GROUPED_POLICY Procedure</a> on page 124-17	Drops a policy associated with a policy group
<a href="#">DROP_POLICY Procedure</a> on page 124-18	Drops a fine-grained access control policy from a table, view, or synonym
<a href="#">DROP_POLICY_CONTEXT Procedure</a> on page 124-19	Drops a driving context from the object so that it will have one less driving context
<a href="#">ENABLE_GROUPED_POLICY Procedure</a> on page 124-20	Enables or disables a row-level group security policy
<a href="#">ENABLE_POLICY Procedure</a> on page 124-21	Enables or disables a fine-grained access control policy
<a href="#">REFRESH_GROUPED_POLICY Procedure</a> on page 124-22	Reparses the SQL statements associated with a refreshed policy
<a href="#">REFRESH_POLICY Procedure</a> on page 124-23	Causes all the cached statements associated with the policy to be reparsed

## ADD\_GROUPED\_POLICY Procedure

This procedure adds a policy associated with a policy group.

### Syntax

```
DBMS_RLS.ADD_GROUPED_POLICY (
  object_schema      IN VARCHAR2 NULL,
  object_name        IN VARCHAR2,
  policy_group       IN VARCHAR2 'SYS_DEFAULT',
  policy_name        IN VARCHAR2,
  function_schema    IN VARCHAR2 NULL,
  policy_function     IN VARCHAR2,
  statement_types    IN VARCHAR2 NULL,
  update_check       IN BOOLEAN FALSE,
  enable             IN BOOLEAN TRUE,
  static_policy      IN BOOLEAN FALSE,
  policy_type        IN BINARY_INTEGER NULL,
  long_predicate     BOOLEAN FALSE,
  sec_relevant_cols  IN VARCHAR2,
  sec_relevant_cols_opt IN BINARY_INTEGER NULL);
```

### Parameters

**Table 124–2 ADD\_GROUPED\_POLICY Procedure Parameters**

Parameter	Description
object_schema	The schema containing the table, view, or synonym. The default is NULL, which means that the current user schema is used as the object_schema.
object_name	The name of the table, view, or synonym to which the policy is added.
policy_group	The name of the policy group that the policy belongs to.
policy_name	The name of the policy; must be unique for the same table or view.
function_schema	The schema owning the policy function. The default is NULL, which means that the current user schema is used as the function_schema.
policy_function	The name of the function that generates a predicate for the policy. If the function is defined within a package, the name of the package must be present.
statement_types	Statement types to which the policy applies. It can be any combination of INDEX, SELECT, INSERT, UPDATE, or DELETE. The default is to apply to all of these types except INDEX.
update_check	For INSERT and UPDATE statements only, setting update_check to TRUE causes the server to check the policy against the value after INSERT or UPDATE.
enable	Indicates if the policy is enable when it is added. The default is TRUE.
static_policy	The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.

**Table 124–2 (Cont.) ADD\_GROUPED\_POLICY Procedure Parameters**

Parameter	Description
<code>policy_type</code>	Default is <code>NULL</code> , which means <code>policy_type</code> is decided by the value of <code>static_policy</code> . The available policy types are listed in <a href="#">Table 124–4</a> . Specifying any of these policy types overrides the value of <code>static_policy</code> .
<code>long_predicate</code>	Default is <code>FALSE</code> , which means the policy function can return a predicate with a length of up to 4000 bytes. <code>TRUE</code> means the predicate text string length can be up to 32K bytes. Policies existing prior to the availability of this parameter retain a 32K limit.
<code>sec_relevant_cols</code>	Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.
<code>sec_relevant_cols_opt</code>	Use with <code>sec_relevant_cols</code> to display all rows for column-level VPD filtered queries ( <code>SELECT</code> only), but where sensitive columns appear as <code>NULL</code> . Default is set to <code>NULL</code> , which allows the filtering defined with <code>sec_relevant_cols</code> to take effect. Set to <code>dbms_rls.ALL_ROWS</code> to display all rows, but with sensitive column values, which are filtered by <code>sec_relevant_cols</code> , displayed as <code>NULL</code> . See " <a href="#">Usage Notes</a> " on page 124-11 for restrictions and additional information about this option.

## Usage Notes

- This procedure adds a policy to the specified table, view, or synonym and associates the policy with the specified policy group.
- The policy group must have been created by using the [CREATE\\_POLICY\\_GROUP Procedure](#) on page 124-14.
- The policy name must be unique within a policy group for a specific object.
- Policies from the default policy group, `SYS_DEFAULT`, are always executed regardless of the active policy group; however, fine-grained access control policies do not apply to users with `EXEMPT ACCESS POLICY` system privilege.
- If no `object_schema` is specified, the current user's schema is assumed.
- If no `function_schema` is specified, the current user's schema is assumed.

## ADD\_POLICY Procedure

This procedure adds a fine-grained access control policy to a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

**See Also:** [Operational Notes](#) on page 124-5

A COMMIT is also performed at the end of the operation.

### Syntax

```
DBMS_RLS.ADD_POLICY (
    object_schema          IN VARCHAR2 NULL,
    object_name            IN VARCHAR2,
    policy_name            IN VARCHAR2,
    function_schema        IN VARCHAR2 NULL,
    policy_function        IN VARCHAR2,
    statement_types        IN VARCHAR2 NULL,
    update_check           IN BOOLEAN  FALSE,
    enable                 IN BOOLEAN  TRUE,
    static_policy          IN BOOLEAN  FALSE,
    policy_type            IN BINARY_INTEGER NULL,
    long_predicate         IN BOOLEAN  FALSE,
    sec_relevant_cols      IN VARCHAR2 NULL,
    sec_relevant_cols_opt IN BINARY_INTEGER NULL);
```

### Parameters

**Table 124–3 ADD\_POLICY Procedure Parameters**

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_schema is specified, the current user's schema is assumed.
object_name	Name of table, view, or synonym to which the policy is added.
policy_name	Name of policy to be added. It must be unique for the same table or view.
function_schema	Schema of the policy function (current default schema, if NULL). If no function_schema is specified, the current user's schema is assumed.
policy_function	Name of a function which generates a predicate for the policy. If the function is defined within a package, then the name of the package must be present.
statement_types	Statement types to which the policy applies. It can be any combination of INDEX, SELECT, INSERT, UPDATE, or DELETE. The default is to apply to all of these types except INDEX.
update_check	Optional argument for INSERT or UPDATE statement types. The default is FALSE. Setting update_check to TRUE causes the server to also check the policy against the value after insert or update.
enable	Indicates if the policy is enabled when it is added. The default is TRUE.

**Table 124–3 (Cont.) ADD\_POLICY Procedure Parameters**

Parameter	Description
<code>static_policy</code>	The default is <code>FALSE</code> . If it is set to <code>TRUE</code> , the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for <code>SYS</code> or the privileged user who has the <code>EXEMPT ACCESS POLICY</code> privilege.
<code>policy_type</code>	Default is <code>NULL</code> , which means <code>policy_type</code> is decided by the value of <code>static_policy</code> . The available policy types are listed in <a href="#">Table 124–4</a> . Specifying any of these policy types overrides the value of <code>static_policy</code> .
<code>long_predicate</code>	Default is <code>FALSE</code> , which means the policy function can return a predicate with a length of up to 4000 bytes. <code>TRUE</code> means the predicate text string length can be up to 32K bytes. Policies existing prior to the availability of this parameter retain a 32K limit.
<code>sec_relevant_cols</code>	Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.
<code>sec_relevant_cols_opt</code>	Use with <code>sec_relevant_cols</code> to display all rows for column-level VPD filtered queries ( <code>SELECT</code> only), but where sensitive columns appear as <code>NULL</code> . Default is set to <code>NULL</code> , which allows the filtering defined with <code>sec_relevant_cols</code> to take effect. Set to <code>dbms_rls.ALL_ROWS</code> to display all rows, but with sensitive column values, which are filtered by <code>sec_relevant_cols</code> , displayed as <code>NULL</code> . See "Usage Notes" on page 124-11 for restrictions and additional information about this option.

**Table 124–4 DBMS\_RLS.ADD\_POLICY Policy Types**

Policy Type	Description
<code>STATIC</code>	Predicate is assumed to be the same regardless of the runtime environment. Static policy functions are executed once and then cached in SGA. Statements accessing the same object do not reexecute the policy function. However, each execution of the same cursor could produce a different row set even for the same predicate because the predicate may filter the data differently based on attributes such as <code>SYS_CONTEXT</code> or <code>SYSDATE</code> . Applies to only one object.
<code>SHARED_STATIC</code>	Same as <code>STATIC</code> except that the server first looks for a cached predicate generated by the same policy function of the same policy type. Shared across multiple objects.
<code>CONTEXT_SENSITIVE</code>	Server re-evaluates the policy function at statement execution time if it detects context changes since the last use of the cursor. For session pooling where multiple clients share a database session, the middle tier must reset context during client switches. Note that the server does not cache the value returned by the function for this policy type; it always executes the policy function on statement parsing. Applies to only one object.
<code>SHARED_CONTEXT_SENSITIVE</code>	Same as <code>CONTEXT_SENSITIVE</code> except that the server first looks for a cached predicate generated by the same policy function of the same policy type within the same database session. If the predicate is found in the session memory, the policy function is not reexecuted and the cached value is valid until session private application context changes occur. Shared across multiple objects.
<code>DYNAMIC</code>	The default policy type. Server assumes the predicate may be affected by any system or session environment at any time, and so always reexecutes the policy function upon each statement parsing and execution. Applies to only one object.



## Usage Notes

- SYS is free of any security policy.
- If no `object_schema` is specified, the current user's schema is assumed.
- If no `function_schema` is specified, the current user's schema is assumed.
- The policy functions are called by the server. Following is the interface for the function:

```
FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2)
    RETURN VARCHAR2
--- object_schema is the schema owning the table or view.
--- object_name is the name of table, view, or synonym to which the policy
applies.
```

- The policy functions must have the purity level of `WNDS` (write no database state).

**See Also:** The *Oracle Database Advanced Application Developer's Guide* has more details about the `RESTRICT_REFERENCES` pragma.

- Predicates generated from different VPD policies for the same object have the combined effect of a conjunction (ANDed) of all the predicates.
- The security check and object lookup are performed against the owner of the policy function for objects in the subqueries of the dynamic predicates.
- If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.
- When a table alias is required (for example, parent object is a type table) in the predicate, the name of the table or view itself must be used as the name of the alias. The server constructs the transient view as something like

```
"select c1, c2, ... from tab tab where <predicate>"
```

- Validity of the function is checked at runtime for ease of installation and other dependency issues during import and export.
- Column-level VPD column masking behavior (specified with `sec_relevant_cols_opt => dbms_ols.ALL_ROWS`) is fundamentally different from all other VPD policies, which return only a subset of rows. Instead the column masking behavior returns all rows specified by the user's query, but the sensitive column values display as `NULL`. The restrictions for this option are as follows:
  - Only applies to `SELECT` statements
  - Unlike regular VPD predicates, the masking condition that is generated by the policy function must be a simple boolean expression.
  - If your application performs calculations, or does not expect `NULL` values, then you should use the default behavior of column-level VPD, which is specified with the `sec_relevant_cols` parameter.
  - If you use `UPDATE AS SELECT` with this option, then only the values in the columns you are allowed to see will be updated.
  - This option may prevent some rows from displaying. For example:

```
select * from employees
where salary = 10
```

This query may not return rows if the `salary` column returns a `NULL` value because the column masking option has been set.

- When you add a VPD policy to a synonym, it causes all the dependent objects of the synonym, including policy functions that reference the synonym, to be marked `INVALID`.

## Examples

As the first of two examples, the following creates a policy that applies to the `hr.employee` table. This is a column-level VPD policy that will be enforced only if a `SELECT` or an `INDEX` statement refers to the `salary`, `birthdate`, or `SSN` columns of the table explicitly, or implicitly through a view. It is also a `CONTEXT_SENSITIVE` policy, so the server will invoke the policy function `hr.hrfun` at parse time. During execution, it will only invoke the function if there has been any session private context change since the last use of the statement cursor. The predicate generated by the policy function must not exceed 4000 bytes, the default length limit, since the `long_predicate` parameter is omitted from the call.

```
BEGIN
dbms_ols.add_policy(object_schema => 'hr',
object_name => 'employee',
policy_name => 'hr_policy',
function_schema => 'hr',
policy_function => 'hrfun',
statement_types => 'select,index',
policy_type => dbms_ols.CONTEXT_SENSITIVE,
sec_relevant_cols=>'salary,birthdate,ssn');
END;
/
```

As the second example, the following command creates another policy that applies to the same object for hosting, so users can access only data based on their subscriber ID. Since it is defined as a `SHARED_STATIC` policy type, the server will first try to find the predicate in the SGA cache. The server will only invoke the policy function, `subfun`, if that search fails.

```
BEGIN
dbms_ols.add_policy(object_schema => 'hr',
object_name => 'employee',
policy_name => 'hosting_policy',
function_schema => 'hr',
policy_function => 'subfun',
policy_type => dbms_ols.SHARED_STATIC);
END;
/
```

## ADD\_POLICY\_CONTEXT Procedure

This procedure adds the context for the active application.

### Syntax

```
DBMS_RLS.ADD_POLICY_CONTEXT (
  object_schema  IN VARCHAR2 NULL,
  object_name    IN VARCHAR2,
  namespace     IN VARCHAR2,
  attribute      IN VARCHAR2);
```

### Parameters

**Table 124–5 ADD\_POLICY\_CONTEXT Procedure Parameters**

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
namespace	The namespace of the driving context
attribute	The attribute of the driving context.

### Usage Notes

Note the following:

- This procedure indicates the application context that drives the enforcement of policies; this is the context that determines which application is running.
- If no object\_schema is specified, the current user's schema is assumed.
- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is NULL, policies from all policy groups are used.
- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the SYS\_DEFAULT policy group.
- To add a policy to table hr.employees in group access\_control\_group, the following command is issued:

```
DBMS_RLS.ADD_GROUPED_POLICY('hr','employees','access_control_
group','policy1','SYS','HR.ACCESS');
```

## CREATE\_POLICY\_GROUP Procedure

This procedure creates a policy group.

### Syntax

```
DBMS_RLS.CREATE_POLICY_GROUP (  
    object_schema    IN VARCHAR2 NULL,  
    object_name      IN VARCHAR2,  
    policy_group     IN VARCHAR2);
```

### Parameters

**Table 124–6 CREATE\_POLICY\_GROUP Procedure Parameters**

Parameter	Description
<code>object_schema</code>	Schema containing the table, view, or synonym.
<code>object_name</code>	Name of the table, view, or synonym to which the policy is added.
<code>policy_group</code>	Name of the policy group that the policy belongs to.

### Usage Notes

The group must be unique for each table or view.

## DELETE\_POLICY\_GROUP Procedure

This procedure deletes a policy group.

### Syntax

```
DBMS_RLS.DELETE_POLICY_GROUP (  
  object_schema  IN VARCHAR2 NULL,  
  object_name    IN VARCHAR2,  
  policy_group   IN VARCHAR2);
```

### Parameters

**Table 124–7** *DELETE\_POLICY\_GROUP Procedure Parameters*

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is added.
policy_group	The name of the policy group that the policy belongs to.

### Usage Notes

Note the following:

- This procedure deletes a policy group for the specified table, view, or synonym.
- No policy can be in the policy group.

## DISABLE\_GROUPED\_POLICY Procedure

This procedure disables a row-level group security policy.

### Syntax

```
DBMS_RLS.DISABLE_GROUPED_POLICY (  
    object_schema    IN VARCHAR2 NULL,  
    object_name      IN VARCHAR2,  
    group_name       IN VARCHAR2,  
    policy_name      IN VARCHAR2);
```

### Parameters

**Table 124–8** *ENABLE\_GROUPED\_POLICY Procedure Parameters*

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym with which the policy is associated.
group_name	The name of the group of the policy.
policy_name	The name of the policy to be enabled or disabled.

### Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is disabled when this procedure is executed or when the ENABLE\_GROUPED\_POLICY procedure is executed with "enable" set to FALSE.

## DROP\_GROUPED\_POLICY Procedure

This procedure drops a policy associated with a policy group.

### Syntax

```
DBMS_RLS.DROP_GROUPED_POLICY (  
    object_schema  IN VARCHAR2 NULL,  
    object_name    IN VARCHAR2,  
    policy_group   IN VARCHAR2 'SYS_DEFAULT',  
    policy_name    IN VARCHAR2);
```

### Parameters

**Table 124–9** *DROP\_GROUPED\_POLICY Procedure Parameters*

Parameter	Description
object_schema	The schema containing the table, view, or synonym.
object_name	The name of the table, view, or synonym to which the policy is dropped.
policy_group	The name of the policy group that the policy belongs to.
policy_name	The name of the policy.

## DROP\_POLICY Procedure

This procedure drops a fine-grained access control policy from a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

**See Also:** [Operational Notes](#) on page 124-5

A COMMIT is also performed at the end of the operation.

### Syntax

```
DBMS_RLS.DROP_POLICY (
  object_schema  IN VARCHAR2 NULL,
  object_name    IN VARCHAR2,
  policy_name    IN VARCHAR2);
```

### Parameters

**Table 124–10 DROP\_GROUPED\_POLICY Procedure Parameters**

Parameter	Description
object_schema	Schema containing the table, view or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.
object_name	Name of the table, view, or synonym for which the policy is dropped.
policy_name	Name of policy to be dropped from table, view, or synonym.

### Usage Notes

- When you drop a VPD policy from a synonym, it causes all the dependent objects of the synonym, including policy functions that reference the synonym, to be marked INVALID.



## DROP\_POLICY\_CONTEXT Procedure

This procedure drops a driving context from the object so that it will have one less driving context.

### Syntax

```
DBMS_RLS.DROP_POLICY_CONTEXT (  
    object_schema IN VARCHAR2 NULL,  
    object_name   IN VARCHAR2,  
    namespace    IN VARCHAR2,  
    attribute     IN VARCHAR2);
```

### Parameters

**Table 124–11** *DROP\_POLICY\_CONTEXT Procedure Parameters*

Parameter	Description
object_schema	The schema containing the table, view, or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.
object_name	The name of the table, view, or synonym to which the policy is dropped.
namespace	The namespace of the driving context.
attribute	The attribute of the driving context.

## ENABLE\_GROUPED\_POLICY Procedure

This procedure enables or disables a row-level group security policy.

### Syntax

```
DBMS_RLS.ENABLE_GROUPED_POLICY (
  object_schema  IN VARCHAR2 NULL,
  object_name    IN VARCHAR2,
  group_name     IN VARCHAR2,
  policy_name    IN VARCHAR2,
  enable        IN BOOLEAN TRUE);
```

### Parameters

**Table 124–12** *ENABLE\_GROUPED\_POLICY Procedure Parameters*

Parameter	Description
object_schema	The schema containing the table, view, or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.
object_name	The name of the table, view, or synonym with which the policy is associated.
group_name	The name of the group of the policy.
policy_name	The name of the policy to be enabled or disabled.
enable	TRUE enables the policy; FALSE disables the policy.

### Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is enabled when it is created.

## ENABLE\_POLICY Procedure

This procedure enables or disables a fine-grained access control policy. A policy is enabled when it is created.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

**See Also:** [Operational Notes](#) on page 124-5

A COMMIT is also performed at the end of the operation.

### Syntax

```
DBMS_RLS.ENABLE_POLICY (
  object_schema IN VARCHAR2 NULL,
  object_name   IN VARCHAR2,
  policy_name   IN VARCHAR2,
  enable       IN BOOLEAN TRUE);
```

### Parameters

**Table 124–13** *ENABLE\_POLICY Procedure Parameters*

Parameter	Description
object_schema	Schema containing table, view, or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.
object_name	Name of table, view, or synonym with which the policy is associated.
policy_name	Name of policy to be enabled or disabled.
enable	TRUE to enable the policy, FALSE to disable the policy.

## REFRESH\_GROUPED\_POLICY Procedure

This procedure reparses the SQL statements associated with a refreshed policy.

### Syntax

```
DBMS_RLS.REFRESH_GROUPED_POLICY (
  object_schema  IN VARCHAR2 NULL,
  object_name    IN VARCHAR2 NULL,
  group_name     IN VARCHAR2 NULL,
  policy_name    IN VARCHAR2 NULL);
```

### Parameters

**Table 124–14 REFRESH\_GROUPED\_POLICY Procedure Parameters**

Parameter	Description
object_schema	The schema containing the table, view, or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.
object_name	The name of the table, view, or synonym with which the policy is associated.
group_name	The name of the group of the policy.
policy_name	The name of the policy.

### Usage Notes

- This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to the policy has immediate effect after the procedure is executed.
- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- The procedure returns an error if it tries to refresh a disabled policy.

## REFRESH\_POLICY Procedure

This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to this policy will have immediate effect after the procedure is executed.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

**See Also:** [Operational Notes](#) on page 124-5

A COMMIT is also performed at the end of the operation.

### Syntax

```
DBMS_RLS.REFRESH_POLICY (
  object_schema IN VARCHAR2 NULL,
  object_name   IN VARCHAR2 NULL,
  policy_name   IN VARCHAR2 NULL);
```

### Parameters

**Table 124–15 REFRESH\_POLICY Procedure Parameters**

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_schema is specified, or NULL is provided, then the current user's schema is assumed.
object_name	Name of table, view, or synonym with which the policy is associated.
policy_name	Name of policy to be refreshed.

### Usage Notes

The procedure returns an error if it tries to refresh a disabled policy.



The DBMS\_ROWID package lets you create ROWIDs and obtain information about ROWIDs from PL/SQL programs and SQL statements. You can find the data block number, the object number, and other ROWID components without writing code to interpret the base-64 character external ROWID. DBMS\_ROWID is intended for upgrading from Oracle database version 7 to Oracle database version 8.X.

---

---

**Note:** DBMS\_ROWID is not to be used with universal ROWIDs (UROWIDs).

---

---

This chapter contains the following topics:

- [Using DBMS\\_ROWID](#)
  - Security Model
  - Types
  - Exceptions
  - Operational Notes
  - Examples
- [Summary of DBMS\\_ROWID Subprograms](#)

## Using DBMS\_ROWID

- [Security Model](#)
- [Types](#)
- [Exceptions](#)
- [Operational Notes](#)
- [Examples](#)



## Security Model

This package runs with the privileges of calling user, rather than the package owner SYS.

## Types

- [Extension and Restriction Types](#)
- [Verification Types](#)
- [Object Types](#)
- [Conversion Types](#)

### Extension and Restriction Types

The types are as follows:

- RESTRICTED—restricted ROWID
- EXTENDED—extended ROWID

For example:

```
rowid_type_restricted constant integer := 0;
rowid_type_extended   constant integer := 1;
```

---



---

**Note:** Extended ROWIDs are only used in Oracle database version 8.Xi and higher.

---



---

### Verification Types

**Table 125–1** *Verification Types*

Result	Description
VALID	Valid ROWID
INVALID	Invalid ROWID

For example:

```
rowid_is_valid   constant integer := 0;
rowid_is_invalid constant integer := 1;
```

### Object Types

**Table 125–2** *Object Types*

Result	Description
UNDEFINED	Object Number not defined (for restricted ROWIDs)

For example:

```
rowid_object_undefined constant integer := 0;
```

### Conversion Types

**Table 125–3** *Conversion Types*

Result	Description
INTERNAL	Convert to/from column of ROWID type

**Table 125-3 (Cont.) Conversion Types**

<b>Result</b>	<b>Description</b>
EXTERNAL	Convert to/from string format

For example:

```
rowid_convert_internal constant integer := 0;  
rowid_convert_external constant integer := 1;
```

## Exceptions

**Table 125-4** *Exceptions*

<b>Exception</b>	<b>Description</b>
ROWID_INVALID	Invalid rowid format
ROWID_BAD_BLOCK	Block is beyond end of file

For example:

```
ROWID_INVALID exception;  
  pragma exception_init(ROWID_INVALID, -1410);
```

```
ROWID_BAD_BLOCK exception;  
  pragma exception_init(ROWID_BAD_BLOCK, -28516);
```

## Operational Notes

- Some of the functions in this package take a single parameter, such as a ROWID. This can be a character or a PL/SQL ROWID, either restricted or extended, as required.
- You can call the DBMS\_ROWID functions and procedures from PL/SQL code, and you can also use the functions in SQL statements.

---

---

**Note:** ROWID\_INFO is a procedure. It can only be used in PL/SQL code.

---

---

- You can use functions from the DBMS\_ROWID package just like built-in SQL functions; in other words, you can use them wherever you can use an expression. In this example, the ROWID\_BLOCK\_NUMBER function is used to return just the block number of a single row in the EMP table:

```
SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid)
       FROM emp
       WHERE ename = 'KING';
```

- If Oracle returns the error "ORA:452, 0, 'Subprogram '%s' violates its associated pragma' for pragma restrict\_references, it could mean the violation is due to:
  - A problem with the current procedure or function
  - Calling a procedure or function without a pragma or due to calling one with a less restrictive pragma
  - Calling a package procedure or function that touches the initialization code in a package or that sets the default values

## Examples

This example returns the ROWID for a row in the EMP table, extracts the data object number from the ROWID, using the ROWID\_OBJECT function in the DBMS\_ROWID package, then displays the object number:

```
DECLARE
  object_no  INTEGER;
  row_id     ROWID;
  ...
BEGIN
  SELECT ROWID INTO row_id FROM emp
     WHERE empno = 7499;
  object_no := DBMS_ROWID.ROWID_OBJECT(row_id);
  DBMS_OUTPUT.PUT_LINE('The obj. # is '|| object_no);
  ...
```

---

## Summary of DBMS\_ROWID Subprograms

**Table 125–5 DBMS\_ROWID Package Subprograms**

Subprogram	Description
<a href="#">ROWID_BLOCK_NUMBER Function</a> on page 125-10	Returns the block number of a ROWID
<a href="#">ROWID_CREATE Function</a> on page 125-11	Creates a ROWID, for testing only
<a href="#">ROWID_INFO Procedure</a> on page 125-12	Returns the type and components of a ROWID
<a href="#">ROWID_OBJECT Function</a> on page 125-13	Returns the object number of the extended ROWID
<a href="#">ROWID_RELATIVE_FNO Function</a> on page 125-14	Returns the file number of a ROWID
<a href="#">ROWID_ROW_NUMBER Function</a> on page 125-15	Returns the row number
<a href="#">ROWID_TO_ABSOLUTE_FNO Function</a> on page 125-16	Returns the absolute file number associated with the ROWID for a row in a specific table
<a href="#">ROWID_TO_EXTENDED Function</a> on page 125-17	Converts a ROWID from restricted format to extended
<a href="#">ROWID_TO_RESTRICTED Function</a> on page 125-19	Converts an extended ROWID to restricted format
<a href="#">ROWID_TYPE Function</a> on page 125-20	Returns the ROWID type: 0 is restricted, 1 is extended
<a href="#">ROWID_VERIFY Function</a> on page 125-21	Checks if a ROWID can be correctly extended by the ROWID_TO_EXTENDED function

## ROWID\_BLOCK\_NUMBER Function

This function returns the database block number for the input ROWID.

### Syntax

```
DBMS_ROWID.ROWID_BLOCK_NUMBER (  
    row_id      IN    ROWID,  
    ts_type_in  IN    VARCHAR2 DEFAULT 'SMALLFILE')  
RETURN NUMBER;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_block_number,WNDS,RNDS,WNPS,RNPS);
```

### Parameters

**Table 125–6** ROWID\_BLOCK\_NUMBER Function Parameters

Parameter	Description
row_id	ROWID to be interpreted
ts_type_in	The type of the tablespace (bigfile/smallfile) to which the row belongs

### Examples

The example SQL statement selects the block number from a ROWID and inserts it into another table:

```
INSERT INTO T2 (SELECT dbms_rowid.rowid_block_number(ROWID, 'BIGFILE')  
    FROM some_table  
    WHERE key_value = 42);
```



## ROWID\_CREATE Function

This function lets you create a ROWID, given the component parts as parameters.

This is useful for testing ROWID operations, because only the Oracle Server can create a valid ROWID that points to data in a database.

### Syntax

```
DBMS_ROWID.ROWID_CREATE (
  rowid_type    IN NUMBER,
  object_number IN NUMBER,
  relative_fno  IN NUMBER,
  block_number  IN NUMBER,
  row_number    IN NUMBER)
RETURN ROWID;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_create,WNDS,RNDS,WNPS,RNPS);
```

### Parameters

**Table 125–7 ROWID\_CREATE Function Parameters**

Parameter	Description
rowid_type	Type (restricted or extended) Set the rowid_type parameter to 0 for a restricted ROWID. Set it to 1 to create an extended ROWID. If you specify rowid_type as 0, then the required object_number parameter is ignored, and ROWID_CREATE returns a restricted ROWID.
object_number	Data object number (rowid_object_undefined for restricted)
relative_fno	Relative file number
block_number	Block number in this file
row_number	Returns row number in this block

### Examples

Create a dummy extended ROWID:

```
my_rowid := DBMS_ROWID.ROWID_CREATE(1, 9999, 12, 1000, 13);
```

Find out what the rowid\_object function returns:

```
obj_number := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

The variable obj\_number now contains 9999.

## ROWID\_INFO Procedure

This procedure returns information about a ROWID, including its type (restricted or extended), and the components of the ROWID. This is a procedure, and it cannot be used in a SQL statement.

### Syntax

```
DBMS_ROWID.ROWID_INFO (
  rowid_in      IN  ROWID,
  rowid_type    OUT NUMBER,
  object_number OUT NUMBER,
  relative_fno  OUT NUMBER,
  block_number  OUT NUMBER,
  row_number    OUT NUMBER);
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_info,WNDS,RNDS,WNPS,RNPS);
```

### Parameters

**Table 125–8 ROWID\_INFO Procedure Parameters**

Parameter	Description
rowid_in	ROWID to be interpreted. This determines if the ROWID is a restricted (0) or extended (1) ROWID.
rowid_type	Returns type (restricted/extended)
object_number	Returns data object number (rowid_object_undefined for restricted)
relative_fno	Returns relative file number
block_number	Returns block number in this file
row_number	Returns row number in this block

**See Also:** ["ROWID\\_TYPE Function"](#) on page 125-20

### Examples

This example reads back the values for the ROWID that you created in the ROWID\_CREATE:

```
DBMS_ROWID.ROWID_INFO (
  my_rowid, rid_type, obj_num, file_num, block_num, row_num, 'BIGFILE');
```

## ROWID\_OBJECT Function

This function returns the data object number for an extended ROWID. The function returns zero if the input ROWID is a restricted ROWID.

### Syntax

```
DBMS_ROWID.ROWID_OBJECT (
  rowid_id IN ROWID)
RETURN NUMBER;
```

### Pragmas

```
pragma RESTRICT_REFERENCES (rowid_object, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 125–9** ROWID\_OBJECT Function Parameters

Parameter	Description
row_id	ROWID to be interpreted

---

**Note:** The ROWID\_OBJECT\_UNDEFINED constant is returned for restricted ROWIDs.

---

### Examples

```
SELECT dbms_rowid.rowid_object(ROWID)
FROM emp
WHERE empno = 7499;
```

## ROWID\_RELATIVE\_FNO Function

This function returns the relative file number of the ROWID specified as the IN parameter. (The file number is relative to the tablespace.)

### Syntax

```
DBMS_ROWID.ROWID_RELATIVE_FNO (
    rowid_id      IN   ROWID,
    ts_type_in    IN   VARCHAR2 DEFAULT 'SMALLFILE')
RETURN NUMBER;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_relative_fno,WNDS,RNDS,WNPS,RNPS);
```

### Parameters

**Table 125–10 ROWID\_RELATIVE\_FNO Function Parameters**

Parameter	Description
row_id	ROWID to be interpreted
ts_type_in	Type of the tablespace (bigfile/smallfile) to which the row belongs

### Examples

The example PL/SQL code fragment returns the relative file number:

```
DECLARE
    file_number    INTEGER;
    rowid_val      ROWID;
BEGIN
    SELECT ROWID INTO rowid_val
        FROM dept
        WHERE loc = 'Boston';
    file_number :=
        dbms_rowid.rowid_relative_fno(rowid_val, 'SMALLFILE');
    ...
```

## ROWID\_ROW\_NUMBER Function

This function extracts the row number from the ROWID IN parameter.

### Syntax

```
DBMS_ROWID.ROWID_ROW_NUMBER (  
    row_id IN ROWID)  
RETURN NUMBER;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES (rowid_row_number, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 125–11** ROWID\_ROW\_NUMBER Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

### Examples

Select a row number:

```
SELECT dbms_rowid.rowid_row_number(ROWID)  
FROM emp  
WHERE ename = 'ALLEN';
```

## ROWID\_TO\_ABSOLUTE\_FNO Function

This function extracts the absolute file number from a ROWID, where the file number is absolute for a row in a given schema and table. The schema name and the name of the schema object (such as a table name) are provided as IN parameters for this function.

### Syntax

```
DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO (  
    row_id      IN ROWID,  
    schema_name IN VARCHAR2,  
    object_name IN VARCHAR2)  
RETURN NUMBER;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_absolute_fno,WNDS,WNPS,RNPS);
```

### Parameters

**Table 125–12** ROWID\_TO\_ABSOLUTE\_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted
schema_name	Name of the schema which contains the table
object_name	Table name

### Examples

```
DECLARE  
    abs_fno      INTEGER;  
    rowid_val    CHAR(18);  
    object_name  VARCHAR2(20) := 'EMP';  
BEGIN  
    SELECT ROWID INTO rowid_val  
    FROM emp  
    WHERE empno = 9999;  
    abs_fno := dbms_rowid.rowid_to_absolute_fno(  
        rowid_val, 'SCOTT', object_name);
```

---

---

**Note:** For partitioned objects, the name must be a table name, not a partition or a sub/partition name.

---

---

## ROWID\_TO\_EXTENDED Function

This function translates a restricted ROWID that addresses a row in a schema and table that you specify to the extended ROWID format. Later, it may be removed from this package into a different place.

### Syntax

```
DBMS_ROWID.ROWID_TO_EXTENDED (
  old_rowid      IN ROWID,
  schema_name    IN VARCHAR2,
  object_name    IN VARCHAR2,
  conversion_type IN INTEGER)
RETURN ROWID;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_extended,WNDS,WNPS,RNPS);
```

### Parameters

**Table 125–13 ROWID\_TO\_EXTENDED Function Parameters**

Parameter	Description
old_rowid	ROWID to be converted
schema_name	Name of the schema which contains the table (optional)
object_name	Table name (optional).
conversion_type	The following constants are defined: ROWID_CONVERT_INTERNAL (:=0) ROWID_CONVERT_EXTERNAL (:=1)

### Return Values

ROWID\_TO\_EXTENDED returns the ROWID in the extended character format. If the input ROWID is NULL, then the function returns NULL. If a zero-valued ROWID is supplied (00000000.0000.0000), then a zero-valued restricted ROWID is returned.

### Examples

Assume that there is a table called RIDS in the schema SCOTT, and that the table contains a column ROWID\_COL that holds ROWIDs (restricted), and a column TABLE\_COL that point to other tables in the SCOTT schema. You can convert the ROWIDs to extended format with the statement:

```
UPDATE SCOTT.RIDS
SET rowid_col =
  dbms_rowid.rowid_to_extended (
    rowid_col, 'SCOTT', TABLE_COL, 0);
```

### Usage Notes

- If the schema and object names are provided as IN parameters, then this function verifies SELECT authority on the table named, and converts the restricted ROWID provided to an extended ROWID, using the data object number of the table. That ROWID\_TO\_EXTENDED returns a value, however, does not guarantee that the

converted ROWID actually references a valid row in the table, either at the time that the function is called, or when the extended ROWID is actually used.

- If the schema and object name are not provided (are passed as NULL), then this function attempts to fetch the page specified by the restricted ROWID provided. It treats the file number stored in this ROWID as the absolute file number. This can cause problems if the file has been dropped, and its number has been reused prior to the migration. If the fetched page belongs to a valid table, then the data object number of this table is used in converting to an extended ROWID value. This is very inefficient, and Oracle recommends doing this only as a last resort, when the target table is not known. The user must still know the correct table name at the time of using the converted value.
- If an extended ROWID value is supplied, the data object number in the input extended ROWID is verified against the data object number computed from the table name parameter. If the two numbers do not match, the `INVALID_ROWID` exception is raised. If they do match, the input ROWID is returned.
- `ROWID_TO_EXTENDED` cannot be used with partition tables.

**See Also:** The [ROWID\\_VERIFY Function](#) has a method to determine if a given ROWID can be converted to the extended format.



## ROWID\_TO\_RESTRICTED Function

This function converts an extended ROWID into restricted ROWID format.

### Syntax

```
DBMS_ROWID.ROWID_TO_RESTRICTED (
    old_rowid      IN ROWID,
    conversion_type IN INTEGER)
RETURN ROWID;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_restricted,WNDS,RNDS,WNPS,RNPS);
```

### Parameters

**Table 125–14 ROWID\_TO\_RESTRICTED Function Parameters**

Parameter	Description
old_rowid	ROWID to be converted
conversion_type	The following constants are defined: ROWID_CONVERT_INTERNAL (:=0) ROWID_CONVERT_EXTERNAL (:=1)

## ROWID\_TYPE Function

This function returns 0 if the ROWID is a restricted ROWID, and 1 if it is extended.

### Syntax

```
DBMS_ROWID.ROWID_TYPE (  
    rowid_id IN ROWID)  
RETURN NUMBER;
```

### Pragmas

```
pragma RESTRICT_REFERENCES (rowid_type, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 125–15** ROWID\_TYPE Function Parameters

Parameter	Description
row_id	ROWID to be interpreted

### Examples

```
IF DBMS_ROWID.ROWID_TYPE(my_rowid) = 1 THEN  
    my_obj_num := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

## ROWID\_VERIFY Function

This function verifies the ROWID. It returns 0 if the input restricted ROWID can be converted to extended format, given the input schema name and table name, and it returns 1 if the conversion is not possible.

---



---

**Note:** You can use this function in a WHERE clause of a SQL statement, as shown in the example.

---



---

### Syntax

```
DBMS_ROWID.ROWID_VERIFY (
    rowid_in          IN ROWID,
    schema_name       IN VARCHAR2,
    object_name       IN VARCHAR2,
    conversion_type   IN INTEGER
) RETURN NUMBER;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(rowid_verify,WNDS,WNPS,RNPS);
```

### Parameters

**Table 125–16** ROWID\_VERIFY Function Parameters

Parameter	Description
rowid_in	ROWID to be verified
schema_name	Name of the schema which contains the table
object_name	Table name
conversion_type	The following constants are defined: ROWID_CONVERT_INTERNAL (:=0) ROWID_CONVERT_EXTERNAL (:=1)

### Examples

Considering the schema in the example for the ROWID\_TO\_EXTENDED function, you can use the following statement to find bad ROWIDs prior to conversion. This enables you to fix them beforehand.

```
SELECT ROWID, rowid_col
   FROM SCOTT.RIDS
  WHERE dbms_rowid.rowid_verify(rowid_col, NULL, NULL, 0) =1;
```

**See Also:** [Chapter 232, "UTL\\_RAW"](#), [Chapter 234, "UTL\\_REF"](#)



The DBMS\_RULE package contains subprograms that enable the evaluation of a rule set for a specified event.

This chapter contains the following topics:

- [Using DBMS\\_RULE](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_RULE Subprograms](#)

---

## Using DBMS\_RULE

This section contains topics which relate to using the DBMS\_RULE package.

- [Overview](#)
- [Security Model](#)

## Overview

This package contains subprograms that enable the evaluation of a rule set for a specified event.

**See Also:**

- [Chapter 256, "Rule TYPES"](#) for more information about the types used with the DBMS\_RULE package
- [Chapter 127, "DBMS\\_RULE\\_ADM"](#) and *Oracle Streams Concepts and Administration* for more information about this package and rules

## Security Model

PUBLIC is granted EXECUTE privilege on this package.

**See Also:** *Oracle Database Security Guide* for more information about user group PUBLIC



---

## Summary of DBMS\_RULE Subprograms

**Table 126–1 DBMS\_RULE Package Subprograms**

Subprogram	Description
<a href="#">CLOSE_ITERATOR Procedure</a> on page 126-6	Closes an open iterator
<a href="#">EVALUATE Procedures</a> on page 126-7	Evaluates the rules in the specified rule set that use the evaluation context specified
<a href="#">GET_NEXT_HIT Function</a> on page 126-11	Returns the next rule that evaluated to TRUE from a true rules iterator, or returns the next rule that evaluated to MAYBE from a maybe rules iterator; returns NULL if there are no more rules that evaluated to TRUE or MAYBE.

## CLOSE\_ITERATOR Procedure

This procedure closes an open iterator.

### Syntax

```
DBMS_RULE.CLOSE_ITERATOR(  
    iterator IN BINARY_INTEGER);
```

### Parameter

**Table 126–2** *CLOSE\_ITERATOR Procedure Parameter*

Parameter	Description
<code>iterator</code>	The iterator to be closed

### Usage Notes

This procedure requires an open iterator that was returned by an earlier call to `DBMS_RULE.EVALUATE` in the same session. The user who runs this procedure does not require any privileges on the rule set being evaluated.

Closing an iterator frees resources, such as memory, associated with the iterator. Therefore, Oracle recommends that you close an iterator when it is no longer needed.

**See Also:** [EVALUATE Procedures](#) on page 126-7

## EVALUATE Procedures

This procedure evaluates the rules in the specified rule set that use the evaluation context specified for a specified event.

This procedure is overloaded. The `true_rules` and `maybe_rules` parameters are mutually exclusive with the `true_rules_iterator` and `maybe_rules_iterator` parameters. In addition, the procedure with the `true_rules` and `maybe_rules` parameters includes the `stop_on_first_hit` parameter, but the other procedure does not.

### Syntax

```
DBMS_RULE.EVALUATE (
  rule_set_name      IN  VARCHAR2,
  evaluation_context IN  VARCHAR2,
  event_context      IN  SYS.RE$NV_LIST           DEFAULT NULL,
  table_values       IN  SYS.RE$TABLE_VALUE_LIST  DEFAULT NULL,
  column_values      IN  SYS.RE$COLUMN_VALUE_LIST DEFAULT NULL,
  variable_values    IN  SYS.RE$VARIABLE_VALUE_LIST DEFAULT NULL,
  attribute_values   IN  SYS.RE$ATTRIBUTE_VALUE_LIST DEFAULT NULL,
  stop_on_first_hit  IN  BOOLEAN                 DEFAULT FALSE,
  simple_rules_only  IN  BOOLEAN                 DEFAULT FALSE,
  true_rules         OUT SYS.RE$RULE_HIT_LIST,
  maybe_rules        OUT SYS.RE$RULE_HIT_LIST);
```

```
DBMS_RULE.EVALUATE (
  rule_set_name      IN  VARCHAR2,
  evaluation_context IN  VARCHAR2,
  event_context      IN  SYS.RE$NV_LIST           DEFAULT NULL,
  table_values       IN  SYS.RE$TABLE_VALUE_LIST  DEFAULT NULL,
  column_values      IN  SYS.RE$COLUMN_VALUE_LIST DEFAULT NULL,
  variable_values    IN  SYS.RE$VARIABLE_VALUE_LIST DEFAULT NULL,
  attribute_values   IN  SYS.RE$ATTRIBUTE_VALUE_LIST DEFAULT NULL,
  simple_rules_only  IN  BOOLEAN                 DEFAULT FALSE,
  true_rules_iterator OUT BINARY_INTEGER,
  maybe_rules_iterator OUT BINARY_INTEGER);
```

### Parameters

**Table 126–3 EVALUATE Procedure Parameters**

Parameter	Description
<code>rule_set_name</code>	Name of the rule set in the form [ <i>schema_name.</i> ] <i>rule_set_name</i> . For example, to evaluate all of the rules in a rule set named <code>hr_rules</code> in the <code>hr</code> schema, enter <code>hr.hr_rules</code> for this parameter. If the schema is not specified, then the schema of the current user is used.
<code>evaluation_context</code>	An evaluation context name in the form [ <i>schema_name.</i> ] <i>evaluation_context_name</i> . If the schema is not specified, then the name of the current user is used.  Only rules that use the specified evaluation context are evaluated.
<code>event_context</code>	A list of name-value pairs that identify events that cause evaluation

**Table 126–3 (Cont.) EVALUATE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>table_values</code>	Contains the data for table rows using the table aliases specified when the evaluation context was created. Each table alias in the list must be unique.
<code>column_values</code>	Contains the partial data for table rows. It must not contain column values for tables, whose values are already specified in <code>table_values</code> .
<code>variable_values</code>	A list containing the data for variables.  The only way for an explicit variable value to be known is to specify its value in this list.  If an implicit variable value is not specified in the list, then the function used to obtain the value of the implicit variable is invoked. If an implicit variable value is specified in the list, then this value is used and the function is not invoked.
<code>attribute_values</code>	Contains the partial data for variables. It must not contain attribute values for variables whose values are already specified in <code>variable_values</code> .
<code>stop_on_first_hit</code>	If <code>TRUE</code> , then the rules engine stops evaluation as soon as it finds a <code>TRUE</code> rule.  If <code>TRUE</code> and there are no <code>TRUE</code> rules, then the rules engine stops evaluation as soon as it finds a rule that may evaluate to <code>TRUE</code> given more data.  If <code>FALSE</code> , then the rules engine continues to evaluate rules even after it finds a <code>TRUE</code> rule.
<code>simple_rules_only</code>	If <code>TRUE</code> , then only those rules that are simple enough to be evaluated fast (without issuing SQL) are considered for evaluation.  If <code>FALSE</code> , then evaluates all rules.
<code>true_rules</code>	Receives the output of the <code>EVALUATE</code> procedure into a varray of <code>RE\$RULE_HIT_LIST</code> type.  If no rules evaluate to <code>TRUE</code> , then <code>true_rules</code> is empty.  If at least one rule evaluates to <code>TRUE</code> and <code>stop_on_first_hit</code> is <code>TRUE</code> , then <code>true_rules</code> contains one rule that evaluates to <code>TRUE</code> .  If <code>stop_on_first_hit</code> is <code>FALSE</code> , then <code>true_rules</code> contains all rules that evaluate to <code>TRUE</code> .
<code>maybe_rules</code>	If all rules can be evaluated completely, without requiring any additional data, then <code>maybe_rules</code> is empty.  If <code>stop_on_first_hit</code> is <code>TRUE</code> , then if there is at least one rule that may evaluate to <code>TRUE</code> given more data, and no rules evaluate to <code>TRUE</code> , then <code>maybe_rules</code> contains one rule that may evaluate to <code>TRUE</code> .  If <code>stop_on_first_hit</code> is <code>FALSE</code> , then <code>maybe_rules</code> contains all rules that may evaluate to <code>TRUE</code> given more data.
<code>true_rules_iterator</code>	Contains the iterator for accessing rules that are <code>TRUE</code>
<code>maybe_rules_iterator</code>	Contains the iterator for accessing rules that may be <code>TRUE</code> given additional data or the ability to issue SQL

## Usage Notes

---

**Note:** Rules in the rule set that use an evaluation context different from the one specified are not considered for evaluation.

---

The rules in the rule set are evaluated using the data specified for `table_values`, `column_values`, `variable_values`, and `attribute_values`. These values must refer to tables and variables in the specified evaluation context. Otherwise, an error is raised.

The caller may specify, using `stop_on_first_hit`, if evaluation must stop as soon as the first TRUE rule or the first MAYBE rule (if there are no TRUE rules) is found.

The caller may also specify, using `simple_rules_only`, if only rules that are simple enough to be evaluated fast (which means without SQL) should be considered for evaluation. This makes evaluation faster, but causes rules that cannot be evaluated without SQL to be returned as MAYBE rules.

Partial evaluation is supported. The `EVALUATE` procedure can be called with data for only some of the tables, columns, variables, or attributes. In such a case, rules that cannot be evaluated because of a lack of data are returned as MAYBE rules, unless they can be determined to be TRUE or FALSE based on the values of one or more simple expressions within the rule. For example, given a value of 1 for attribute "a.b" of variable "x", a rule with the following rule condition can be returned as TRUE, without a value for table "tab":

```
(:x.a.b = 1) or (tab.c > 10)
```

The results of an evaluation are the following:

- TRUE rules, which is the list of rules that evaluate to TRUE based on the given data. These rules are returned either in the OUT parameter `true_rules`, which returns all of the rules that evaluate to TRUE, or in the OUT parameter `true_rules_iterator`, which returns each rule that evaluates to TRUE one at a time.
- MAYBE rules, which is the list of rules that could not be evaluated for one of the following reasons:
  - The rule refers to data that was unavailable. For example, a variable attribute "x.a.b" is specified, but no value is specified for the variable "x", the attribute "a", or the attribute "a.b".
  - The rule is not simple enough to be evaluated fast (without SQL) and `simple_rules_only` is specified as TRUE, or partial data is available.

Maybe rules are returned either in the OUT parameter `maybe_rules`, which returns all of the rules that evaluate to MAYBE, or in the OUT parameter `maybe_rules_iterator`, which returns each rule that evaluates to MAYBE one at a time.

The caller may specify whether the procedure returns all of the rules that evaluate to TRUE and MAYBE for the event or an iterator for rules that evaluate to TRUE and MAYBE. A true rules iterator enables the client to fetch each rule that evaluates to TRUE one at a time, and a maybe rules iterator enables the client to fetch each rule that evaluates to MAYBE one at a time.

If you use an iterator, then you use the `GET_NEXT_HIT` function in the `DBMS_RULE` package to retrieve the next rule that evaluates to TRUE or MAYBE from an iterator. Oracle recommends that you close an iterator if it is no longer needed to free

resources, such as memory, used by the iterator. An iterator can be closed in the following ways:

- The `CLOSE_ITERATOR` procedure in the `DBMS_RULE` package is run with the iterator specified.
- The iterator returns `NULL` because no more rules evaluate to `TRUE` or `MAYBE`.
- The session in which the iterator is running ends.

To run the `DBMS_RULE.EVALUATE` procedure, a user must meet at least one of the following requirements:

- Have `EXECUTE_ON_RULE_SET` privilege on the rule set
- Have `EXECUTE_ANY_RULE_SET` system privilege
- Be the rule set owner

---

---

**Note:** The rules engine does not invoke any actions. An action context can be returned with each returned rule, but the client of the rules engine must invoke any necessary actions.

---

---

**See Also:**

- [Chapter 256, "Rule TYPES"](#) for more information about the types used with the `DBMS_RULE` package
- [GET\\_NEXT\\_HIT Function](#) on page 126-11
- [CLOSE\\_ITERATOR Procedure](#) on page 126-6

## GET\_NEXT\_HIT Function

This function returns the next rule that evaluated to `TRUE` from a true rules iterator, or returns the next rule that evaluated to `MAYBE` from a maybe rules iterator. The function returns `NULL` if there are no more rules that evaluated to `TRUE` or `MAYBE`.

### Syntax

```
DBMS_RULE.GET_NEXT_HIT(  
    iterator IN BINARY_INTEGER)  
RETURN SYS.RE$RULE_HIT;
```

### Parameter

**Table 126-4** *GET\_NEXT\_HIT Function Parameter*

Parameter	Description
iterator	The iterator from which the rule that evaluated to <code>TRUE</code> or <code>MAYBE</code> is retrieved

### Usage Notes

This procedure requires an open iterator that was returned by an earlier call to `DBMS_RULE.EVALUATE` in the same session. The user who runs this procedure does not require any privileges on the rule set being evaluated.

When an iterator returns `NULL`, it is closed automatically. If an open iterator is no longer needed, then use the `CLOSE_ITERATOR` procedure in the `DBMS_RULE` package to close it.

---

**Note:** This function raises an error if the rule set being evaluated was modified after the call to the `DBMS_RULE.EVALUATE` procedure that returned the iterator. Modifications to a rule set include added rules to the rule set, changing existing rules in the rule set, dropping rules from the rule set, and dropping the rule set.

---

#### See Also:

- [Chapter 256, "Rule TYPES"](#) for more information about the types used with the `DBMS_RULE` package
- [EVALUATE Procedures](#) on page 126-7
- [CLOSE\\_ITERATOR Procedure](#) on page 126-6





---

---

## DBMS\_RULE\_ADM

The DBMS\_RULE\_ADM package provides the subprograms for creating and managing rules, rule sets, and rule evaluation contexts.

This chapter contains the following topics:

- [Using DBMS\\_RULE\\_ADM](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_RULE\\_ADM Subprograms](#)

## Using DBMS\_RULE\_ADM

This section contains topics which relate to using the DBMS\_RULE\_ADM package.

- [Overview](#)
- [Security Model](#)

## Overview

This package provides the subprograms for creating and managing rules, rule sets, and rule evaluation contexts.

**See Also:**

- [Chapter 256, "Rule TYPES"](#) for more information about the types used with the DBMS\_RULE\_ADM package
- [Chapter 126, "DBMS\\_RULE"](#) and *Oracle Streams Concepts and Administration* for more information about this package and rules

## Security Model

PUBLIC is granted EXECUTE privilege on this package.

**See Also:** *Oracle Database Security Guide* for more information about user group PUBLIC

---

## Summary of DBMS\_RULE\_ADM Subprograms

**Table 127-1 DBMS\_RULE\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ADD_RULE Procedure</a> on page 127-6	Adds the specified rule to the specified rule set
<a href="#">ALTER_EVALUATION_CONTEXT Procedure</a> on page 127-8	Alters a rule evaluation context
<a href="#">ALTER_RULE Procedure</a> on page 127-11	Changes one or more aspects of the specified rule
<a href="#">CREATE_EVALUATION_CONTEXT Procedure</a> on page 127-13	Creates a rule evaluation context
<a href="#">CREATE_RULE Procedure</a> on page 127-15	Creates a rule with the specified name
<a href="#">CREATE_RULE_SET Procedure</a> on page 127-17	Creates a rule set with the specified name
<a href="#">DROP_EVALUATION_CONTEXT Procedure</a> on page 127-18	Drops the rule evaluation context with the specified name
<a href="#">DROP_RULE Procedure</a> on page 127-19	Drops the rule with the specified name
<a href="#">DROP_RULE_SET Procedure</a> on page 127-20	Drops the rule set with the specified name
<a href="#">GRANT_OBJECT_PRIVILEGE Procedure</a> on page 127-21	Grants the specified object privilege on the specified object to the specified user or role
<a href="#">GRANT_SYSTEM_PRIVILEGE Procedure</a> on page 127-23	Grants the specified system privilege to the specified user or role
<a href="#">REMOVE_RULE Procedure</a> on page 127-25	Removes the specified rule from the specified rule set
<a href="#">REVOKE_OBJECT_PRIVILEGE Procedure</a> on page 127-27	Revokes the specified object privilege on the specified object from the specified user or role
<a href="#">REVOKE_SYSTEM_PRIVILEGE Procedure</a> on page 127-28	Revokes the specified system privilege from the specified user or role

---

**Note:** All subprograms commit unless specified otherwise.

---

## ADD\_RULE Procedure

This procedure adds the specified rule to the specified rule set.

### Syntax

```
DBMS_RULE_ADM.ADD_RULE(
    rule_name          IN  VARCHAR2,
    rule_set_name     IN  VARCHAR2,
    evaluation_context IN  VARCHAR2  DEFAULT NULL,
    rule_comment      IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 127–2 ADD\_RULE Procedure Parameters**

Parameter	Description
rule_name	The name of the rule you are adding to the rule set, specified as <code>[schema_name.] rule_name</code> . For example, to add a rule named <code>all_a</code> in the <code>hr</code> schema, enter <code>hr.all_a</code> for this parameter. If the schema is not specified, then the current user is the default.
rule_set_name	The name of the rule set to which you are adding the rule, specified as <code>[schema_name.] rule_set_name</code> . For example, to add the rule to a rule set named <code>apply_rules</code> in the <code>hr</code> schema, enter <code>hr.apply_rules</code> for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context	An evaluation context name in the form <code>[schema_name.] evaluation_context_name</code> . If the schema is not specified, then the current user is the default.  Only specify an evaluation context if the rule itself does not have an evaluation context and you do not want to use the rule set's evaluation context for the rule.
rule_comment	Optional description, which can contain the reason for adding the rule to the rule set

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have `ALTER_ON_RULE_SET` privilege on the rule set
- Have `ALTER_ANY_RULE_SET` system privilege
- Be the owner of the rule set

Also, the rule set owner must meet at least one of the following requirements:

- Have `EXECUTE_ON_RULE` privilege on the rule
- Have `EXECUTE_ANY_RULE` system privilege
- Be the rule owner

If the rule has no evaluation context and no evaluation context is specified when you run this procedure, then the rule uses the evaluation context associated with the rule set. In such a case, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have EXECUTE\_ON\_EVALUATION\_CONTEXT privilege on the evaluation context
- Have EXECUTE\_ANY\_EVALUATION\_CONTEXT system privilege, and the owner of the evaluation context must not be SYS
- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

## ALTER\_EVALUATION\_CONTEXT Procedure

This procedure alters a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

### Syntax

```
DBMS_RULE_ADM.ALTER_EVALUATION_CONTEXT (
  evaluation_context_name      IN  VARCHAR2,
  table_aliases                IN  SYS.RE$TABLE_ALIAS_LIST   DEFAULT NULL,
  remove_table_aliases        IN  BOOLEAN                   DEFAULT FALSE,
  variable_types              IN  SYS.RE$VARIABLE_TYPE_LIST  DEFAULT NULL,
  remove_variable_types       IN  BOOLEAN                   DEFAULT FALSE,
  evaluation_function          IN  VARCHAR2                 DEFAULT NULL,
  remove_evaluation_function   IN  BOOLEAN                   DEFAULT FALSE,
  evaluation_context_comment   IN  VARCHAR2                 DEFAULT NULL,
  remove_eval_context_comment  IN  BOOLEAN                   DEFAULT FALSE);
```

### Parameters

**Table 127-3 ALTER\_EVALUATION\_CONTEXT Procedure Parameters**

Parameter	Description
<code>evaluation_context_name</code>	<p>The name of the evaluation context you are altering, specified as <code>[schema_name.] evaluation_context_name</code>.</p> <p>For example, to alter an evaluation context named <code>dept_eval_context</code> in the <code>hr</code> schema, enter <code>hr.dept_eval_context</code> for this parameter. If the schema is not specified, then the current user is the default.</p>
<code>table_aliases</code>	<p>If <code>NULL</code> and <code>remove_table_aliases</code> is <code>FALSE</code>, then the procedure retains the existing table aliases. If <code>NULL</code> and <code>remove_table_aliases</code> is <code>TRUE</code>, then the procedure removes the existing table aliases.</p> <p>If non-<code>NULL</code>, then the procedure replaces the existing table aliases for the evaluation context with the specified table aliases.</p> <p>Table aliases specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.</p>
<code>remove_table_aliases</code>	<p>If <code>TRUE</code> and <code>table_aliases</code> is <code>NULL</code>, then the procedure removes the existing table aliases for the evaluation context. If <code>TRUE</code> and <code>table_aliases</code> is non-<code>NULL</code>, then the procedure raises an error.</p> <p>If <code>FALSE</code>, then the procedure does not remove table aliases.</p>
<code>variable_types</code>	<p>If <code>NULL</code> and <code>remove_variable_types</code> is <code>FALSE</code>, then the procedure retains the variable types. If <code>NULL</code> and <code>remove_variable_types</code> is <code>TRUE</code>, then the procedure removes the existing variable types.</p> <p>If non-<code>NULL</code>, then the procedure replaces the existing variable types for the evaluation context with the specified variable types.</p>



**Table 127-3 (Cont.) ALTER\_EVALUATION\_CONTEXT Procedure Parameters**

Parameter	Description
remove_variable_types	<p>If TRUE and variable_types is NULL, then the procedure removes the existing variable types for the evaluation context. If TRUE and variable_types is non-NULL, then the procedure raises an error.</p> <p>If FALSE, then the procedure does not remove the variable types.</p>
evaluation_function	<p>If NULL and remove_evaluation_function is FALSE, then the procedure retains the existing evaluation function. If NULL and remove_evaluation_function is TRUE, then the procedure removes the existing evaluation function.</p> <p>If non-NULL, then the procedure replaces the existing evaluation function for the evaluation context with the specified evaluation function.</p> <p>An evaluation function is an optional function that will be called to evaluate rules that use the evaluation context. It must have the same form as the DBMS_RULE.EVALUATE procedure. If the schema is not specified, then the current user is the default.</p> <p>See <a href="#">CREATE_EVALUATION_CONTEXT Procedure</a> on page 127-13 for more information about evaluation functions.</p>
remove_evaluation_function	<p>If TRUE and evaluation_function is NULL, then the procedure removes the existing evaluation function for the evaluation context. If TRUE and evaluation_function is non-NULL, then the procedure raises an error.</p> <p>If FALSE, then the procedure does not remove the evaluation function.</p>
evaluation_context_comment	<p>If NULL and remove_eval_context_comment is FALSE, then the procedure retains the existing evaluation context comment. If NULL and remove_evaluation_function is TRUE, then the procedure removes the existing evaluation context comment.</p> <p>If non-NULL, then the procedure replaces the existing comment for the evaluation context with the specified comment.</p> <p>An evaluation context comment is an optional description of the rule evaluation context.</p>
remove_eval_context_comment	<p>If TRUE and evaluation_context_comment is NULL, then the procedure removes the existing comment for the evaluation context. If TRUE and evaluation_context_comment is non-NULL, then the procedure raises an error.</p> <p>If FALSE, then the procedure does not remove the evaluation context comment.</p>

## Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being altered
- Have ALL\_ON\_EVALUATION\_CONTEXT or ALTER\_ON\_EVALUATION\_CONTEXT object privilege on an evaluation context owned by another user

- Have ALTER\_ANY\_EVALUATION\_CONTEXT system privilege

**See Also:** [Chapter 256, "Rule TYPES"](#) for more information about the types used with the DBMS\_RULE\_ADM package

## ALTER\_RULE Procedure

This procedure changes one or more aspects of the specified rule.

### Syntax

```
DBMS_RULE_ADM.ALTER_RULE (
  rule_name           IN  VARCHAR2,
  condition           IN  VARCHAR2           DEFAULT NULL,
  evaluation_context  IN  VARCHAR2           DEFAULT NULL,
  remove_evaluation_context IN  BOOLEAN       DEFAULT FALSE,
  action_context     IN  SYS.RE$NV_LIST     DEFAULT NULL,
  remove_action_context IN  BOOLEAN         DEFAULT FALSE,
  rule_comment       IN  VARCHAR2           DEFAULT NULL,
  remove_rule_comment IN  BOOLEAN           DEFAULT FALSE);
```

### Parameters

**Table 127-4 ALTER\_RULE Procedure Parameters**

Parameter	Description
rule_name	The name of the rule you are altering, specified as [ <i>schema_name</i> .] <i>rule_name</i> . For example, to alter a rule named <i>all_a</i> in the <i>hr</i> schema, enter <i>hr.all_a</i> for this parameter. If the schema is not specified, then the current user is the default.
condition	The condition to be associated with the rule. If non-NULL, then the procedure replaces the existing condition of the rule with the specified condition.
evaluation_context	An evaluation context name in the form [ <i>schema_name</i> .] <i>evaluation_context_name</i> . If the schema is not specified, then the current user is the default. If non-NULL, then the procedure replaces the existing evaluation context of the rule with the specified evaluation context.
remove_evaluation_context	If TRUE, then the procedure sets the evaluation context for the rule to NULL, which effectively removes the evaluation context from the rule. If FALSE, then the procedure retains any evaluation context for the specified rule. If the <i>evaluation_context</i> parameter is non-NULL, then this parameter should be set to FALSE.
action_context	If non-NULL, then the procedure changes the action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.
remove_action_context	If TRUE, then the procedure sets the action context for the rule to NULL, which effectively removes the action context from the rule. If FALSE, then the procedure retains any action context for the specified rule. If the <i>action_context</i> parameter is non-NULL, then this parameter should be set to FALSE.
rule_comment	If non-NULL, then the existing comment of the rule is replaced by the specified comment.

**Table 127-4 (Cont.) ALTER\_RULE Procedure Parameters**

Parameter	Description
<code>remove_rule_comment</code>	<p>If <code>TRUE</code>, then the procedure sets the comment for the rule to <code>NULL</code>, which effectively removes the comment from the rule.</p> <p>If <code>FALSE</code>, then the procedure retains any comment for the specified rule.</p> <p>If the <code>rule_comment</code> parameter is non-<code>NULL</code>, then this parameter should be set to <code>FALSE</code>.</p>

## Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have `ALTER_ON_RULE` privilege on the rule
- Have `ALTER_ANY_RULE` system privilege
- Be the owner of the rule being altered

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

- Have `EXECUTE_ON_EVALUATION_CONTEXT` privilege on the evaluation context
- Have `EXECUTE_ANY_EVALUATION_CONTEXT` system privilege, and the owner of the evaluation context must not be `SYS`
- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

**See Also:** [Chapter 256, "Rule TYPES"](#) for more information about the types used with the `DBMS_RULE_ADM` package

## CREATE\_EVALUATION\_CONTEXT Procedure

This procedure creates a rule evaluation context. A rule evaluation context defines external data that can be referenced in rule conditions. The external data can either exist as variables or as table data.

### Syntax

```
DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT (
  evaluation_context_name      IN  VARCHAR2,
  table_aliases                IN  SYS.RE$TABLE_ALIAS_LIST   DEFAULT NULL,
  variable_types               IN  SYS.RE$VARIABLE_TYPE_LIST DEFAULT NULL,
  evaluation_function           IN  VARCHAR2                 DEFAULT NULL,
  evaluation_context_comment   IN  VARCHAR2                 DEFAULT NULL);
```

### Parameters

**Table 127–5 CREATE\_EVALUATION\_CONTEXT Procedure Parameters**

Parameter	Description
evaluation_context_name	The name of the evaluation context you are creating, specified as <i>[schema_name.] evaluation_context_name</i> .  For example, to create an evaluation context named <code>dept_eval_context</code> in the <code>hr</code> schema, enter <code>hr.dept_eval_context</code> for this parameter. If the schema is not specified, then the current user is the default.
table_aliases	Table aliases that specify the tables in an evaluation context. The table aliases can be used to reference tables in rule conditions.
variable_types	A list of variables for the evaluation context
evaluation_function	An optional function that will be called to evaluate rules using the evaluation context. It must have the same form as the <code>DBMS_RULE.EVALUATE</code> procedure. If the schema is not specified, then the current user is the default.  See "Usage Notes" for more information about the evaluation function.
evaluation_context_comment	An optional description of the rule evaluation context.

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context being created and have `CREATE_EVALUATION_CONTEXT_OBJ` system privilege
- Have `CREATE_ANY_EVALUATION_CONTEXT` system privilege

**See Also:** [Chapter 256, "Rule TYPES"](#) for more information about the types used with the `DBMS_RULE_ADM` package

The evaluation function must have the following signature:

```
FUNCTION evaluation_function_name(
  rule_set_name      IN  VARCHAR2,
```

```

evaluation_context IN VARCHAR2,
event_context     IN SYS.RE$NV_LIST           DEFAULT NULL,
table_values      IN SYS.RE$TABLE_VALUE_LIST  DEFAULT NULL,
column_values     IN SYS.RE$COLUMN_VALUE_LIST DEFAULT NULL,
variable_values   IN SYS.RE$VARIABLE_VALUE_LIST DEFAULT NULL,
attribute_values  IN SYS.RE$ATTRIBUTE_VALUE_LIST DEFAULT NULL,
stop_on_first_hit IN BOOLEAN                 DEFAULT FALSE,
simple_rules_only  IN BOOLEAN                 DEFAULT FALSE,
true_rules        OUT SYS.RE$RULE_HIT_LIST,
maybe_rules      OUT SYS.RE$RULE_HIT_LIST);
RETURN BINARY_INTEGER;

```

---

**Note:** Each parameter is required and must have the specified data type. However, you can change the names of the parameters.

---

The return value of the function must be one of the following:

- `DBMS_RULE_ADM.EVALUATION_SUCCESS`: The user specified evaluation function completed the rule set evaluation successfully. The rules engine returns the results of the evaluation obtained by the evaluation function to the rules engine client using the `DBMS_RULE.EVALUATE` procedure.
- `DBMS_RULE_ADM.EVALUATION_CONTINUE`: The rules engine evaluates the rule set as if there were no evaluation function. The evaluation function is not used, and any results returned by the evaluation function are ignored.
- `DBMS_RULE_ADM.EVALUATION_FAILURE`: The user specified evaluation function failed. Rule set evaluation stops, and an error is raised.

## CREATE\_RULE Procedure

This procedure creates a rule.

### Syntax

```
DBMS_RULE_ADM.CREATE_RULE(
  rule_name          IN  VARCHAR2,
  condition          IN  VARCHAR2,
  evaluation_context IN  VARCHAR2          DEFAULT NULL,
  action_context     IN  SYS.RE$NV_LIST  DEFAULT NULL,
  rule_comment       IN  VARCHAR2          DEFAULT NULL);
```

### Parameters

**Table 127–6 CREATE\_RULE Procedure Parameters**

Parameter	Description
rule_name	The name of the rule you are creating, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . For example, to create a rule named all_a in the hr schema, enter hr.all_a for this parameter. If the schema is not specified, then the current user is the default.
condition	The condition to be associated with the rule. A condition evaluates to TRUE or FALSE and can be any condition allowed in the WHERE clause of a SELECT statement. For example, the following is a valid rule condition:  department_id = 30  Ensure that the proper case is used for text in rule conditions. <b>Note:</b> Do not include the word "WHERE" in the condition.
evaluation_context	An optional evaluation context name in the form [ <i>schema_name.</i> ] <i>evaluation_context_name</i> , which is associated with the rule. If the schema is not specified, then the current user is the default.  If <i>evaluation_context</i> is not specified, then the rule inherits the evaluation context from its rule set.
action_context	The action context associated with the rule. A rule action context is information associated with a rule that is interpreted by the client of the rules engine when the rule is evaluated.
rule_comment	An optional description of the rule

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule being created and have the CREATE\_RULE\_OBJ system privilege
- Have CREATE\_ANY\_RULE system privilege

If an evaluation context is specified, then the rule owner must meet at least one of the following requirements:

- Have EXECUTE\_ON\_EVALUATION\_CONTEXT privilege on the evaluation context
- Have EXECUTE\_ANY\_EVALUATION\_CONTEXT system privilege, and the owner of the evaluation context must not be SYS.

- Be the evaluation context owner

Also, the rule owner must have the necessary privileges on all the base objects accessed by the rule using the evaluation context.

**See Also:** [Chapter 256, "Rule TYPES"](#) for more information about the types used with the `DBMS_RULE_ADM` package



## CREATE\_RULE\_SET Procedure

This procedure creates a rule set.

### Syntax

```
DBMS_RULE_ADM.CREATE_RULE_SET(
  rule_set_name      IN  VARCHAR2,
  evaluation_context IN  VARCHAR2  DEFAULT NULL,
  rule_set_comment   IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 127-7 CREATE\_RULE\_SET Procedure Parameters**

Parameter	Description
rule_set_name	The name of the rule set you are creating, specified as [ <i>schema_name</i> .] <i>rule_set_name</i> . For example, to create a rule set named <i>apply_rules</i> in the <i>hr</i> schema, enter <i>hr.apply_rules</i> for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context	An optional evaluation context name in the form [ <i>schema_name</i> .] <i>evaluation_context_name</i> , which applies to all rules in the rule set that are not associated with an evaluation context explicitly. If the schema is not specified, then the current user is the default.
rule_set_comment	An optional description of the rule set

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule set being created and have `CREATE_RULE_SET_OBJ` system privilege
- Have `CREATE_ANY_RULE_SET` system privilege

If an evaluation context is specified, then the rule set owner must meet at least one of the following requirements:

- Have `EXECUTE_ON_EVALUATION_CONTEXT` privilege on the evaluation context
- Have `EXECUTE_ANY_EVALUATION_CONTEXT` system privilege, and the owner of the evaluation context must not be `SYS`
- Be the evaluation context owner

## DROP\_EVALUATION\_CONTEXT Procedure

This procedure drops a rule evaluation context.

### Syntax

```
DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT (
  evaluation_context_name IN VARCHAR2,
  force                    IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 127–8 DROP\_EVALUATION\_CONTEXT Procedure Parameters**

Parameter	Description
evaluation_context_name	<p>The name of the evaluation context you are dropping, specified as [<i>schema_name</i>.]<i>evaluation_context_name</i>.</p> <p>For example, to drop an evaluation context named <code>dept_eval_context</code> in the <code>hr</code> schema, enter <code>hr.dept_eval_context</code> for this parameter. If the schema is not specified, then the current user is the default.</p>
force	<p>If <code>TRUE</code>, then the procedure removes the rule evaluation context from all rules and rule sets that use it.</p> <p>If <code>FALSE</code> and no rules or rule sets use the rule evaluation context, then the procedure drops the rule evaluation context.</p> <p>If <code>FALSE</code> and one or more rules or rule sets use the rule evaluation context, then the procedure raises an exception.</p> <p><b>Caution:</b> Setting <code>force</code> to <code>TRUE</code> can result in rules and rule sets that do not have an evaluation context. If neither a rule nor the rule set it is in has an evaluation context, and no evaluation context was specified for the rule by the <code>ADD_RULE</code> procedure, then the rule cannot be evaluated.</p>

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the evaluation context
- Have `DROP_ANY_EVALUATION_CONTEXT` system privilege

## DROP\_RULE Procedure

This procedure drops a rule.

### Syntax

```
DBMS_RULE_ADM.DROP_RULE(
  rule_name IN VARCHAR2,
  force     IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 127–9 DROP\_RULE Procedure Parameters**

Parameter	Description
rule_name	The name of the rule you are dropping, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . For example, to drop a rule named <code>all_a</code> in the <code>hr</code> schema, enter <code>hr.all_a</code> for this parameter. If the schema is not specified, then the current user is the default.
force	If <code>TRUE</code> , then the procedure removes the rule from all rule sets that contain it.  If <code>FALSE</code> and no rule sets contain the rule, then the procedure drops the rule.  If <code>FALSE</code> and one or more rule sets contain the rule, then the procedure raises an exception.

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the rule
- Have `DROP_ANY_RULE` system privilege

---



---

**Note:**

- To remove a rule from a rule set without dropping the rule from the database, use the `REMOVE_RULE` procedure.
  - The rule evaluation context associated with the rule, if any, is not dropped when you run this procedure.
- 
-

## DROP\_RULE\_SET Procedure

This procedure drops a rule set.

### Syntax

```
DBMS_RULE_ADM.DROP_RULE_SET(
    rule_set_name IN VARCHAR2,
    delete_rules IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 127–10 DROP\_RULE\_SET Procedure Parameters**

Parameter	Description
rule_set_name	The name of the rule set you are dropping, specified as <code>[schema_name.] rule_set_name</code> . For example, to drop a rule set named <code>apply_rules</code> in the <code>hr</code> schema, enter <code>hr.apply_rules</code> for this parameter. If the schema is not specified, then the current user is the default.
delete_rules	If <code>TRUE</code> , then the procedure drops any rules that are in the rule set. If any of the rules in the rule set are also in another rule set, then these rules are not dropped.  If <code>FALSE</code> , then the procedure does not drop the rules in the rule set.

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have `DROP_ANY_RULE_SET` system privilege
- Be the owner of the rule set

---



---

**Note:** The rule evaluation context associated with the rule set, if any, is not dropped when you run this procedure.

---



---

## GRANT\_OBJECT\_PRIVILEGE Procedure

This procedure grants the specified object privilege on the specified object to the specified user or role. If a user owns the object, then the user automatically is granted all privileges on the object, with grant option.

### Syntax

```
DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE (
  privilege      IN  BINARY_INTEGER,
  object_name    IN  VARCHAR2,
  grantee        IN  VARCHAR2,
  grant_option   IN  BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 127–11 GRANT\_OBJECT\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The name of the object privilege to grant to the grantee on the object. See "Usage Notes" on page 127-21 for the available object privileges.
object_name	The name of the object for which you are granting the privilege to the grantee, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, to grant the privilege on a rule set named <i>apply_rules</i> in the <i>hr</i> schema, enter <i>hr.apply_rules</i> for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.
grantee	The name of the user or role for which the privilege is granted. The specified user cannot be the owner of the object.
grant_option	If TRUE, then the specified user or users granted the specified privilege can grant this privilege to others.  If FALSE, then the specified user or users granted the specified privilege cannot grant this privilege to others.

### Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Be the owner of the object on which the privilege is granted
- Have the same privilege as the privilege being granted with the grant option

In addition, if the object is a rule set, then the user must have EXECUTE privilege on all the rules in the rule set with grant option or must own the rules in the rule set.

[Table 127–12](#) lists the object privileges.

**Table 127–12 Object Privileges for Evaluation Contexts, Rules, and Rule Sets**

Privilege	Description
<code>SYS.DBMS_RULE_ADM.ALL_ON_EVALUATION_CONTEXT</code>	Alter and execute a particular evaluation context in another user's schema
<code>SYS.DBMS_RULE_ADM.ALL_ON_RULE</code>	Alter and execute a particular rule in another user's schema
<code>SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET</code>	Alter and execute a particular rule set in another user's schema
<code>SYS.DBMS_RULE_ADM.ALTER_ON_EVALUATION_CONTEXT</code>	Alter a particular evaluation context in another user's schema
<code>SYS.DBMS_RULE_ADM.ALTER_ON_RULE</code>	Alter a particular rule in another user's schema
<code>SYS.DBMS_RULE_ADM.ALTER_ON_RULE_SET</code>	Alter a particular rule set in another user's schema
<code>SYS.DBMS_RULE_ADM.EXECUTE_ON_EVALUATION_CONTEXT</code>	Execute a particular evaluation context in another user's schema
<code>SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE</code>	Execute a particular rule in another user's schema
<code>SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET</code>	Execute a particular rule set in another user's schema

## Examples

For example, to grant the HR user the privilege to alter a rule named `hr_dml` in the `strmadmin` schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALTER_ON_RULE,
    object_name  => 'strmadmin.hr_dml',
    grantee     => 'hr',
    grant_option => FALSE);
END;
/
```

## GRANT\_SYSTEM\_PRIVILEGE Procedure

This procedure grant the specified system privilege to the specified user or role.

### Syntax

```
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE (
  privilege      IN  BINARY_INTEGER,
  grantee        IN  VARCHAR2,
  grant_option   IN  BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 127–13 GRANT\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The name of the system privilege to grant to the grantee.
grantee	The name of the user or role for which the privilege is granted
grant_option	If TRUE, then the specified user or users granted the specified privilege can grant the system privilege to others. If FALSE, then the specified user or users granted the specified privilege cannot grant the system privilege to others.

### Usage Notes

[Table 127–14](#) lists the system privileges.

**Table 127–14 System Privileges for Evaluation Contexts, Rules, and Rule Sets**

Privilege	Description
SYS.DBMS_RULE_ADM.ALTER_ANY_EVALUATION_CONTEXT	Alter any evaluation context owned by any user
SYS.DBMS_RULE_ADM.ALTER_ANY_RULE	Alter any rule owned by any user
SYS.DBMS_RULE_ADM.ALTER_ANY_RULE_SET	Alter any rule set owned by any user
SYS.DBMS_RULE_ADM.CREATE_ANY_EVALUATION_CONTEXT	Create a new evaluation context in any schema
SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ	Create a new evaluation context in the grantee's schema
SYS.DBMS_RULE_ADM.CREATE_ANY_RULE	Create a new rule in any schema
SYS.DBMS_RULE_ADM.CREATE_RULE_OBJ	Create a new rule in the grantee's schema
SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET	Create a new rule set in any schema
SYS.DBMS_RULE_ADM.CREATE_RULE_SET_OBJ	Create a new rule set in the grantee's schema
SYS.DBMS_RULE_ADM.DROP_ANY_EVALUATION_CONTEXT	Drop any evaluation context in any schema
SYS.DBMS_RULE_ADM.DROP_ANY_RULE	Drop any rule in any schema
SYS.DBMS_RULE_ADM.DROP_ANY_RULE_SET	Drop any rule set in any schema

**Table 127–14 (Cont.) System Privileges for Evaluation Contexts, Rules, and Rule Sets**

Privilege	Description
<code>SYS.DBMS_RULE_ADM.EXECUTE_ANY_EVALUATION_CONTEXT</code>	Execute any evaluation context owned by any user
<code>SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE</code>	Execute any rule owned by any user
<code>SYS.DBMS_RULE_ADM.EXECUTE_ANY_RULE_SET</code>	Execute any rule set owned by any user

For example, to grant the `strmadmin` user the privilege to create a rule set in any schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => SYS.DBMS_RULE_ADM.CREATE_ANY_RULE_SET,
    grantee   => 'strmadmin',
    grant_option => FALSE);
END;
/
```

---

**Note:** When you grant a privilege on "ANY" object (for example, `ALTER_ANY_RULE`), and the initialization parameter `O7_DICTIONARY_ACCESSIBILITY` is set to `FALSE`, you give the user access to that type of object in all schemas except the `SYS` schema. By default, the initialization parameter `O7_DICTIONARY_ACCESSIBILITY` is set to `FALSE`.

If you want to grant access to an object in the `SYS` schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter to `TRUE`. Then privileges granted on "ANY" object allows access to any schema, including `SYS`.

---



## REMOVE\_RULE Procedure

This procedure removes the specified rule from the specified rule set.

### Syntax

```
DBMS_RULE_ADM.REMOVE_RULE(
  rule_name           IN  VARCHAR2,
  rule_set_name      IN  VARCHAR2,
  evaluation_context IN  VARCHAR2  DEFAULT NULL,
  all_evaluation_contexts IN  BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 127–15 REMOVE\_RULE Procedure Parameters**

Parameter	Description
rule_name	The name of the rule you are removing from the rule set, specified as [ <i>schema_name</i> .] <i>rule_name</i> . For example, to remove a rule named <i>all_a</i> in the <i>hr</i> schema, enter <i>hr.all_a</i> for this parameter. If the schema is not specified, then the current user is the default.
rule_set_name	The name of the rule set from which you are removing the rule, specified as [ <i>schema_name</i> .] <i>rule_set_name</i> . For example, to remove the rule from a rule set named <i>apply_rules</i> in the <i>hr</i> schema, enter <i>hr.apply_rules</i> for this parameter. If the schema is not specified, then the current user is the default.
evaluation_context_name	The name of the evaluation context associated with the rule you are removing, specified as [ <i>schema_name</i> .] <i>evaluation_context_name</i> . For example, to specify an evaluation context named <i>dept_eval_context</i> in the <i>hr</i> schema, enter <i>hr.dept_eval_context</i> for this parameter. If the schema is not specified, then the current user is the default.  If an evaluation context was specified for the rule you are removing when you added the rule to the rule set using the <i>ADD_RULE</i> procedure, then specify the same evaluation context. If you added the same rule more than once with different evaluation contexts, then specify the rule with the evaluation context you want to remove. If you specify an evaluation context that is not associated with the rule, then the procedure raises an error.  Specify <i>NULL</i> if you did not specify an evaluation context when you added the rule to the rule set. If you specify <i>NULL</i> and there are one or more evaluation contexts associated with the rule, then the procedure raises an error.
all_evaluation_contexts	If <i>TRUE</i> , then the procedure removes the rule from the rule set with all of its associated evaluation contexts.  If <i>FALSE</i> , then the procedure only removes the rule with the specified evaluation context.  This parameter is relevant only if the same rule is added more than once to the rule set with different evaluation contexts.

## Usage Notes

To run this procedure, a user must meet at least one of the following requirements:

- Have ALTER\_ON\_RULE\_SET privilege on the rule set
- Have ALTER\_ANY\_RULE\_SET system privilege
- Be the owner of the rule set

---

---

**Note:** This procedure does not drop a rule from the database. To drop a rule from the database, use the DROP\_RULE procedure.

---

---

## REVOKE\_OBJECT\_PRIVILEGE Procedure

This procedure revokes the specified object privilege on the specified object from the specified user or role.

### Syntax

```
DBMS_RULE_ADM.REVOKE_OBJECT_PRIVILEGE (
  privilege      IN  BINARY_INTEGER,
  object_name    IN  VARCHAR2,
  revokee       IN  VARCHAR2);
```

### Parameters

**Table 127–16 REVOKE\_OBJECT\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The name of the object privilege on the object to revoke from the revokee. See <a href="#">GRANT_OBJECT_PRIVILEGE Procedure</a> on page 127-21 for a list of the object privileges.
object_name	The name of the object for which you are revoking the privilege from the revokee, specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, to revoke an object privilege on a rule set named <i>apply_rules</i> in the <i>hr</i> schema, enter <i>hr.apply_rules</i> for this parameter. If the schema is not specified, then the current user is the default. The object must be an existing rule, rule set, or evaluation context.
revokee	The name of the user or role from which the privilege is revoked. The user who owns the object cannot be specified.

## REVOKE\_SYSTEM\_PRIVILEGE Procedure

This procedure revokes the specified system privilege from the specified user or role.

### Syntax

```
DBMS_RULE_ADM.REVOKE_SYSTEM_PRIVILEGE(  
    privilege IN BINARY_INTEGER,  
    revokee   IN VARCHAR2);
```

### Parameters

**Table 127–17 REVOKE\_SYSTEM\_PRIVILEGE Procedure Parameters**

Parameter	Description
privilege	The name of the system privilege to revoke from the revokee. See <a href="#">GRANT_SYSTEM_PRIVILEGE Procedure</a> on page 127-23 for a list of the system privileges.
revokee	The name of the user or role from which the privilege is revoked

---

---

## DBMS\_SCHEDULER

The DBMS\_SCHEDULER package provides a collection of scheduling functions and procedures that can be called from any PL/SQL program.

**See Also:** *Oracle Database Administrator's Guide* for more information regarding how to use DBMS\_SCHEDULER

This chapter contains the following topics:

- [Data Structures](#)
- [Using DBMS\\_SCHEDULER](#)
  - Security Model
  - Rules and Limits
  - Operational Notes
- [Summary of DBMS\\_SCHEDULER Subprograms](#)

## Data Structures

The DBMS\_SCHEDULER package defines OBJECT types and TABLE types.

### OBJECT Types

- [JOBARG Object Type](#)
- [JOB Object Type](#)
- [JOB\\_DEFINITION Object Type](#)
- [JOBATTR Object Type](#)
- [SCHEDULER\\$\\_STEP\\_TYPE Object Type](#)
- [SCHEDULER\\$\\_EVENT\\_INFO Object Type](#)
- [SCHEDULER\\_FILEWATCHER\\_RESULT Object Type](#)
- [SCHEDULER\\_FILEWATCHER\\_REQUEST Object Type](#)

### TABLE Types

- [JOBARG\\_ARRAY Table Type](#)
- [JOB\\_ARRAY Table Type](#)
- [JOB\\_DEFINITION\\_ARRAY Table Type](#)
- [JOBATTR\\_ARRAY Table Type](#)
- [SCHEDULER\\$\\_STEP\\_TYPE\\_LIST Table Type](#)

## JOBARG Object Type

This type is used by the JOB and JOBATTR object types. It represents a job argument in a batch of job arguments.

### Syntax

```
TYPE jobarg IS OBJECT (
  arg_position      NUMBER,
  arg_text_value    VARCHAR2(4000),
  arg_anydata_value ANYDATA,
  arg_operation     VARCHAR2(5));
```

### Attributes

**Table 128–1** JOBARG Object Type Attributes

Attribute	Description
arg_position	Position of the argument
arg_text_value	Value of the argument if the type is VARCHAR2
arg_anydata_value	Value of the argument if the type is AnyData
arg_operation	Type of the operation: <ul style="list-style-type: none"> <li>■ SET</li> <li>■ RESET</li> </ul>

## JOBARG Constructor Function

This constructor function constructs a job argument. It is overloaded to construct job arguments with different types of values.

### Syntax

Constructs a job argument with a text value.

```
constructor function jobarg (
  arg_position      IN POSITIVEN,
  arg_value         IN VARCHAR2)
RETURN SELF AS RESULT;
```

Constructs a job argument with an AnyData value.

```
constructor function jobarg (
  arg_position      IN POSITIVEN,
  arg_value         IN ANYDATA)
RETURN SELF AS RESULT;
```

Constructs a job argument with a NULL value.

```
constructor function jobarg (
  arg_position      IN POSITIVEN,
  arg_reset        IN BOOLEAN DEFAULT FALSE)
RETURN SELF AS RESULT;
```

## Parameters

**Table 128–2** *JOBARG Constructor Function Parameters*

Parameter	Description
<code>arg_position</code>	Position of the argument
<code>arg_value</code>	Value of the argument
<code>arg_reset</code>	If <code>arg_reset</code> is <code>TRUE</code> , then the argument at that position is reset. Setting <code>arg_reset</code> to <code>FALSE</code> (which is the default) will create an argument with a <code>NULL</code> value.

## JOBARG\_ARRAY Table Type

### Syntax

```
TYPE jobarg_array IS TABLE OF jobarg;
```



## **JOB Object Type**

**\*\*\*\* Deprecated.** Use the `JOB_DEFINITION` object type instead.

## **JOB\_ARRAY Table Type**

**\*\*\* Deprecated.** Use the `JOB_DEFINITION_ARRAY` table type instead.

## JOB\_DEFINITION Object Type

This type is used by the CREATE\_JOBS procedure and represents a job in a batch of jobs.

### Syntax

```

TYPE job_definition IS OBJECT (
  job_name          VARCHAR2(100),
  job_class         VARCHAR2(32),
  job_style         VARCHAR2(11),
  program_name     VARCHAR2(100),
  job_action        VARCHAR2(4000),
  job_type          VARCHAR2(20),
  schedule_name    VARCHAR2(65),
  repeat_interval  VARCHAR2(4000),
  schedule_limit   INTERVAL DAY TO SECOND,
  start_date       TIMESTAMP WITH TIME ZONE,
  end_date         TIMESTAMP WITH TIME ZONE,
  event_condition  VARCHAR2(4000),
  queue_spec       VARCHAR2(100),
  number_of_arguments NUMBER,
  arguments        SYS.JOBARG_ARRAY,
  job_priority     NUMBER,
  job_weight       NUMBER,
  max_run_duration INTERVAL DAY TO SECOND,
  max_runs         NUMBER,
  max_failures     NUMBER,
  logging_level    NUMBER,
  restartable     VARCHAR2(5),
  stop_on_window_close VARCHAR2(5),
  raise_events     NUMBER,
  comments        VARCHAR2(240),
  auto_drop       VARCHAR2(5),
  enabled         VARCHAR2(5),
  follow_default_timezone VARCHAR2(5),
  parallel_instances VARCHAR2(5),
  aq_job          VARCHAR2(5),
  instance_id     NUMBER,
  credential_name VARCHAR2(65),
  destination     VARCHAR2(4000),
  database_role   VARCHAR2(20),
  allow_runs_in_restricted_mode VARCHAR2(5));

```

### Object Attributes

Table 128–3 provides brief descriptions of the attributes of the JOB\_DEFINITION object type. For more complete information about these attributes, see the "CREATE\_JOB Procedure" on page 128-57 and the "SET\_ATTRIBUTE Procedure" on page 128-130.

**Table 128–3** JOB\_DEFINITION Object Type Attributes

Attribute	Description
job_name	Name of the job
job_class	Name of the job class

**Table 128–3 (Cont.) JOB\_DEFINITION Object Type Attributes**

Attribute	Description
job_style	Style of the job: <ul style="list-style-type: none"> <li>■ REGULAR</li> <li>■ LIGHTWEIGHT</li> </ul>
program_name	Name of the program that the job runs
job_action	Inline action of the job. This is either the code for an anonymous PL/SQL block or the name of a stored procedure, external executable, or chain.
job_type	Job action type ('PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', or 'CHAIN')
schedule_name	Name of the schedule that specifies when the job has to execute
repeat_interval	Inline time-based schedule
schedule_limit	Maximum delay time between scheduled and actual job start before a job run is canceled
start_date	Start date and time of the job
end_date	End date and time of the job
event_condition	Event condition for event-based jobs
queue_spec	File watcher name or queue specification for event-based jobs
number_of_arguments	Number of job arguments
arguments	Array of job arguments
job_priority	Job priority
job_weight	*** Deprecated in Oracle Database 11g Release 2 (11.2). Do not change the value of this attribute from the default, which is 1. Weight of the job for parallel execution.
max_run_duration	Maximum run duration of the job
max_runs	Maximum number of runs before the job is marked as completed
max_failures	Maximum number of failures tolerated before the job is marked as broken
logging_level	Job logging level
restartable	Indicates whether the job is restartable (TRUE) or not (FALSE)
stop_on_window_exit	Indicates whether the job is stopped when the window that it runs in ends (TRUE) or not (FALSE). Equivalent to the stop_on_window_close job attribute described in the <a href="#">SET_ATTRIBUTE Procedure</a> .
raise_events	State changes that raise events
comments	Comments on the job
auto_drop	If TRUE (the default), indicates that the job should be dropped once completed
enabled	Indicates whether the job should be enabled immediately after creating it (TRUE) or not (FALSE)
follow_default_timezone	If TRUE and if the job start_date is null, then when the default_timezone scheduler attribute is changed, the Scheduler recomputes the next run date and time for this job so that it is in accordance with the new time zone.

**Table 128–3 (Cont.) JOB\_DEFINITION Object Type Attributes**

Attribute	Description
parallel_instances	For event-based jobs only.  If TRUE, on the arrival of the specified event, the Scheduler creates a new lightweight job to handle that event, so multiple instances of the same event-based job can run in parallel.  If FALSE, then an event is discarded if it is raised while the job that handles it is already running,
aq_job	For internal use only
instance_id	The instance ID of the instance that the job must run on
credential_name	The credential to use for a single destination or the default credential for a group of destinations
destination	The name of a single external destination or database destination, or a group name of type external destination or database destination
database_role	In an Oracle Data Guard environment, the database role ('PRIMARY' or 'LOGICAL STANDBY') for which the job runs
allow_runs_in_restricted_mode	If TRUE, the job is permitted to run when the database is in restricted mode, provided that the job owner is permitted to log in during this mode

## JOB\_DEFINITION Constructor Function

This constructor function constructs a `job_definition` object.

### Syntax

```

constructor function job_definition (
  job_name           IN      VARCHAR2,
  job_style          IN      VARCHAR2 DEFAULT 'REGULAR',
  program_name      IN      VARCHAR2 DEFAULT NULL,
  job_action        IN      VARCHAR2 DEFAULT NULL,
  job_type          IN      VARCHAR2 DEFAULT NULL,
  schedule_name     IN      VARCHAR2 DEFAULT NULL,
  repeat_interval   IN      VARCHAR2 DEFAULT NULL,
  event_condition   IN      VARCHAR2 DEFAULT NULL,
  queue_spec        IN      VARCHAR2 DEFAULT NULL,
  start_date        IN      TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  end_date          IN      TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  number_of_arguments IN    NATURAL DEFAULT NULL,
  arguments         IN      SYS.JOBARG_ARRAY DEFAULT NULL,
  job_class         IN      VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS',
  schedule_limit    IN      INTERVAL DAY TO SECOND DEFAULT NULL,
  job_priority      IN      NATURAL DEFAULT NULL,
  job_weight        IN      NATURAL DEFAULT NULL,
  max_run_duration  IN      INTERVAL DAY TO SECOND DEFAULT NULL,
  max_runs          IN      NATURAL DEFAULT NULL,
  max_failures      IN      NATURAL DEFAULT NULL,
  logging_level     IN      NATURALN DEFAULT 64,
  restartable       IN      BOOLEAN DEFAULT FALSE,
  stop_on_window_close IN    BOOLEAN DEFAULT FALSE,
  raise_events      IN      NATURAL DEFAULT NULL,
  comments         IN      VARCHAR2 DEFAULT NULL,
  auto_drop         IN      BOOLEAN DEFAULT TRUE,
  enabled           IN      BOOLEAN DEFAULT FALSE,
  follow_default_timezone IN  BOOLEAN DEFAULT FALSE,

```

---

```
parallel_instances    IN    BOOLEAN DEFAULT FALSE,  
aq_job               IN    BOOLEAN DEFAULT FALSE,  
instance_id         IN    NATURAL DEFAULT NULL,  
credential_name      IN    VARCHAR2 DEFAULT NULL,  
destination          IN    VARCHAR2 DEFAULT NULL,  
database_role        IN    VARCHAR2 DEFAULT NULL,  
allow_runs_in_restricted_mode IN BOOLEAN DEFAULT FALSE)  
RETURN SELF AS RESULT;
```

## **JOB\_DEFINITION\_ARRAY Table Type**

### **Syntax**

```
TYPE job_definition_array IS TABLE OF job_definition;
```

## JOBATTR Object Type

This type is used by the `SET_JOB_ATTRIBUTES` procedure and represents a job attribute in a batch of job attributes.

### Syntax

```
TYPE jobattr IS OBJECT (
  job_name          VARCHAR2(100),
  attr_name         VARCHAR2(30),
  char_value        VARCHAR2(4000),
  char_value2       VARCHAR2(4000),
  args_value        JOBARG_ARRAY,
  num_value         NUMBER,
  timestamp_value   TIMESTAMP(6) WITH TIME ZONE,
  interval_value    INTERVAL DAY(2) TO SECOND(6));
```

### Attributes

**Table 128–4** *JOBATTR Object Type Attributes*

Attribute	Description
<code>job_name</code>	Name of the job
<code>attr_name</code>	Name of the attribute
<code>char_value</code>	Value of the argument if the type is <code>VARCHAR2</code>
<code>char_value2</code>	Second <code>VARCHAR2</code> attribute value
<code>args_value</code>	Value of the argument if the type is a <code>JOBARG</code> array
<code>num_value</code>	Value of the argument if the type is <code>NUMBER</code>
<code>timestamp_value</code>	Value of the argument if the type is <code>TIMESTAMP WITH TIME ZONE</code>
<code>interval_value</code>	Value of the argument if the type is <code>INTERVAL DAY TO SECOND</code>

### JOBATTR Constructor Function

This constructor function constructs a job attribute. It is overloaded to create attribute values of the following types: `VARCHAR2`, `NUMBER`, `TIMESTAMP WITH TIME ZONE`, `INTERVAL DAY TO SECOND`, and an array of `JOBARG` types.

### Syntax

```
constructor function jobattr (
  job_name          IN VARCHAR2,
  attr_name         IN VARCHAR2,
  attr_value        IN VARCHAR2,
  attr_value2       IN VARCHAR2 DEFAULT NULL)
  RETURN SELF AS RESULT;

constructor function jobattr (
  job_name          IN VARCHAR2,
  attr_name         IN VARCHAR2,
  attr_value        IN [NUMBER, BOOLEAN,
  TIMESTAMP WITH TIME ZONE,
  INTERVAL DAY TO SECOND, JOBARG_ARRAY])
```

```
RETURN SELF AS RESULT;

constructor function jobattr (
  job_name          IN VARCHAR2,
  attr_name         IN VARCHAR2)
RETURN SELF AS RESULT;
```

## Parameters

**Table 128–5** *JOBATTR Constructor Function Parameters*

Parameter	Description
job_name	Name of the job
attr_name	Name of the argument
attr_value	Value of the argument
attr_value2	Most attributes have only one value associated with them, but some can have two. The attr_value2 argument is for this optional second value.

## JOBATTR\_ARRAY Table Type

### Syntax

```
TYPE jobattr_array IS TABLE OF jobattr;
```

## SCHEDULER\$\_STEP\_TYPE Object Type

This type is used by RUN\_CHAIN to return a list of chain steps with an initial state.

### Syntax

```
TYPE scheduler$_step_type IS OBJECT (  
    step_name VARCHAR2(32),  
    step_type VARCHAR2(32));
```

### Attributes

**Table 128–6 SCHEDULER\$\_STEP\_TYPE Object Type Attributes**

Attribute	Description
step_name	Name of the step
step_type	State of the step

## SCHEDULER\$\_STEP\_TYPE\_LIST Table Type

### Syntax

```
TYPE scheduler$_step_type_list IS TABLE OF scheduler$_step_type;
```



## SCHEDULER\$\_EVENT\_INFO Object Type

This is the data type of the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`, from which your application consumes job state events raised by the Scheduler. It is a secure queue owned by `SYS`.

### Syntax

```
TYPE SCHEDULER$_EVENT_INFO IS OBJECT (
  event_type          VARCHAR2(4000),
  object_owner       VARCHAR2(4000),
  object_name        VARCHAR2(4000),
  event_timestamp    TIMESTAMP WITH TIME ZONE,
  error_code         NUMBER,
  error_msg          VARCHAR2(4000),
  event_status       NUMBER,
  log_id             NUMBER,
  run_count          NUMBER,
  failure_count      NUMBER,
  retry_count        NUMBER,
  spare1             NUMBER,
  spare2             NUMBER,
  spare3             VARCHAR2(4000),
  spare4             VARCHAR2(4000),
  spare5             TIMESTAMP WITH TIME ZONE,
  spare6             TIMESTAMP WITH TIME ZONE,
  spare7             RAW(2000),
  spare8            RAW(2000));
```

### Attributes

**Table 128–7 SCHEDULER\_EVENT\_INFO Object Type Attributes**

Attribute	Description
event_type	One of "JOB_STARTED", "JOB_SUCCEEDED", "JOB_FAILED", "JOB_BROKEN", "JOB_COMPLETED", "JOB_STOPPED", "JOB_SCH_LIM_REACHED", "JOB_DISABLED", "JOB_CHAIN_STALLED", "JOB_OVER_MAX_DUR".  For descriptions of these event types, see <a href="#">Table 128–82, "Event Types Raised by the Scheduler"</a> on page 128-138.
object_owner	Owner of the job that raised the event
object_name	Name of the job that raised the event
event_timestamp	Time at which the event occurred
error_code	Applicable only when an error is thrown during job execution. Contains the top-level error code.
error_msg	Applicable only when an error is thrown during job execution. Contains the entire error stack.

**Table 128-7 (Cont.) SCHEDULER\_EVENT\_INFO Object Type Attributes**

<b>Attribute</b>	<b>Description</b>
event_status	<p>Adds further qualification to the event type. If event_type is "JOB_STARTED," status 1 indicates that it is a normal start, and status 2 indicates that it is a retry.</p> <p>If event_type is "JOB_FAILED," status 4 indicates that it was a failure due to an error that was thrown during job execution, and status 8 indicates that it was an abnormal termination of some kind.</p> <p>If event_type is "JOB_STOPPED," status 16 indicates that it was a normal stop, and status 32 indicates that it was a stop with the FORCE option set to TRUE.</p>
log_id	Points to the ID in the scheduler job log from which additional information can be obtained. Note that there need not always be a log entry corresponding to an event. In such cases, log_id is NULL.
run_count	Run count for the job when the event was raised.
failure_count	Failure count for the job when the event was raised.
retry_count	Retry count for the job when the event was raised.
spare1 - spare8	Not currently in use.

## SCHEDULER\_FILEWATCHER\_RESULT Object Type

This is the data type of a file arrival event message. You access the event message as a parameter of an event-based job (or a parameter of a program referenced by an event-based job). The message contains information needed to locate and process a file that arrived on a local or remote system.

### Syntax

```
TYPE scheduler_filewatcher_result IS OBJECT (
  destination          VARCHAR2(4000),
  directory_path       VARCHAR2(4000),
  actual_file_name     VARCHAR2(4000),
  file_size            NUMBER,
  file_timestamp       TIMESTAMP WITH TIME ZONE,
  ts_ms_from_epoch    NUMBER,
  matching_requests    SYS.SCHEDULER_FILEWATCHER_REQ_LIST);
```

### Attributes

**Table 128–8 SCHEDULER\_FILEWATCHER\_RESULT Object Type Attributes**

Attribute	Description
destination	Destination at which the file was found, expressed as a host name or IP address.
directory_path	Absolute path of directory in which the file was found.
actual_file_name	Actual name of the file that was found. If the file name specified in the file watcher did not contain wildcards, then this is the same as the name specified in the file watcher.
file_size	Size of the file that was found, in bytes.
file_timestamp	Timestamp assigned to the file when the file watcher considered the file found, based on the minimum file size and steady state duration attributes.
ts_ms_from_epoch	For internal use only.
matching_requests	List of matching requests. This is a TABLE of type objects SCHEDULER_FILEWATCHER_REQUEST. Each matching request corresponds to a file watcher whose destination, directory_path, and file_name attributes matched the arrived file. See " <a href="#">SCHEDULER_FILEWATCHER_REQUEST Object Type</a> " on page 128-16.

## SCHEDULER\_FILEWATCHER\_REQUEST Object Type

This type is returned in the `matching_requests` attribute of the [SCHEDULER\\_FILEWATCHER\\_RESULT Object Type](#). Its attributes are similar to the attributes of a file watcher.

### Syntax

```
TYPE scheduler_filewatcher_request IS OBJECT (
  owner          VARCHAR2(4000),
  name           VARCHAR2(4000),
  requested_path_name VARCHAR2(4000),
  requested_file_name VARCHAR2(4000),
  credential_owner VARCHAR2(4000),
  credential_name VARCHAR2(4000),
  min_file_size  NUMBER,
  steady_state_dur NUMBER);
```

### Attributes

**Table 128–9 SCHEDULER\_FILEWATCHER\_RESULT Object Type Attributes**

Attribute	Description
<code>owner</code>	Owner of the matched file watcher.
<code>name</code>	Name of the matched file watcher.
<code>requested_path_name</code>	Value of the <code>directory_path</code> attribute of the matched file watcher.
<code>requested_file_name</code>	Value of the <code>file_name</code> attribute of the matched file watcher.
<code>credential_owner</code>	Owner of the credential referenced by the matched file watcher.
<code>credential_name</code>	Name of the credential referenced by the matched file watcher.
<code>min_file_size</code>	Value of the <code>min_file_size</code> attribute of the matched file watcher.
<code>steady_state_dur</code>	Value of the <code>steady_state_duration</code> attribute of the matched file watcher.

## Using DBMS\_SCHEDULER

This section contains:

- [Security Model](#)
- [Rules and Limits](#)
- [Operational Notes](#)

## Security Model

The `DBMS_SCHEDULER` package ignores privileges granted on scheduler objects, such as jobs or chains, through roles. Object privileges must be granted directly to the user.

## Rules and Limits

The following rules apply when using the DBMS\_SCHEDULER package:

- Only SYS can perform actions on objects in the SYS schema.
- Several of the procedures accept comma-delimited lists of object names. If you provide a list of names, then the Scheduler stops executing the list at the first object that returns an error. Therefore, the Scheduler does not perform the tasks needed for the remaining objects on the list.

For example, consider the statement `DBMS_SCHEDULER.STOP_JOB ('job1, job2, job3, sys.jobclass1, sys.jobclass2, sys.jobclass3');`

If job3 cannot be stopped, then the jobs that follow it, jobclass1, jobclass2, and jobclass3 cannot be stopped. The jobs that preceded job3, job1 and job2, are stopped.

- Performing an action on an object that does not exist returns a PL/SQL exception stating that the object does not exist.

## Operational Notes

The Scheduler uses a rich **calendar**ing syntax to enable you to define repeating schedules, such as "every Tuesday and Friday at 4:00 p.m." or "the second Wednesday of every month." This calendaring syntax is used in calendaring expressions in the `repeat_interval` argument of a number of package subprograms. Evaluating a calendaring expression results in a set of discrete timestamps.

See *Oracle Database Administrator's Guide* for examples of the calendaring syntax.

## Calendaring Syntax

This section starts with the calendaring syntax. It is followed by descriptions of various parts of the syntax.

In the calendaring syntax, \* means 0 or more.

```
repeat_interval = regular_schedule | combined_schedule

regular_schedule = frequency_clause
[";" interval_clause] [";" bymonth_clause] [";" byweekno_clause]
[";" byyearday_clause] [";" bydate_clause] [";" bymonthday_clause]
[";" byday_clause] [";" byhour_clause] [";" byminute_clause]
[";" bysecond_clause] [";" bysetpos_clause] [";" include_clause]
[";" exclude_clause] [";" intersect_clause] [";" periods_clause]
[";" byperiod_clause]

combined_schedule = schedule_list

frequency_clause = "FREQ" "=" ( predefined_frequency | user_defined_frequency )
predefined_frequency = "YEARLY" | "MONTHLY" | "WEEKLY" | "DAILY" |
    "HOURLY" | "MINUTELY" | "SECONDLY"
user_defined_frequency = named_schedule

interval_clause = "INTERVAL" "=" intervalnum
    intervalnum = 1 through 99
bymonth_clause = "BYMONTH" "=" monthlist
    monthlist = month ( "," month)*
    month = numeric_month | char_month
    numeric_month = 1 | 2 | 3 ... 12
    char_month = "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
        "JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"
byweekno_clause = "BYWEEKNO" "=" weeknumber_list
    weeknumber_list = weeknumber ( "," weeknumber)*
    weeknumber = [minus] weekno
    weekno = 1 through 53
byyearday_clause = "BYYEARDAY" "=" yearday_list
    yearday_list = yearday ( "," yearday)*
    yearday = [minus] yeardaynum
    yeardaynum = 1 through 366
bydate_clause = "BYDATE" "=" date_list
    date_list = date ( "," date)*
    date = [YYYY]MMDD [ offset | span ]
bymonthday_clause = "BYMONTHDAY" "=" monthday_list
    monthday_list = monthday ( "," monthday)*
    monthday = [minus] monthdaynum
    monthdaynum = 1 through 31
byday_clause = "BYDAY" "=" byday_list
```



```

byday_list = byday ( "," byday)*
byday = [weekdaynum] day
weekdaynum = [minus] daynum
daynum = 1 through 53 /* if frequency is yearly */
daynum = 1 through 5 /* if frequency is monthly */
day = "MON" | "TUE" | "WED" | "THU" | "FRI" | "SAT" | "SUN"
byhour_clause = "BYHOUR" "=" hour_list
hour_list = hour ( "," hour)*
hour = 0 through 23
byminute_clause = "BYMINUTE" "=" minute_list
minute_list = minute ( "," minute)*
minute = 0 through 59
bysecond_clause = "BYSECOND" "=" second_list
second_list = second ( "," second)*
second = 0 through 59
bysetpos_clause = "BYSETPOS" "=" setpos_list
setpos_list = setpos ( "," setpos)*
setpos = [minus] setpos_num
setpos_num = 1 through 9999

include_clause = "INCLUDE" "=" schedule_list
exclude_clause = "EXCLUDE" "=" schedule_list
intersect_clause = "INTERSECT" "=" schedule_list
schedule_list = schedule_clause ( "," schedule_clause)*
schedule_clause = named_schedule [ offset ]
named_schedule = [schema "."] schedule
periods_clause = "PERIODS" "=" periodnum
byperiod_clause = "BYPERIOD" "=" period_list
period_list = periodnum ( "," periodnum)*
periodnum = 1 through 100

offset = ("+" | "-") ["OFFSET:"] duration_val
span = ("+" | "-" | "^") "SPAN:" duration_val
duration_val = dur-weeks | dur_days
dur_weeks = numofweeks "W"
dur_days = numofdays "D"
numofweeks = 1 through 53
numofdays = 1 through 376
minus = "-"

```

**Table 128–10 Values for repeat\_interval**

Name	Description
FREQ	This specifies the type of recurrence. It must be specified. The possible predefined frequency values are YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY. Alternatively, specifies an existing schedule to use as a user-defined frequency.
INTERVAL	This specifies a positive integer representing how often the recurrence repeats. The default is 1, which means every second for secondly, every day for daily, and so on. The maximum value is 99.
BYMONTH	This specifies which month or months you want the job to execute in. You can use numbers such as 1 for January and 3 for March, as well as three-letter abbreviations such as FEB for February and JUL for July.

**Table 128–10 (Cont.) Values for repeat\_interval**

Name	Description
BYWEEKNO	<p>This specifies the week of the year as a number. It follows ISO-8601, which defines the week as starting with Monday and ending with Sunday; and the first week of a year as the first week, which is mostly within the Gregorian year. The first week is equivalent to the following two variants: the week that contains the first Thursday of the Gregorian year; and the week containing January 4th.</p> <p>The ISO-8601 week numbers are integers from 1 to 52 or 53; parts of week 1 may be in the previous calendar year; parts of week 52 may be in the following calendar year; and if a year has a week 53, parts of it must be in the following calendar year.</p> <p>As an example, in the year 1998, the ISO week 1 began on Monday December 29th, 1997; and the last ISO week (week 53) ended on Sunday January 3rd, 1999. So December 29th, 1997, is in the ISO week 1998-01, and January 1st, 1999, is in the ISO week 1998-53.</p> <p>byweekno is only valid for YEARLY.</p> <p>Examples of invalid specifications are "FREQ=YEARLY; BYWEEKNO=1; BYMONTH=12" and "FREQ=YEARLY; BYWEEKNO=53; BYMONTH=1".</p>
BYYEARDAY	<p>This specifies the day of the year as a number. Valid values are 1 to 366. An example is 69, which is March 10 (31 for January, 28 for February, and 10 for March). 69 evaluates to March 10 for non-leap years and March 9 in leap years. -2 will always evaluate to December 30th independent of whether it is a leap year.</p>
BYDATE	<p>This specifies a list of dates, where each date is of the form [YYYY]MMDD. A list of consecutive dates can be generated by using the SPAN modifier, and a date can be adjusted with the OFFSET modifier. An example of a simple BYDATE clause follows:</p> <pre data-bbox="537 1056 1110 1077">BYDATE=0115,0315,0615,0915,1215,20060115</pre> <p>The following SPAN example is equivalent to  BYDATE=0110,0111,0112,0113,0114, which is a span of 5 days starting at 1/10:</p> <pre data-bbox="537 1186 813 1207">BYDATE=0110+SPAN:5D</pre> <p>The plus sign in front of the SPAN keyword indicates a span starting at the supplied date. The minus sign indicates a span ending at the supplied date, and the "^" sign indicates a span of <i>n</i> days or weeks centered around the supplied date. If <i>n</i> is an even number, it is adjusted up to the next odd number.</p> <p>Offsets adjust the supplied date by adding or subtracting <i>n</i> days or weeks. BYDATE=0205-OFFSET:2W is equivalent to BYDATE=0205-14D (the OFFSET: keyword is optional), which is also equivalent to BYDATE=0122.</p>
BYMONTHDAY	<p>This specifies the day of the month as a number. Valid values are 1 to 31. An example is 10, which means the 10th day of the selected month. You can use the minus sign (-) to count backward from the last day, so, for example, BYMONTHDAY=-1 means the last day of the month and BYMONTHDAY=-2 means the next to last day of the month.</p>
BYDAY	<p>This specifies the day of the week from Monday to Sunday in the form MON, TUE, and so on. Using numbers, you can specify the 26th Friday of the year, if using a YEARLY frequency, or the 4th THU of the month, using a MONTHLY frequency. Using the minus sign, you can say the second to last Friday of the month. For example, -1 FRI is the last Friday of the month.</p>
BYHOUR	<p>This specifies the hour on which the job is to run. Valid values are 0 to 23. As an example, 10 means 10 a.m.</p>
BYMINUTE	<p>This specifies the minute on which the job is to run. Valid values are 0 to 59. As an example, 45 means 45 minutes past the chosen hour.</p>

**Table 128–10 (Cont.) Values for repeat\_interval**

Name	Description
BYSECOND	<p>This specifies the second on which the job is to run. Valid values are 0 to 59. As an example, 30 means 30 seconds past the chosen minute.</p>
BYSETPOS	<p>This selects one or more items, by position, in the list of timestamps that result after the whole calendaring expression is evaluated. It is useful for requirements such as running a job on the last workday of the month. Rather than attempting to express this with the other BY clauses, you can code the calendaring expression to evaluate to a list of every workday of the month, and then add the BYSETPOS clause to select only the last item of that list. Assuming that workdays are Monday through Friday, the syntax would then be:</p> <pre data-bbox="618 562 1260 583">FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1</pre> <p>Valid values are 1 through 9999. A negative number selects an item from the end of the list (-1 is the last item, -2 is the next to last item, and so on) and a positive number selects from the front of the list. The BYSETPOS clause is always evaluated last. BYSETPOS is only supported with the MONTHLY and YEARLY frequencies.</p> <p>The BYSETPOS clause is applied to the list of timestamps once per frequency period. For example, when the frequency is defined as MONTHLY, the Scheduler determines all valid timestamps for the month, orders that list, and then applies the BYSETPOS clause. The Scheduler then moves on to the next month and repeats the procedure. Assuming a start date of Jun 10, 2004, the example evaluates to: Jun 30, Jul 30, Aug 31, Sep 30, Oct 29, and so on.</p>
INCLUDE	<p>This includes one or more named schedules in the calendaring expression. That is, the set of timestamps defined by each included named schedule is added to the results of the calendaring expression. If an identical timestamp is contributed by both an included schedule and the calendaring expression, it is included in the resulting set of timestamps only once. The named schedules must have been defined with the CREATE_SCHEDULE procedure.</p>
EXCLUDE	<p>This excludes one or more named schedules from the calendaring expression. That is, the set of timestamps defined by each excluded named schedule is removed from the results of the calendaring expression. The named schedules must have been defined with the CREATE_SCHEDULE procedure.</p>
INTERSECT	<p>This specifies an intersection between the calendaring expression results and the set of timestamps defined by one or more named schedules. Only the timestamps that appear both in the calendaring expression and in one of the named schedules are included in the resulting set of timestamps.</p> <p>For example, assume that the named schedule <code>last_sat</code> indicates the last Saturday in every month, and that for the year 2005, the only months where the last day of the month is also a Saturday are April and December. Assume also that the named schedule <code>end_qtr</code> indicates the last day of each quarter in 2005:</p> <pre data-bbox="618 1566 1146 1587">3/31/2005, 6/30/2005, 9/30/2005, 12/31/2005</pre> <p>These calendaring expressions result in the dates that follow:</p> <pre data-bbox="618 1671 1284 1692">3/31/2005, 4/30/2005, 6/30/2005, 9/30/2005, 12/31/2005</pre> <pre data-bbox="618 1734 1292 1755">FREQ=MONTHLY; BYMONTHDAY=-1; INTERSECT=last_sat,end_qtr</pre> <p>In this example, the terms <code>FREQ=MONTHLY; BYMONTHDAY=-1</code> indicate the last day of each month.</p>

**Table 128–10 (Cont.) Values for repeat\_interval**

Name	Description
PERIODS	This identifies the number of periods that together form one cycle of a user-defined frequency. It is used in the <code>repeat_interval</code> expression of the schedule that defines the user-defined frequency. It is mandatory when the <code>repeat_interval</code> expression in the main schedule contains a <code>BYPERIOD</code> clause. The following example defines the quarters of a fiscal year.  FREQ=YEARLY;BYDATE=0301,0601,0901,1201;PERIODS=4
BYPERIOD	This selects periods from a user-defined frequency. For example, if a main schedule names a user-defined frequency schedule that defines the fiscal quarters shown in the previous example, the clause <code>BYPERIOD=2,4</code> in the main schedule selects the 2nd and 4th fiscal quarters.

**Combining Schedules** There are two ways to combine schedules:

- Using a combined schedule expression, which is a list of individual schedules  
For example, to create a schedule for all company holidays, you provide a list of individual schedules, where each schedule in the list defines a single holiday. The Scheduler evaluates each individual schedule, and then returns a union of the timestamps returned by each individual schedule.
- Embedding other schedules into the main schedule using `include`, `exclude`, and `intersect` clauses

With this method, the embedded schedules inherit certain attributes from the main schedule.

- Timestamps generated by the `INCLUDE` clause that fall into periods that are skipped by the main schedule are ignored. This is the case when the main schedule skips periods due to the `INTERVAL` clause, the `BYPERIOD` clause, or the `BYMONTH` clause for `freq=monthly`.
- Days that are added by the `INCLUDE` clause follow the hourly/ minutely/secondly execution pattern of the main schedule.
- When the `INCLUDE` clause is present, no date-specific defaults are retrieved from the start date (but time-specific defaults can be). (See "[Start Dates and Repeat Intervals](#)", later in this section.) For example, a `repeat_interval` of `FREQ=MONTHLY; INCLUDE=HOLIDAY` executes only on holidays and not on the month/day defaults retrieved from the start date.

The following is an example:

```
BEGIN
dbms_scheduler.create_schedule('embed_sched', repeat_interval =>
'FREQ=YEARLY;BYDATE=0130,0220,0725');
dbms_scheduler.create_schedule('main_sched', repeat_interval =>
'FREQ=MONTHLY;INTERVAL=2;BYMONTHDAY=15;BYHOUR=9,17;INCLUDE=embed_sched');
END;
/
```

In this example, the dates 1/30, 2/20, and 7/25 are added to the main schedule. However, the Scheduler does not include dates that fall in months that are skipped by the `INTERVAL` clause. If the start date of the main schedule is 1/1/2005, then 2/20 is not added. On the dates that are added, the embedded schedule follows the execution pattern of the main schedule: jobs are executed at 9:00 a.m. and 5:00 p.m. on 1/30 and 7/25. If the embedded schedule does not itself have a start date, it inherits the start date from the main schedule.

**User-Defined Frequencies** Instead of using predefined frequencies like DAILY, WEEKLY, MONTHLY, and so on, you can create your own frequencies by creating a schedule that returns the start date of each period. For example, the following `repeat_interval` expression is used in a schedule named `fiscal_year` that defines the start of each quarter in a fiscal year:

```
FREQ=YEARLY;BYDATE=0301,0601,0901,1201;PERIODS=4
```

To return the last Wednesday of every quarter, you create a schedule (the "main schedule") that uses the `fiscal_year` schedule as a user-defined frequency:

```
FREQ=fiscal_year;BYDAY=-1WED
```

Periods in a user-defined frequency do not have to be equal in length. In the main schedule, the `BYSETPOS` clause and numbered weekdays are recalculated based on the size of each period. To select dates in specific periods, you must use the `BYPERIOD` clause in the main schedule. To enable this, the schedule that is used as the user-defined frequency must include a `PERIODS` clause, and it must set its start date appropriately. The first date returned by this schedule is used as the starting point of period 1.

As another example, assuming work days are Monday through Friday, to get the last work day of the 2nd and 4th quarters of the fiscal year, the `repeat_interval` clause in the main schedule is the following:

```
FREQ=fiscal_year;BYDAY=MON,TUE,WED,THU,FRI;BYPERIOD=2,4;BYSETPOS=-1
```

**Start Dates and Repeat Intervals** The Scheduler retrieves the date and time from the job or schedule start date and incorporates them as defaults into the `repeat_interval`. For example, if the specified frequency is yearly and there is no `BYMONTH` or `BYMONTHDAY` clause in the repeat interval, then the month and day that the job runs on are retrieved from the start date. Similarly, if frequency is monthly but there is no `BYMONTHDAY` clause in the repeat interval, then the day of the month that the job runs on is retrieved from the start date. If present, `BYHOUR`, `BYMINUTE`, and `BYSECOND` defaults are also retrieved from the start date, and used if those clauses are not specified. Note that if the `INCLUDE`, `EXCLUDE`, or `INTERSECT` clauses are present, no date-related defaults are retrieved from the start date, but time-related defaults are.

The following are some examples:

```
start_date:      4/15/05 9:00:00
repeat_interval: freq=yearly
```

is expanded internally to:

```
freq=yearly;bymonth=4;bymonthday=15;byhour=9;byminute=0;bysecond=0
```

The preceding schedule executes on 04/15/05 9:00:00, 04/15/06 9:00:00, 04/15/07 9:00:00, and so on.

For the next example, assume that schedule S1 has a `repeat_interval` of `FREQ=YEARLY;BYDATE=0701`.

```
start_date:      01/20/05 9:00:00
repeat_interval: freq=yearly;include=S1
```

is expanded internally to:

```
freq=yearly;byhour=9;byminute=0;bysecond=0;include=S1
```

Because an `INCLUDE` clause is present, date-related information is not retrieved from the start date. However, time-specific information is, so the preceding schedule executes on 07/01/05 9:00:00, 07/01/06 9:00:00, 07/01/08 9:00:00, and so on.

**General Rules** When using a calendaring expression, consider the following rules:

- For a regular schedule (as opposed to a combined schedule), the calendar string must start with the frequency clause. All other clauses are optional and can be put in any order.
- All clauses are separated by a semicolon, and each clause can be present at most once, with the exception of the `include`, `exclude`, and `intersect` clauses.
- Spaces are allowed between syntax elements and the strings are case-insensitive.
- The list of values for a specific `BY` clause do not need to be ordered.
- When not enough `BY` clauses are present to determine what the next date is, this information is retrieved from the start date. For example, "`FREQ=YEARLY`" with a start date of 02/15/2003 becomes "`FREQ=YEARLY ; BYMONTH=FEB ; BYMONTHDAY=15`", which means every year on the 15th of February.  
"`FREQ=YEARLY ; BYMONTH=JAN, JUL`" with start date 01/21/2003 becomes "`FREQ=YEARLY ; BYMONTH=JAN, JUL ; BYMONTHDAY=21`", which means every year on January 21 and July 21.
- The `byweekno` clause is only allowed if the frequency is `YEARLY`. It cannot be used with other frequencies. When it is present, it will return all days in that week number. If you want to limit it to specific days within the week, you have to add a `BYDAY` clause. For example, "`FREQ=YEARLY ; BYWEEKNO=2`" with a start date of 01/01/2003 will return:

```
01/06/2003, 01/07/2003, 01/08/2003, 01/09/2003, 01/10/2003, 01/11/2003,
01/12/2003, 01/05/2004, 01/06/2004, 01/07/2004, ... and so on.
```

Note that when the `byweekno` clause is used, it is possible that the dates returned are from a year other than the current year. For example, if returning dates for the year 2004 and the calendar string is "`FREQ=YEARLY ; BYWEEKNO=1, 53`" for the specified week numbers in 2004, it will return the dates:

```
12/29/03, 12/30/03, 12/31/03, 01/01/04, 01/02/04, 01/03/04, 01/04/04, 12/27/04,
12/28/04, 12/29/04, 12/30/04, 12/31/04, 01/01/05, 01/02/05
```

- For those `BY` clauses that do not have a consistent range of values, you can count backward by putting a "-" in front of the numeric value. For example, specifying `BYMONTHDAY=31` will not give you the last day of every month, because not every month has 31 days. Instead, `BYMONTHDAY=-1` will give you the last day of the month.

This is not supported for `BY` clauses that are fixed in size. In other words, `BYMONTH`, `BYHOUR`, `BYMINUTE`, and `BYSECOND` are not supported.

- The basic values for the `BYDAY` clause are the days of the week. When the frequency is `YEARLY`, or `MONTHLY`, you are allowed to specify a positive or negative number in front of each day of the week. In the case of `YEARLY`, `BYDAY=40MON`, indicates the 40th Monday of the year. In the case of `MONTHLY`, `BYDAY=-2SAT`, indicates the second to last Saturday of the month.

Note that positive or negative numbers in front of the weekdays are not supported for other frequencies and that in the case of yearly, the number ranges from -53 ... -1, 1 ... 53, whereas for the monthly frequency it is limited to -5 ... -1, 1... 5.

If no number is present in front of the weekday it specifies, every occurrence of that weekday in the specified frequency.

- The first day of the week is Monday.
- Repeating jobs with frequencies smaller than daily follow their frequencies exactly across daylight savings adjustments. For example, suppose that a job is scheduled to repeat every 3 hours, the clock is moved forward from 1:00 a.m. to 2:00 a.m., and the last time the job ran was midnight. Its next scheduled time will be 4:00 a.m. Thus, the 3 hour period between subsequent job runs is retained. The same applies when the clock is moved back. This behavior is not the case for repeating jobs that have frequencies of daily or larger. For example, if a repeating job is supposed to be executed on a daily basis at midnight, it will continue to run at midnight if the clock is moved forward or backward. When the execution time of such a daily (or larger frequency) job happens to fall inside a window where the clock is moved forward, the job executes at the end of the window.
- The calendaring syntax does not allow you to specify a time zone. Instead the Scheduler retrieves the time zone from the `start_date` argument. If jobs must follow daylight savings adjustments, then you must specify a region name for the time zone of the `start_date`. For example specifying the `start_date` time zone as 'US/Eastern' in New York ensures that daylight saving adjustments are automatically applied. If instead, the time zone of the `start_date` is set to an absolute offset, such as '-5:00', then daylight savings adjustments are not followed and your job execution is off by an hour for half the year.
- When `start_date` is NULL, the Scheduler determines the time zone for the repeat interval as follows:
  1. It checks whether or not the session time zone is a region name. The session time zone can be set by either:
    - Issuing an ALTER SESSION statement, for example:
 

```
SQL> ALTER SESSION SET time_zone = 'Asia/Shanghai';
```
    - Setting the ORA\_SDTZ environment variable.
  2. If the session time zone is an absolute offset instead of a region name, the Scheduler uses the value of the DEFAULT\_TIMEZONE Scheduler attribute. For more information, see the [SET\\_SCHEDULER\\_ATTRIBUTE Procedure](#).
  3. If the DEFAULT\_TIMEZONE attribute is NULL, the Scheduler uses the time zone of `sysimestamp` when the job or window is enabled.

**BYSETPOS Clause Rules** The following are rules for the BYSETPOS clause.

- The BYSETPOS clause is the last clause to be evaluated. It is processed after all other BY clauses and the INCLUDE, EXCLUDE and INTERSECT clauses have been evaluated.
- The INTERVAL clause does not change the size of the period to which the BYSETPOS clause is applied. For example, when the frequency is set to monthly and interval is set to 3, the list of timestamps to which BYSETPOS is applied is generated from a month, not a quarter. The only impact of the INTERVAL clause is to cause months to be skipped. However, you can still select the second to last workday of the quarter like this:

```
FREQ=MONTHLY; INTERVAL=3; BYDAY=MON, TUE, WED, THU, FRI; BYSETPOS=-2
```

provided that you set the start date in the right month. This example returns the next to last workday of a month, and repeats once a quarter.

- To get consistent results, the set to which BYSETPOS is applied is determined from the beginning of the frequency period independently of when the evaluation occurs. Whether the Scheduler evaluates

```
FREQ=MONTHLY;BYDAY=MON,TUE,FRI;BYSETPOS=1,3
```

on 01/01/2004 or 01/15/2004, in both cases the expression evaluates to Friday 01/02/2004, and Tuesday 01/06/2004. The only difference is that when the expression is evaluated on 01/15/2004, the Scheduler determines that there are no matches in January because the timestamps found are in the past, and it moves on to the matches in the next month, February.

**BYDATE Clause Rules** The following are rules for the BYDATE clause.

- If dates in the BYDATE clause do not have their optional year component, the job runs on those dates every year.
- The job execution times on the included dates are derived from the BY clauses in the calendaring expression. For example, if `repeat_interval` is defined as

```
freq=daily;byhour=8,13,18;byminute=0;bysecond=0;bydate=0502,0922
```

then the execution times on 05/02 and 09/22 are 8:00 a.m., 1:00 p.m., and 6:00 p.m.

**EXCLUDE Clause Rules** Excluded dates without a time component are 24 hour periods. All timestamps that fall on an excluded date are removed. In the following example, `jan_fifteen` is a named schedule that resolves to the single date of 01/15:

```
freq=monthly;bymonthday=15,30;byhour=8,13,18;byminute=0;bysecond=0;
exclude=jan_fifteenth
```

In this case, all three instances of the job are removed for 01/15.

**OFFSET Rules** You can adjust the dates of individual named schedules by adding positive offsets to them. For example, to execute `JOB2` exactly 15 days after every occurrence of `JOB1`, add `+OFFSET:15D` to the schedule of `JOB1`, as follows:

```
BEGIN
dbms_scheduler.create_schedule('job2_schedule', repeat_interval =>
'job1_schedule+OFFSET:15D');
END;
/
```

Note that negative offsets to named schedules are not supported.

### **Example 128–1 Putting It All Together**

This example demonstrates the use of user-defined frequencies, spans, offsets, and the BYSETPOS and INCLUDE clauses. (Note that the `OFFSET:` keyword is optional in an offset clause.)

Many companies in the retail industry share the same fiscal year. The fiscal year starts on the Sunday closest to February 1st, and subsequent quarters start exactly 13 weeks later. The fiscal year schedule for the retail industry can be defined as the following:

```
begin
dbms_scheduler.create_schedule('year_start', repeat_interval=>
'FREQ=YEARLY;BYDATE=0201^SPAN:1W;BYDAY=SUN');
dbms_scheduler.create_schedule('retail_fiscal_year',
to_timestamp_tz('15-JAN-2005 12:00:00','DD-MON-YYYY HH24:MI:SS'),
'year_start,year_start+13w,year_start+26w,year_start+39w;periods=4');
```



```
end;  
/
```

The following schedule can be used to execute a job on the 5th day off in the 2nd and the 4th quarters of the retail industry. This assumes that Saturday and Sunday are off days as well as the days in the existing holiday schedule.

```
begin  
  dbms_scheduler.create_schedule('fifth_day_off', repeat_interval=>  
    'FREQ=retail_fiscal_year;BYDAY=SAT,SUN;INCLUDE=holiday;  
    BYPERIOD=2,4;BYSETPOS=5');  
end;  
/
```

---

## Summary of DBMS\_SCHEDULER Subprograms

**Table 128–11 DBMS\_SCHEDULER Package Subprograms**

Subprogram	Description
<a href="#">ADD_EVENT_QUEUE_SUBSCRIBER Procedure</a> on page 128-34	Adds a user as a subscriber to the Scheduler event queue SYS.SCHEDULER\$_EVENT_QUEUE
<a href="#">ADD_GROUP_MEMBER Procedure</a> on page 128-35	Adds one or more members to an existing group
<a href="#">ADD_JOB_EMAIL_NOTIFICATION Procedure</a> on page 128-36	Adds e-mail notifications for a job for a list of recipients and a list of job state events
<a href="#">ADD_WINDOW_GROUP_MEMBER Procedure</a> on page 128-39	***Deprecated. Use CREATE_GROUP and ADD_GROUP_MEMBER instead
<a href="#">ALTER_CHAIN Procedure</a> on page 128-40	Alters specified steps of a chain
<a href="#">ALTER_RUNNING_CHAIN Procedure</a> on page 128-43	Alters specified steps of a running chain
<a href="#">CLOSE_WINDOW Procedure</a> on page 128-45	Closes an open window prematurely
<a href="#">COPY_JOB Procedure</a> on page 128-46	Copies an existing job
<a href="#">CREATE_CHAIN Procedure</a> on page 128-47	Creates a chain, which is a named series of programs that are linked together for a combined objective
<a href="#">CREATE_CREDENTIAL Procedure</a> on page 128-49	Creates a credential
<a href="#">CREATE_DATABASE_DESTINATION Procedure</a> on page 128-50	Creates a database destination for use with remote database jobs
<a href="#">CREATE_EVENT_SCHEDULE Procedure</a> on page 128-51	Creates an event schedule, which is a schedule that starts a job based on the detection of an event
<a href="#">CREATE_FILE_WATCHER Procedure</a> on page 128-53	Creates a file watcher, which is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job
<a href="#">CREATE_GROUP Procedure</a> on page 128-55	Creates a group
<a href="#">CREATE_JOB Procedure</a> on page 128-57	Creates a single job
<a href="#">CREATE_JOB_CLASS Procedure</a> on page 128-64	Creates a job class, which provides a way to group jobs for resource allocation and prioritization
<a href="#">CREATE_JOBS Procedure</a> on page 128-66	Creates multiple jobs
<a href="#">CREATE_PROGRAM Procedure</a> on page 128-67	Creates a program
<a href="#">CREATE_SCHEDULE Procedure</a> on page 128-70	Creates a schedule
<a href="#">CREATE_WINDOW Procedure</a> on page 128-71	Creates a window, which provides a way to automatically activate different resource plans at different times
<a href="#">CREATE_WINDOW_GROUP Procedure</a> on page 128-74	*** Deprecated. Use CREATE_GROUP

**Table 128–11 (Cont.) DBMS\_SCHEDULER Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DEFINE_ANYDATA_ARGUMENT Procedure</a> on page 128-75	Defines a program argument whose value is of a complex type and must be passed encapsulated in an AnyData object
<a href="#">DEFINE_CHAIN_EVENT_STEP Procedure</a> on page 128-76	Adds or replaces a chain step and associates it with an event schedule or inline event. See also: <a href="#">DEFINE_CHAIN_STEP</a> .
<a href="#">DEFINE_CHAIN_RULE Procedure</a> on page 128-77	Adds a rule to an existing chain
<a href="#">DEFINE_CHAIN_STEP Procedure</a> on page 128-80	Defines a chain step, which can be a program or another (nested) chain. See also: <a href="#">DEFINE_CHAIN_EVENT_STEP</a> .
<a href="#">DEFINE_METADATA_ARGUMENT Procedure</a> on page 128-81	Defines a special metadata argument for the program. You can retrieve specific metadata through this argument.
<a href="#">DEFINE_PROGRAM_ARGUMENT Procedure</a> on page 128-83	Defines a program argument whose value can be passed as a string literal to the program
<a href="#">DISABLE Procedure</a> on page 128-85	Disables a program, job, chain, window, database destination, external destination, file watcher, or group
<a href="#">DROP_AGENT_DESTINATION Procedure</a> on page 128-88	Drops one or more external destinations. Use only when the preferred method of dropping external destinations—unregistering the Scheduler agent with the database—fails.
<a href="#">DROP_CHAIN Procedure</a> on page 128-89	Drops an existing chain
<a href="#">DROP_CHAIN_RULE Procedure</a> on page 128-90	Removes a rule from an existing chain
<a href="#">DROP_CHAIN_STEP Procedure</a> on page 128-91	Drops a chain step
<a href="#">DROP_CREDENTIAL Procedure</a> on page 128-92	Drops a credential
<a href="#">DROP_DATABASE_DESTINATION Procedure</a> on page 128-93	Drops one or more database destinations
<a href="#">DROP_FILE_WATCHER Procedure</a> on page 128-94	Drops one or more file watchers
<a href="#">DROP_GROUP Procedure</a> on page 128-95	Drops one or more groups
<a href="#">DROP_JOB Procedure</a> on page 128-96	Drops a job or all jobs in a job class
<a href="#">DROP_JOB_CLASS Procedure</a> on page 128-97	Drops a job class
<a href="#">DROP_PROGRAM Procedure</a> on page 128-98	Drops a program
<a href="#">DROP_PROGRAM_ARGUMENT Procedure</a> on page 128-99	Drops a program argument
<a href="#">DROP_SCHEDULE Procedure</a> on page 128-100	Drops a schedule
<a href="#">DROP_WINDOW Procedure</a> on page 128-101	Drops a window
<a href="#">DROP_WINDOW_GROUP Procedure</a> on page 128-102	*** Deprecated. Use <a href="#">CREATE_GROUP</a> and <a href="#">DROP_GROUP</a> instead.

**Table 128–11 (Cont.) DBMS\_SCHEDULER Package Subprograms**

Subprogram	Description
<a href="#">ENABLE Procedure</a> on page 128-103	Enables a program, job, chain, window, database destination, external destination, file watcher, or group
<a href="#">END_DETACHED_JOB_RUN Procedure</a> on page 128-105	Ends a running detached job
<a href="#">EVALUATE_CALENDAR_STRING Procedure</a> on page 128-106	Evaluates the calendar string and tells you what the next execution date of a job or window will be
<a href="#">EVALUATE_RUNNING_CHAIN Procedure</a> on page 128-108	Forces reevaluation of the rules of a running chain to trigger any rules for conditions that have been satisfied
<a href="#">GENERATE_JOB_NAME Function</a> on page 128-109	Generates a unique name for a job. This enables you to identify jobs by adding a prefix, so, for example, Sally's jobs would be named <code>sally1</code> , <code>sally2</code> , and so on
<a href="#">"GET_AGENT_INFO Function"</a> on page 128-110	Returns job information specific to an agent, such as how many are running and so on, depending on the attribute selected
<a href="#">"GET_AGENT_VERSION Function"</a> on page 128-111	Returns the version string of a Scheduler agent that is registered with the database and is currently running
<a href="#">GET_ATTRIBUTE Procedure</a> on page 128-112	Retrieves the value of an attribute of an object
<a href="#">GET_FILE Procedure</a> on page 128-113	Retrieves a file from a host
<a href="#">GET_SCHEDULER_ATTRIBUTE Procedure</a> on page 128-115	Retrieves the value of a Scheduler attribute
<a href="#">OPEN_WINDOW Procedure</a> on page 128-116	Opens a window prematurely. The window is opened immediately for the duration
<a href="#">PURGE_LOG Procedure</a> on page 128-118	Purges specific rows from the job and window logs
<a href="#">PUT_FILE Procedure</a> on page 128-119	Saves a file to one or more hosts
<a href="#">REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure</a> on page 128-120	Unsubscribes a user from the Scheduler event queue <code>SYS.SCHEDULER\$_EVENT_QUEUE</code>
<a href="#">REMOVE_GROUP_MEMBER Procedure</a> on page 128-121	Removes one or more members from a group
<a href="#">REMOVE_JOB_EMAIL_NOTIFICATION Procedure</a> on page 128-122	Removes e-mail notifications for a job
<a href="#">REMOVE_WINDOW_GROUP_MEMBER Procedure</a> on page 128-123	***Deprecated. Use <code>CREATE_GROUP</code> and <code>REMOVE_GROUP_MEMBER</code> instead
<a href="#">RESET_JOB_ARGUMENT_VALUE Procedure</a> on page 128-124	Resets the current value assigned to an argument defined with the associated program
<a href="#">RUN_CHAIN Procedure</a> on page 128-125	Immediately runs a chain by creating a run-once job
<a href="#">RUN_JOB Procedure</a> on page 128-127	Runs a job immediately
<a href="#">SET_AGENT_REGISTRATION_PASS Procedure</a> on page 128-129	Sets the agent registration password for a database
<a href="#">SET_ATTRIBUTE Procedure</a> on page 128-130	Changes an attribute of a job, schedule, or other Scheduler object

**Table 128-11 (Cont.) DBMS\_SCHEDULER Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SET_ATTRIBUTE_NULL Procedure</a> on page 128-145	Changes an attribute of an object to NULL
<a href="#">SET_JOB_ANYDATA_VALUE Procedure</a> on page 128-146	Sets the value of a job argument encapsulated in an AnyData object
<a href="#">SET_JOB_ARGUMENT_VALUE Procedure</a> on page 128-147	Sets the value of a job argument
<a href="#">SET_JOB_ATTRIBUTES Procedure</a> on page 128-149	Sets the value of a job attribute
<a href="#">SET_SCHEDULER_ATTRIBUTE Procedure</a> on page 128-150	Sets the value of a Scheduler attribute
<a href="#">STOP_JOB Procedure</a> on page 128-153	Stops a currently running job or all jobs in a job class

## ADD\_EVENT\_QUEUE\_SUBSCRIBER Procedure

This procedure adds a user as a subscriber to the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`, and grants the user permission to dequeue from this queue using the designated agent.

### Syntax

```
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER (  
    subscriber_name          IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–12 ADD\_EVENT\_QUEUE\_SUBSCRIBER Procedure Parameters**

Parameter	Description
<code>subscriber_name</code>	Name of the Oracle Streams Advanced Queuing (AQ) agent to be used to subscribe to the Scheduler event queue. If <code>NULL</code> , an agent is created and assigned the user name of the calling user.

### Usage Notes

The subscription is rule-based. The rule permits the user to see only events raised by jobs that the user owns, and filters out all other messages. If an AQ agent with the same name already exists, an error is raised.

## ADD\_GROUP\_MEMBER Procedure

This procedure adds one or more members to an existing group.

### Syntax

```
DBMS_SCHEDULER.ADD_GROUP_MEMBER (
  group_name          IN VARCHAR2,
  member              IN VARCHAR2);
```

### Parameters

**Table 128–13** ADD\_GROUP\_MEMBER Procedure Parameters

Parameter	Description
group_name	The name of the group.
member	<p>A comma-separated list of members to add to the group. Members must match the group type. A group of the same type can be a member. The Scheduler immediately expands the included group name into its list of members.</p> <p>An error is returned if any of the members do not exist. A member that is already in the group is skipped, and no error is generated.</p> <p>The keyword LOCAL can be included as a member for database destination or external destination groups. See the "<a href="#">CREATE_GROUP Procedure</a>" on page 55 for information about this keyword.</p>

### Usage Notes

The following users may add members to a group:

- The group owner
- A user that has been granted the ALTER object privilege on the group
- A user with the CREATE ANY JOB system privilege

You must have the MANAGE SCHEDULER privilege to add a member to a group of type WINDOW.

**See Also:** "[CREATE\\_GROUP Procedure](#)" on page 128-55

## ADD\_JOB\_EMAIL\_NOTIFICATION Procedure

This procedure adds e-mail notifications for a job. E-mails are then sent to the specified list of recipients whenever any of the specified job state events is raised.

### Syntax

```
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name          IN VARCHAR2,
  recipients        IN VARCHAR2,
  sender            IN VARCHAR2 DEFAULT NULL,
  subject           IN VARCHAR2 DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_SUBJECT,
  body              IN VARCHAR2 DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_BODY,
  events            IN VARCHAR2 DEFAULT 'JOB_FAILED,JOB_BROKEN,JOB_SCH_LIM_REACHED,
  JOB_CHAIN_STALLED,JOB_OVER_MAX_DUR',
  filter_condition IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–14 ADD\_JOB\_EMAIL\_NOTIFICATION Procedure Parameters**

Parameter	Description
job_name	Name of the job that e-mail notifications are added for. Cannot be NULL.
recipients	Comma-separated list of e-mail addresses to send notifications to. E-mail notifications for all listed events are sent to all recipients. Cannot be NULL.
sender	e-mail address to use as the sender address (the From: address) in the e-mail header. If NULL or omitted, the e-mail address specified in the Scheduler attribute <code>email_sender</code> is used. See <i>Oracle Database Administrator's Guide</i> for more information on this Scheduler attribute.
subject	The subject to use in the e-mail header. <a href="#">Table 128–15</a> describes the variables that you can include within this parameter. The Scheduler assigns values to these variables before sending the notification. If <code>subject</code> is omitted, the default subject is used. The default subject is the following text, where text enclosed in the '%' character represents a variable:  'Oracle Scheduler Job Notification - %job_owner%.%job_name%.%job_subname% %event_type%'
body	The body of the e-mail message. <a href="#">Table 128–15</a> describes the variables that you can include within this parameter. The Scheduler assigns values to these variables before sending the notification. If <code>body</code> is omitted, the default body is used. The default body is the following text, where text enclosed in the '%' character represents a variable:  'Job: %job_owner%.%job_name%.%job_subname% Event: %event_type% Date: %event_timestamp% Log id: %log_id% Job class: %job_class_name% Run count: %run_count% Failure count: %failure_count% Retry count: %retry_count% Error code: %error_code% Error message: %error_message%'



**Table 128–14 (Cont.) ADD\_JOB\_EMAIL\_NOTIFICATION Procedure Parameters**

Parameter	Description
events	Comma-separated list of job state events to send e-mail notifications for. Cannot be NULL. A notification is sent to all recipients if any of the listed events is raised. <a href="#">Table 128–82</a> on page 128-138 lists the valid events for this parameter. If <code>events</code> is omitted, notifications are sent for the following default events:  JOB_FAILED, JOB_BROKEN, JOB_SCH_LIM_REACHED, JOB_CHAIN_STALLED, JOB_OVER_MAX_DUR
filter_condition	Used to filter events to send e-mail notifications for. If NULL, all occurrences of the specified events cause e-mail notifications to be sent. <code>filter_condition</code> must be a boolean SQL WHERE clause that may refer to the <code>:event</code> bind variable. This bind variable is automatically bound to an object of type <code>SCHEDULER\$_EVENT_INFO</code> that represents the raised event.  For example, to send an e-mail notification only when the error number in an event is 600 or 700, use the following <code>filter_condition</code> :  <code>:event.error_code=600 or :event.error_code=700</code>  See " <a href="#">SCHEDULER\$_EVENT_INFO Object Type</a> " on page 128-13.

[Table 128–15](#) lists the variables that you can use in the subject and body arguments.

**Table 128–15 Variables Used in the SUBJECT and BODY Parameters**

Variable	Comment
<code>%job_owner%</code>	Schema in which job was created
<code>%job_name%</code>	Name of the job that e-mail notifications are added for
<code>%job_subname%</code>	Present for event-based jobs with the <code>parallel_instances</code> attribute set and for chain steps
<code>%event_type%</code>	Valid values are listed in <a href="#">Table 128–82</a> on page 128-138
<code>%event_timestamp%</code>	Time at which the event occurred
<code>%log_id%</code>	Refers to the <code>LOG_ID</code> column in views <code>*_SCHEDULER_JOB_LOG</code> and <code>*_SCHEDULER_JOB_RUN_DETAILS</code>
<code>%error_code%</code>	Number of the error code.
<code>%error_message%</code>	The text of the error message
<code>%run_count%</code>	Run count for the job when the event was raised
<code>%failure_count%</code>	Failure count for the job when the event was raised
<code>%retry_count%</code>	Retry count for the job when the event was raised

## Usage Notes

You can call `ADD_JOB_EMAIL_NOTIFICATION` once for each different set of notifications that you want to configure for a particular job. For example, you may want to send notifications for the `JOB_FAILED`, `JOB_BROKEN`, `JOB_SCH_LIM_REACHED`, and `JOB_CHAIN_STALLED` events to the principle DBA and all senior DBAs, but send a notification for the `JOB_OVER_MAX_DUR` event only to the principle DBA.

This procedure succeeds only if the Scheduler attribute `email_server` is set to a valid SMTP server. See *Oracle Database Administrator's Guide* for more information.

To call this procedure, you must be the job owner or have the `CREATE ANY JOB` system privilege or have the `ALTER` object privilege on the job.

## ADD\_WINDOW\_GROUP\_MEMBER Procedure

**\*\*\* Deprecated in Oracle Database 11g Release 2 (11.2). Use ADD\_GROUP\_MEMBER instead.**

This procedure adds one or more windows to an existing window group.

### Syntax

```
DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER (
    group_name          IN VARCHAR2,
    window_list        IN VARCHAR2);
```

### Parameters

**Table 128–16 ADD\_WINDOW\_GROUP\_MEMBER Procedure Parameters**

Parameter	Description
group_name	The name of the window group
window_list	The name of the window or windows

### Usage Notes

If an already open window is added to a window group, the Scheduler will not pick up jobs that point to this window group until the next window in the window group opens.

Adding a window to a group requires the `MANAGE SCHEDULER` privilege.

Note that a window group cannot be a member of another window group.

## ALTER\_CHAIN Procedure

This procedure alters an attribute of the specified steps of a chain. This affects all future runs of the specified steps, both in the currently running chain job and in future runs of the same chain job or other chain jobs that point to the chain.

### Syntax

Alters the value of a boolean attribute of one or more steps:

```
DBMS_SCHEDULER.ALTER_CHAIN (
  chain_name          IN VARCHAR2,
  step_name           IN VARCHAR2,
  attribute            IN VARCHAR2,
  value               IN BOOLEAN);
```

Alters the value of a character attribute of one or more steps:

```
DBMS_SCHEDULER.ALTER_CHAIN (
  chain_name          IN VARCHAR2,
  step_name           IN VARCHAR2,
  attribute            IN VARCHAR2,
  char_value          IN VARCHAR2);
```

### Parameters

**Table 128–17 ALTER\_CHAIN Procedure Parameters**

Parameter	Description
chain_name	The name of the chain to alter
step_name	The name of the step or a comma-separated list of steps to alter. This cannot be NULL.

**Table 128–17 (Cont.) ALTER\_CHAIN Procedure Parameters**

Parameter	Description
attribute	<p>The attribute of the steps to change. Must be one of the following:</p> <ul style="list-style-type: none"> <li> <p>■ 'PAUSE'</p> <p>If set to TRUE for a step, after the step has run, its state changes to PAUSED (and the completed attribute remains FALSE).</p> <p>If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state is set to its completion state (SUCCEEDED, FAILED, or STOPPED) and the completed attribute is set to TRUE.</p> <p>Setting PAUSE has no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps.</p> </li> <li> <p>■ 'PAUSED_BEFORE'</p> <p>If set to TRUE for a step and if any of the rule conditions that start the step are true, then its state changes to PAUSED and the step does not run.</p> <p>If PAUSED_BEFORE is reset to FALSE for a chain step that has paused before starting (using ALTER_RUNNING_CHAIN), then the step starts running if any of the rule conditions that start the step are true.</p> <p>Setting PAUSED_BEFORE has no effect on steps that are running or have already run. This allows execution of a chain to be suspended before the execution of certain steps.</p> </li> <li> <p>■ 'SKIP'</p> <p>If set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met, the step immediately changes to state PAUSED.</p> </li> <li> <p>■ 'RESTART_ON_FAILURE'</p> <p>If set to TRUE for a step and the step fails due to an application error, then the step is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the chain step is marked FAILED.</p> <p>If set to FALSE (the default), a failed chain step is immediately marked FAILED.</p> </li> <li> <p>■ 'RESTART_ON_RECOVERY'</p> <p>If set to TRUE for a step and the step is stopped by a database shutdown, then the step is restarted when the database is recovered.</p> <p>If set to FALSE, and the step is stopped by a database shutdown, then the step is marked as stopped when the database is recovered and the chain continues.</p> </li> <li> <p>■ 'DESTINATION_NAME'</p> <p>The name of an existing database destination or external destination. You can view external destination names in the view ALL_SCHEDULER_EXTERNAL_DESTS, and database destination names in the views *_SCHEDULER_DB_DESTS. You cannot specify a destination group for this attribute. This parameter is NULL by default.</p> </li> <li> <p>■ 'CREDENTIAL_NAME'</p> <p>The credential to use when running this step. NULL by default.</p> </li> </ul>

**Table 128–17 (Cont.) ALTER\_CHAIN Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
value	The value to set for the attribute (for a boolean attribute).
char_value	The value to set for the attribute (for a character attribute).

**Usage Notes**

Altering a chain requires ALTER privileges on the chain either by being the owner of the chain, or by having the ALTER object privilege on the chain or by having the CREATE ANY JOB system privilege.

## ALTER\_RUNNING\_CHAIN Procedure

This procedure alters an attribute of the specified steps of a chain. This affects only steps of the instance of the chain for the specified running chain job.

### Syntax

```
DBMS_SCHEDULER.ALTER_RUNNING_CHAIN (
  job_name          IN VARCHAR2,
  step_name        IN VARCHAR2,
  attribute         IN VARCHAR2,
  value            IN {BOOLEAN|VARCHAR2});
```

### Parameters

**Table 128–18 ALTER\_RUNNING\_CHAIN Procedure Parameters**

Parameter	Description
job_name	The name of the job that is running the chain
step_name	The name of the step or a comma-separated list of steps to alter. If this is set to NULL and attribute is PAUSE or SKIP, then all steps of the running chain are altered.
attribute	<p>The attribute of the steps to change. Valid values are:</p> <ul style="list-style-type: none"> <li> <p>■ 'PAUSE'</p> <p>If the PAUSE attribute is set TRUE for a step, then after the step runs, its state changes to PAUSED (and the completed attribute remains false).</p> <p>If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state is set to completion (SUCCEEDED, FAILED, or STOPPED) and the completed attribute is set to TRUE. Setting PAUSE has no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps. If step_name is set to NULL, PAUSE is set to TRUE for all steps of this running chain.</p> </li> <li> <p>■ 'PAUSE_BEFORE'</p> <p>If set to TRUE for a step that has not yet run and if any of the rule conditions that start the step are true, then its state changes to PAUSED and the step does not run.</p> <p>If PAUSE_BEFORE is reset to FALSE for a chain step that has paused before starting, then the step starts running if any of the rule conditions that start the step are true.</p> <p>Setting PAUSE_BEFORE has no effect on steps that are running or have already run. This allows execution of a chain to be suspended before the execution of certain steps.</p> <p>If step_name is set to NULL, then PAUSE_BEFORE is set to the specified value for all steps of this running chain.</p> </li> </ul>

**Table 128–18 (Cont.) ALTER\_RUNNING\_CHAIN Procedure Parameters**

Parameter	Description
attribute CONTINUED	<ul style="list-style-type: none"> <li>■ 'SKIP' If the SKIP attribute is set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run.  If step_name is set to NULL, SKIP is set TRUE for all steps of this running chain. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met the step immediately changes to state PAUSED.</li> <li>■ 'RESTART_ON_FAILURE' If set to TRUE for a step and the step fails due to an application error, then the step is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the chain step is marked FAILED.  If set to FALSE (the default), a failed chain step is immediately marked FAILED.</li> <li>■ 'RESTART_ON_RECOVERY' If the RESTART_ON_RECOVERY attribute is set to TRUE for a step, then if the step is stopped by a database shutdown, it is restarted when the database is recovered.  If set to FALSE, then if the step is stopped by a database shutdown, the step is marked as stopped when the database is recovered and the chain continues.</li> <li>■ 'STATE' This changes the state of the steps. The state can only be changed if the step is not running. The state can only be changed to one of the following:  'NOT_STARTED', 'SUCCEEDED', 'FAILED error_code'  If the state is being changed to FAILED, an error code must be included (this must be a positive integer).</li> </ul>
value	The value to set for the attribute. Valid values are: TRUE, FALSE, 'NOT_STARTED', 'SUCCEEDED', or 'FAILED error_code'

## Usage Notes

Altering a running chain requires you to have alter privileges on the job that is running (either as the owner, or as a user with ALTER privileges on the job or the CREATE ANY JOB system privilege).



## CLOSE\_WINDOW Procedure

This procedure closes an open window prematurely. A closed window means that it is no longer in effect. When a window is closed, the Scheduler switches the resource plan to the one that is in effect outside the window, or in the case of overlapping windows, to another window.

### Syntax

```
DBMS_SCHEDULER.CLOSE_WINDOW (
    window_name          IN VARCHAR2);
```

### Parameters

**Table 128–19** *CLOSE\_WINDOW Procedure Parameters*

Parameter	Description
window_name	The name of the window

### Usage Notes

If you try to close a window that does not exist or is not open, an error is generated.

A job that is running does not stop when the window it is running in closes, unless the attribute `stop_on_window_close` is set to `TRUE` for the job. However, the resources allocated to the job can change if the resource plan changes.

When a running job has a group of type `WINDOW` as its schedule, the job is not stopped when its window is closed if another window in the same window group becomes active. This is the case even if the job has the attribute `stop_on_window_close` set to `TRUE`.

Closing a window requires the `MANAGE SCHEDULER` privilege.

## COPY\_JOB Procedure

This procedure copies all attributes of an existing job to a new job. The new job is created disabled, while the state of the existing job is unaltered.

### Syntax

```
DBMS_SCHEDULER.COPY_JOB (  
    old_job          IN VARCHAR2,  
    new_job          IN VARCHAR2);
```

### Parameters

**Table 128–20** *COPY\_JOB Procedure Parameters*

Parameter	Description
old_job	The name of the existing job
new_job	The name of the new job

### Usage Notes

To copy a job, you must have privileges to create a job in the schema of the new job (the `CREATE JOB` system privilege if it is in your own schema, otherwise, the `CREATE ANY JOB` system privilege). If the old job is not in your own schema, then you must also have `ALTER` privileges on the old job or the `CREATE ANY JOB` system privilege.

## CREATE\_CHAIN Procedure

This procedure creates a new chain. The chain name can be optionally qualified with a schema name (for example, `myschema.mychain`).

A chain is always created as disabled and must be enabled with the [ENABLE Procedure](#) before it can be used.

### Syntax

```
DBMS_SCHEDULER.CREATE_CHAIN (
  chain_name          IN VARCHAR2,
  rule_set_name       IN VARCHAR2 DEFAULT NULL,
  evaluation_interval IN INTERVAL DAY TO SECOND DEFAULT NULL,
  comments            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–21 CREATE\_CHAIN Procedure Parameters**

Parameter	Description
<code>chain_name</code>	The name to assign to the new chain, which can optionally be qualified with a schema. This must be unique in the SQL namespace, therefore, there cannot already be a table or other object with this name and schema.
<code>rule_set_name</code>	In the normal case, no rule set should be passed in. The Scheduler automatically creates a rule set and associated empty evaluation context. You then use <code>DEFINE_CHAIN_RULE</code> to add rules and <code>DROP_CHAIN_RULE</code> to remove them.  Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created is of the form <code>SCHED_RULESET\${N}</code> and the evaluation context created is of the form <code>SCHED_EVCTX\${N}</code> .  See <i>Oracle Streams Concepts and Administration</i> for information on rules and rule sets.
<code>evaluation_interval</code>	If this is <code>NULL</code> , reevaluation of the rules of a running chain are performed only when the job starts and when a step completes. A non- <code>NULL</code> value causes rule evaluations to also occur periodically at the specified interval. Because evaluation may be CPU-intensive, this should be conservatively set to the highest possible value or left at <code>NULL</code> if possible. <code>evaluation_interval</code> cannot be less than a minute or greater than a day.
<code>comments</code>	An optional comment describing the purpose of the chain

### Usage Notes

To create a chain in your own schema, you must have the `CREATE JOB` system privilege. To create a chain in a different schema you must have the `CREATE ANY JOB` system privilege. If you do not provide a `rule_set_name`, a rule set and evaluation context is created in the schema that the chain is being created in, so you must have the privileges required to create these objects. See the `DBMS_RULE_ADM.CREATE_`

RULE\_SET and DBMS\_RULE\_ADM.CREATE\_EVALUATION\_CONTEXT procedures for more information.

## CREATE\_CREDENTIAL Procedure

This procedure creates a stored username/password pair. Credentials are assigned to jobs so that they can authenticate with a local or remote host operating system or a remote Oracle database.

### Syntax

```
DBMS_SCHEDULER.CREATE_CREDENTIAL (
  credential_name      IN VARCHAR2,
  username             IN VARCHAR2,
  password             IN VARCHAR2,
  database_role        IN VARCHAR2 DEFAULT NULL,
  windows_domain       IN VARCHAR2 DEFAULT NULL,
  comments              IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–22 CREATE\_CREDENTIAL Procedure Parameters**

Parameter	Description
credential_name	The name to assign to the credential. It can optionally be prefixed with a schema name. It cannot be set to NULL. It is converted to uppercase unless enclosed in double-quotes.
username	The user name for logging into to the host operating system or remote Oracle database. This cannot be set to NULL and is case-sensitive. It cannot contain double quotes or spaces. Maximum length is 64.
password	The password for the user name. This cannot be set to NULL and is case sensitive. The password is stored obfuscated and is not displayed in the Scheduler dictionary views. Maximum length is 128.
database_role	The value of the database_role attribute is used as the system privilege for logging into a remote database to run a remote database job. Valid values are: SYSDBA and SYSOPER
windows_domain	For a Windows remote executable target, this is the domain that the specified user belongs to. The domain is converted to uppercase automatically. Maximum length is 64.
comments	A text string that can be used to describe the credential. Scheduler does not use this parameter. Maximum length is 240.

### Usage Notes

Credentials reside in a particular schema and can be created by any user with the CREATE JOB system privilege. To create a credential in a schema other than your own, you must have the CREATE ANY JOB privilege.

## CREATE\_DATABASE\_DESTINATION Procedure

This procedure creates a database destination. A database destination represents an Oracle database on which remote database jobs run.

The host that the remote database resides on must have a running Scheduler agent that is registered with the database that this procedure is called from.

### Syntax

```
DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION (
    destination_name    IN VARCHAR2,
    agent               IN VARCHAR2,
    tns_name            IN VARCHAR2,
    comments            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–23 CREATE\_DATABASE\_DESTINATION Procedure Parameters**

Parameter	Description
<code>destination_name</code>	The name to assign to the database destination. It can optionally be prefixed with a schema name. Cannot be <code>NULL</code> . It is converted to uppercase unless enclosed in double-quotes.
<code>agent</code>	The external destination name of the Scheduler agent to connect. Equivalent to an agent name.  The external destination must already exist. The external destination representing an agent is created automatically on a database instance when the agent registers with that instance.  An agent's name is specified in its agent configuration file. If it is not specified, it defaults to the first part (before the first period) of the name of the host it resides on.
<code>tns_name</code>	An Oracle Net connect identifier that is resolved to the Oracle database instance being connected to. The exact syntax depends on the Oracle Net configuration.  The connect identifier can be a complete Oracle Net connect descriptor (network address and database service name) or a <i>net service name</i> , which is an alias for a connect descriptor. The alias must be resolved in the <code>tnsnames.ora</code> file on the local computer. The maximum size for <code>tns_name</code> is 2000 characters.  If <code>tns_name</code> is <code>NULL</code> , the agent connects to the default Oracle database on its host. You specify the default database by assigning values to the <code>ORACLE_HOME</code> and <code>ORACLE_SID</code> parameters in the agent configuration file, <code>schagent.conf</code> , located in the agent home directory.  See <i>Oracle Database Net Services Administrator's Guide</i> for more information on connect identifiers.
<code>comments</code>	A text string that describes the database destination. Scheduler does not use this argument.

### Usage Notes

Database destinations reside in a particular schema and can be created by any user with the `CREATE JOB` system privilege. To create a database destination in a schema other than your own, you must have the `CREATE ANY JOB` privilege.

## CREATE\_EVENT\_SCHEDULE Procedure

This procedure creates an event schedule, which is used to start a job when a particular event is raised.

### Syntax

```
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
  schedule_name          IN VARCHAR2,
  start_date             IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  event_condition        IN VARCHAR2 DEFAULT NULL,
  queue_spec             IN VARCHAR2,
  end_date               IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  comments               IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–24 CREATE\_EVENT\_SCHEDULE Parameters**

Parameter	Description
schedule_name	The name to assign to the schedule. The name must be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
start_date	This attribute specifies the date and time that this schedule becomes valid. Occurrences of the event before this date and time are ignored in the context of this schedule.
event_condition	This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression, provided that the message payload is an object type, and that you prefix object attributes in the expression with <code>tab.user_data</code> . For more information on rules, see the <code>DBMS_AQADM.ADD_SUBSCRIBER</code> procedure.
queue_spec	This argument specifies either a file watcher name or the queue into which events that start this particular job are enqueued (the <b>source queue</b> ). If the source queue is a secure queue, the <code>queue_spec</code> argument is a string containing a pair of values of the form <code>queue_name, agent name</code> . For non-secure queues, only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue.
end_date	The date and time after which jobs do not run and windows do not open.  An event schedule that has no <code>end_date</code> is valid forever.  <code>end_date</code> must be after the <code>start_date</code> . If it is not, then an error is generated when the schedule is created.
comments	This attribute specifies an optional comment about the schedule. By default, this attribute is <code>NULL</code> .

### Usage Notes

You must have the `CREATE JOB` privilege to create a schedule in your own schema or the `CREATE ANY JOB` privilege to create a schedule in someone else's schema by specifying `schema.schedule_name`. Once a schedule has been created, it can be

used by other users. The schedule is created with access to PUBLIC. Therefore, there is no need to explicitly grant access to the schedule.

**See Also:** ["CREATE\\_FILE\\_WATCHER Procedure"](#) on page 128-53



## CREATE\_FILE\_WATCHER Procedure

This procedure creates a file watcher, which is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job. After you create a file watcher, you reference it in an event-based job or event schedule.

### Syntax

```
DBMS_SCHEDULER.CREATE_FILE_WATCHER (
  file_watcher_name      IN VARCHAR2,
  directory_path         IN VARCHAR2,
  file_name              IN VARCHAR2,
  credential_name        IN VARCHAR2,
  destination           IN VARCHAR2 DEFAULT NULL,
  min_file_size          IN PLS_INTEGER DEFAULT 0,
  steady_state_duration IN INTERVAL DAY TO SECOND DEFAULT NULL,
  comments               IN VARCHAR2 DEFAULT NULL,
  enabled                IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 128–25 CREATE\_FILE\_WATCHER Parameters**

Parameter	Description
file_watcher_name	The name to assign to the file watcher. The name must be unique in the SQL namespace. For example, a file watcher cannot have the same name as a table in a schema. This can optionally be prefixed with a schema name. Cannot be NULL.
directory_path	Directory in which the file is expected to arrive. The single wildcard '?' at the beginning of the path denotes the Oracle home path. For example, '?/rdbms/log' denotes the rdbms/log subdirectory of the Oracle home directory.
file_name	Name of the file to look for. Two wildcards are permitted anywhere in the file name: '?' denotes any single character, and '*' denotes zero or more characters. This attribute cannot be NULL.
credential_name	Name of a valid Scheduler credential object.  The file watcher uses the credential to authenticate itself with the host operating system to access the watched-for file. The file watcher owner must have EXECUTE privileges on the credential. Cannot be NULL.
destination	Name of an external destination. You create an external destination by registering a remote Scheduler agent with the database. See the view ALL_SCHEDULER_EXTERNAL_DESTS for valid external destination names. If this parameter is NULL, the file watcher is created on the local host.
min_file_size	Minimum size in bytes that the file must be before the file watcher considers the file found. Default is 0.
steady_state_duration	Minimum time interval that the file must remain unchanged before the file watcher considers the file found. Cannot exceed one hour. If NULL, an internal value is used.
comments	Optional comment.
enabled	If TRUE (the default), the file watcher is enabled.

## Usage Notes

You must have the `CREATE JOB` system privilege to create a file watcher in your own schema. You require the `CREATE ANY JOB` system privilege to create a file watcher in a schema different from your own (except the `SYS` schema, which is disallowed).

## CREATE\_GROUP Procedure

This procedure creates a group. Groups contain members, which you can specify when you create the group or at a later time. There are three types of groups: window groups, database destination groups, and external destination groups.

You can use a group name in other DBMS\_SCHEDULER package procedures to specify a list of objects. For example, to specify multiple destinations for a remote database job, you provide a group name for the `DESTINATION_NAME` parameter of the job.

### Syntax

```
DBMS_SCHEDULER.CREATE_GROUP (
  group_name      IN VARCHAR2,
  group_type      IN VARCHAR2,
  member          IN VARCHAR2 DEFAULT NULL,
  comments        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–26 CREATE\_GROUP Procedure Parameters**

Parameter	Description
<code>group_name</code>	The name to assign to the group. It can optionally be prefixed with a schema name. It cannot be <code>NULL</code> . It is converted to uppercase unless enclosed in double-quotes.
<code>group_type</code>	<p>The type of members in the group. All members must be of the same type. Possible types are:</p> <ul style="list-style-type: none"> <li>▪ <code>'DB_DEST'</code> Database destination: Members are database destinations, for running remote database jobs.</li> <li>▪ <code>'EXTERNAL_DEST'</code> External destination: Members are external destinations, for running remote external jobs.</li> <li>▪ <code>'WINDOW'</code> Members are Scheduler windows. You must have the <code>MANAGE SCHEDULER</code> privilege to create a group of this type.</li> </ul> <p>Members in database destination and external destination groups have the following format:</p> <pre>[[<i>schema.</i>]<i>credential</i>@][<i>schema.</i>]<i>destination</i></pre> <p>where:</p> <ul style="list-style-type: none"> <li>▪ <i>credential</i> is the name of an existing credential.</li> <li>▪ <i>destination</i> is the name of an existing database destination or external destination.</li> </ul> <p>The credential portion of a destination member is optional. If omitted, the job using this destination member uses its default credential.</p> <p>Members in window groups are window names. Because all Scheduler windows reside in the <code>SYS</code> schema, you do not specify a schema name for windows.</p>

**Table 128–26 (Cont.) CREATE\_GROUP Procedure Parameters**

Parameter	Description
<code>member</code>	<p>Optional comma-separated list of group members. The default is <code>NULL</code>. If <code>NULL</code>, use the <code>ADD_GROUP_MEMBER</code> procedure to add members. You can also use <code>ADD_GROUP_MEMBER</code> to add additional members at a later time.</p> <p>The keyword <code>LOCAL</code> can be used as a member in database destination groups and external destination groups.</p> <ul style="list-style-type: none"> <li>▪ In database destination groups, <code>LOCAL</code> represents the source database on which the job is created. It cannot be preceded with a credential.</li> <li>▪ In external destination groups, <code>LOCAL</code> represents the host on which the source database resides. It can be optionally preceded with a credential name. If no credential is provided, jobs that use this group as their destination must have a default credential.</li> </ul>
<code>comments</code>	A text string that describes the group. Scheduler does not use this argument.

## Usage Notes

Groups reside in a particular schema and can be created by any user with the `CREATE JOB` system privilege. To create a group in a schema other than your own, you must have the `CREATE ANY JOB` privilege. The group name must be unique among all Scheduler objects.

You can grant the `SELECT` privilege on a group so that other users can reference the group when creating jobs or schedules. To enable other users to modify a group, you can grant the `ALTER` privilege on the group.

Each group member must be unique within the group. For destination groups, the credential/destination name pairs must be unique within the group. An error is generated if any of the group members do not exist. For destination groups, both the credential and destination portions of a member must exist.

Another group of the same type can be a group member. The Scheduler immediately expands the included group name into its list of members.

Groups are created enabled, but you can disable them.

This procedure supersedes the procedure `CREATE_WINDOW_GROUP`, which is deprecated in Oracle Database 11g Release 2 (11.2).

## Example

The following PL/SQL block creates a group named `production_dest1`, whose members are database destinations for a collection of production databases.

```
BEGIN
  DBMS_SCHEDULER.CREATE_GROUP (
    GROUP_NAME => 'production_dest1',
    GROUP_TYPE => 'DB_DEST',
    MEMBER     => 'LOCAL, oracle_cred@prodbost1, prodbost2',
    COMMENTS  => 'All sector1 production machines');
END;
```

## CREATE\_JOB Procedure

This procedure creates a single job (regular or lightweight). If you create the job as enabled by setting the `enabled` attribute to `TRUE`, the Scheduler automatically runs the job according to its schedule. If you create the job disabled, the job does not run until you enable it with the [SET\\_ATTRIBUTE Procedure](#).

The procedure is overloaded. The different functionality of each form of syntax is presented along with the syntax declaration.

### Syntax

Creates a job in a single call without using an existing program or schedule:

```
DBMS_SCHEDULER.CREATE_JOB (
  job_name          IN VARCHAR2,
  job_type          IN VARCHAR2,
  job_action        IN VARCHAR2,
  number_of_arguments IN PLS_INTEGER          DEFAULT 0,
  start_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  repeat_interval   IN VARCHAR2              DEFAULT NULL,
  end_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled           IN BOOLEAN                DEFAULT FALSE,
  auto_drop         IN BOOLEAN                DEFAULT TRUE,
  comments          IN VARCHAR2              DEFAULT NULL,
  credential_name   IN VARCHAR2              DEFAULT NULL,
  destination_name  IN VARCHAR2              DEFAULT NULL);
```

Creates a job using a named schedule object and a named program object:

```
DBMS_SCHEDULER.CREATE_JOB (
  job_name          IN VARCHAR2,
  program_name      IN VARCHAR2,
  schedule_name     IN VARCHAR2,
  job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled           IN BOOLEAN                DEFAULT FALSE,
  auto_drop         IN BOOLEAN                DEFAULT TRUE,
  comments          IN VARCHAR2              DEFAULT NULL,
  job_style         IN VARCHAR2              DEFAULT 'REGULAR',
  credential_name   IN VARCHAR2              DEFAULT NULL,
  destination_name  IN VARCHAR2              DEFAULT NULL);
```

Creates a job using a named program object and an inlined schedule:

```
DBMS_SCHEDULER.CREATE_JOB (
  job_name          IN VARCHAR2,
  program_name      IN VARCHAR2,
  start_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  repeat_interval   IN VARCHAR2              DEFAULT NULL,
  end_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled           IN BOOLEAN                DEFAULT FALSE,
  auto_drop         IN BOOLEAN                DEFAULT TRUE,
  comments          IN VARCHAR2              DEFAULT NULL,
  job_style         IN VARCHAR2              DEFAULT 'REGULAR',
  credential_name   IN VARCHAR2              DEFAULT NULL,
  destination_name  IN VARCHAR2              DEFAULT NULL);
```

Creates a job using a named schedule object and an inlined program:

```

DBMS_SCHEDULER.CREATE_JOB (
  job_name          IN VARCHAR2,
  schedule_name     IN VARCHAR2,
  job_type          IN VARCHAR2,
  job_action        IN VARCHAR2,
  number_of_arguments IN PLS_INTEGER          DEFAULT 0,
  job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled          IN BOOLEAN               DEFAULT FALSE,
  auto_drop        IN BOOLEAN               DEFAULT TRUE,
  comments         IN VARCHAR2              DEFAULT NULL,
  credential_name  IN VARCHAR2              DEFAULT NULL,
  destination_name IN VARCHAR2              DEFAULT NULL);

```

Creates a job using an inlined program and an event:

```

DBMS_SCHEDULER.CREATE_JOB (
  job_name          IN VARCHAR2,
  job_type          IN VARCHAR2,
  job_action        IN VARCHAR2,
  number_of_arguments IN PLS_INTEGER          DEFAULT 0,
  start_date       IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  event_condition  IN VARCHAR2              DEFAULT NULL,
  queue_spec       IN VARCHAR2,
  end_date         IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled          IN BOOLEAN               DEFAULT FALSE,
  auto_drop        IN BOOLEAN               DEFAULT TRUE,
  comments         IN VARCHAR2              DEFAULT NULL,
  credential_name  IN VARCHAR2              DEFAULT NULL,
  destination_name IN VARCHAR2              DEFAULT NULL);

```

Creates a job using a named program object and an event:

```

DBMS_SCHEDULER.CREATE_JOB (
  job_name          IN VARCHAR2,
  program_name      IN VARCHAR2,
  start_date       IN TIMESTAMP WITH TIME ZONE,
  event_condition  IN VARCHAR2,
  queue_spec       IN VARCHAR2,
  end_date         IN TIMESTAMP WITH TIME ZONE,
  job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
  enabled          IN BOOLEAN               DEFAULT FALSE,
  auto_drop        IN BOOLEAN               DEFAULT TRUE,
  comments         IN VARCHAR2              DEFAULT NULL,
  job_style        IN VARCHAR2              DEFAULT 'REGULAR',
  credential_name  IN VARCHAR2              DEFAULT NULL,
  destination_name IN VARCHAR2              DEFAULT NULL);

```

## Parameters

**Table 128–27 CREATE\_JOB Procedure Parameters**

Parameter	Description
job_name	<p>The name to assign to the job. The name must be unique in the SQL namespace. For example, a job cannot have the same name as a table in a schema. If the job being created will reside in another schema, it must be qualified with the schema name.</p> <p>If <code>job_name</code> is not specified, an error is generated. If you want to have a name generated by the Scheduler, you can use the <code>GENERATE_JOB_NAME</code> procedure to generate a name and then use the output in the <code>CREATE_JOB</code> procedure. The <code>GENERATE_JOB_NAME</code> procedure generates a number from a sequence, which is the job name. You can prefix the number with a string. The job name will then be the string with the number from the sequence appended to it. See "<a href="#">GENERATE_JOB_NAME Function</a>" on page 128-109 for more information.</p>
job_type	<p>This attribute specifies the type of job that you are creating. If it is not specified, an error is generated. The supported values are:</p> <ul style="list-style-type: none"> <li> <p>■ 'PLSQL_BLOCK'</p> <p>This specifies that the job is an anonymous PL/SQL block. Job or program arguments are not supported when the job or program type is <code>PLSQL_BLOCK</code>. In this case, the number of arguments must be 0.</p> </li> <li> <p>■ 'STORED_PROCEDURE'</p> <p>This specifies that the job is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported.</p> </li> <li> <p>■ 'EXECUTABLE'</p> <p>This specifies that the job is external to the database. External jobs are anything that can be executed from the command line of the operating system. Anydata arguments are not supported with a job or program type of <code>EXECUTABLE</code>. The job owner must have the <code>CREATE EXTERNAL JOB</code> system privilege before the job can be enabled or run.</p> </li> <li> <p>■ 'CHAIN'</p> <p>This specifies that the job is a chain. Arguments are not supported for a chain, so <code>number_of_arguments</code> must be 0.</p> </li> </ul>

**Table 128–27 (Cont.) CREATE\_JOB Procedure Parameters**

Parameter	Description
<code>job_action</code>	<p>This attribute specifies the action of the job. If <code>job_action</code> is not specified for an inline program, then an error is generated when creating the job.</p> <p>The following actions are possible:</p> <ul style="list-style-type: none"> <li>■ For a PL/SQL block: <p>The action is to execute PL/SQL code. These blocks must end with a semicolon. For example, <code>my_proc()</code>; or <code>BEGIN my_proc(); END;</code> or <code>DECLARE arg pls_integer:= 10; BEGIN my_proc2(arg); END;</code>.</p> <p>Note that the Scheduler wraps <code>job_action</code> in its own block and passes the following to PL/SQL for execution: <code>DECLARE ... BEGIN job_action END;</code> This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except <code>event_message</code> in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value.</p> <p>See <a href="#">Table 128–39</a> on page 128-81 for details on available metadata attributes.</p> </li> <li>■ For a stored procedure: <p>The action is the name of the stored procedure. You have to specify the schema if the procedure resides in another schema than the job. If case sensitivity is needed, enclose the schema name and the store procedure name in double quotes. For example, <code>job_action=&gt;' "Schema" ." Procedure"'</code>.</p> <p>PL/SQL procedures with <code>INOUT</code> or <code>OUT</code> arguments are not supported as <code>job_action</code> when the job or program type is <code>STORED_ PROCEDURE</code>.</p> </li> <li>■ For an executable: <p>The action is the name of the external executable, including the full path name, but excluding any command-line arguments. If the action starts with a single question mark ('?'), the question mark is replaced by the path to the Oracle home directory for a local job or to the Scheduler agent home for a remote job. If the action contains an at-sign ('@') and the job is local, the at-sign is replaced with the SID of the current Oracle instance.</p> </li> <li>■ For a chain: <p>The action is the name of a Scheduler chain object. You must specify the schema of the chain if it resides in a different schema than the job.</p> </li> </ul>
<code>number_of_arguments</code>	This attribute specifies the number of arguments that the job expects. The range is 0-255, with the default being 0.
<code>program_name</code>	The name of the program associated with this job. If the program is of type <code>EXECUTABLE</code> , the job owner must have the <code>CREATE EXTERNAL JOB</code> system privilege before the job can be enabled or run.
<code>start_date</code>	<p>This attribute specifies the first date and time on which this job is scheduled to start. If <code>start_date</code> and <code>repeat_interval</code> are left null, then the job is scheduled to run as soon as the job is enabled.</p> <p>For repeating jobs that use a calendaring expression to specify the repeat interval, <code>start_date</code> is used as a reference date. The first time the job runs is the first match of the calendaring expression that is on or after the current date and time.</p> <p>The Scheduler cannot guarantee that a job executes on an exact time because the system may be overloaded and thus resources unavailable.</p>



**Table 128–27 (Cont.) CREATE\_JOB Procedure Parameters**

Parameter	Description
event_condition	This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression provided that the message payload is an object type, and that you prefix object attributes in the expression with <code>tab.user_data</code> . For more information on rules, see the <code>DBMS_AQADM.ADD_SUBSCRIBER</code> procedure.
queue_spec	This argument specifies either of the following: <ul style="list-style-type: none"> <li>■ The source queue where events that start this particular job are enqueued. If it is secure, then the <code>queue_spec</code> argument is a pair of values of the form <code>queue_name, agent_name</code>. If it is not secure, then only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue.</li> <li>■ A file watcher name. For more information on this option, see <i>Oracle Database Administrator's Guide</i>.</li> </ul>
repeat_interval	This attribute specifies how often the job repeats. You can specify the repeat interval by using calendaring or PL/SQL expressions.  The expression specified is evaluated to determine the next time the job should run. If <code>repeat_interval</code> is not specified, the job runs only once at the specified start date. See "Calendaring Syntax" on page 128-20 for further information.
schedule_name	The name of the schedule, window, or window group associated with this job.
job_class	The class this job is associated with.
end_date	This attribute specifies the date and time after which the job expires and is no longer run. After the <code>end_date</code> , if <code>auto_drop</code> is <code>TRUE</code> , the job is dropped. If <code>auto_drop</code> is <code>FALSE</code> , the job is disabled and the <code>STATE</code> of the job is set to <code>COMPLETED</code> .  If no value for <code>end_date</code> is specified, the job repeats forever unless <code>max_runs</code> or <code>max_failures</code> is set, in which case the job stops when either value is reached.  The value for <code>end_date</code> must be after the value for <code>start_date</code> . If it is not, an error is generated when the job is enabled.
comments	This attribute specifies a comment about the job. By default, this attribute is <code>NULL</code> .
job_style	Style of the job being created. This argument can have one of the following values: <ul style="list-style-type: none"> <li>■ 'REGULAR' creates a regular job. This is the default.</li> <li>■ 'LIGHTWEIGHT' creates a lightweight job. This value is permitted only when the job references a program object. Use lightweight jobs when you have many short-duration jobs that run frequently. Under certain circumstances, using lightweight jobs can deliver a small performance gain.</li> </ul>
credential_name	The default credential to use with the job. Applicable only to remote database jobs, remote external jobs, local external jobs, and event-based jobs that process file arrival events. The credential must exist.  For local database jobs, it must be <code>NULL</code> .  For local external jobs only, if this attribute is <code>NULL</code> (the default), then a preferred (default) credential is selected. See <i>Oracle Database Administrator's Guide</i> for information about preferred credentials for local external jobs.  See also: "CREATE_CREDENTIAL Procedure" on page 128-49

**Table 128–27 (Cont.) CREATE\_JOB Procedure Parameters**

Parameter	Description
<code>destination_name</code>	<p>The database destination or external destination for the job. Use for remote database jobs and remote external jobs only. Must be <code>NULL</code> for jobs running on the local database or for local external jobs (executables).</p> <p>This attribute can be a single destination name or the name of a group of type <code>'EXTERNAL_DEST'</code> or <code>'DB_DEST'</code>. The single destination or group must already exist.</p> <p>The following applies to this attribute:</p> <ul style="list-style-type: none"> <li>■ If it is a database destination, it must have been created by the <a href="#">CREATE_DATABASE_DESTINATION Procedure</a>.</li> <li>■ If it is an external destination, it must have been implicitly created by registering a remote Scheduler agent with the local database.</li> <li>■ If it is a group, each member of the group must exist, and the job must run on all destinations named in the group. See "<a href="#">CREATE_GROUP Procedure</a>" on page 128-55.</li> </ul> <p><code>destination_name</code> cannot reference a destination group when:</p> <ul style="list-style-type: none"> <li>■ The job type is <code>'CHAIN'</code></li> <li>■ The job style is <code>'LIGHTWEIGHT'</code></li> </ul> <p>If the <code>credential_name</code> argument of <code>CREATE_JOB</code> is <code>NULL</code>, each destination must be preceded by a credential, in the following format:</p> <p><i>credential.destination</i></p> <p>The credential must already exist. If the <code>credential_name</code> argument is provided, then it serves as the default credential for every destination that is not preceded by a credential.</p> <p>You can query the views <code>*_SCHEDULER_DB_DESTS</code> and <code>ALL_SCHEDULER_EXTERNAL_DESTS</code> for existing destinations and <code>*_SCHEDULER_GROUP_MEMBERS</code> for existing groups and their members.</p> <p>*** <code>destination_name</code> supersedes the <code>destination</code> job attribute, which is deprecated in Oracle Database 11g Release 2 (11.2).</p>
<code>enabled</code>	<p>This attribute specifies whether the job is created enabled or not. The possible settings are <code>TRUE</code> or <code>FALSE</code>. By default, this attribute is set to <code>FALSE</code> and, therefore, the job is created as disabled. A disabled job means that the metadata about the job has been captured, and the job exists as a database object. However, the Scheduler ignores the job and the job coordinator does not pick it for processing. In order for the job coordinator to process the job, the job must be enabled. You can enable a job by setting this argument to <code>TRUE</code> or by using the <code>ENABLE</code> procedure.</p>
<code>auto_drop</code>	<p>This flag, if <code>TRUE</code>, causes a job to be automatically dropped after it has completed or has been automatically disabled. A job is considered completed if:</p> <ul style="list-style-type: none"> <li>■ Its end date (or the end date of the job schedule) has passed.</li> <li>■ It has run <code>max_runs</code> number of times. <code>max_runs</code> must be set with <code>SET_ATTRIBUTE</code>.</li> <li>■ It is not a repeating job and has run once.</li> </ul> <p>A job is disabled when it has failed <code>max_failures</code> times. <code>max_failures</code> is also set with <code>SET_ATTRIBUTE</code>.</p> <p>If this flag is set to <code>FALSE</code>, the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the <code>DROP_JOB</code> procedure.</p> <p>By default, jobs are created with <code>auto_drop</code> set to <code>TRUE</code>.</p>

## Usage Notes

Jobs are created as disabled by default. You must explicitly enable them so that they will become active and scheduled. Before enabling a job, ensure that all program arguments, if any, are defined, either by defining default values in the program object or by supplying values with the job.

The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of processes that can be created for the execution of jobs. Beginning with Oracle Database 11g Release 2 (11.2), `JOB_QUEUE_PROCESSES` applies to `DBMS_SCHEDULER` jobs. Setting this parameter to 0 disables `DBMS_SCHEDULER` jobs.

To create a job in your own schema, you need to have the `CREATE JOB` privilege. A user with the `CREATE ANY JOB` privilege can create a job in any schema. If the job being created will reside in another schema, the job name must be qualified with the schema name. For a job of type `EXECUTABLE` (or for a job that points to a program of type `EXECUTABLE`), the job owner must have the `CREATE EXTERNAL JOB` system privilege before the job can be enabled or run.

Associating a job with a particular class or program requires `EXECUTE` privileges for that class or program.

Not all possible job attributes can be set with `CREATE_JOB`. Some must be set after the job is created. For example, job arguments must be set with the [SET\\_JOB\\_ARGUMENT\\_VALUE Procedure](#) or the [SET\\_JOB\\_ANYDATA\\_VALUE Procedure](#). Other job attributes, such as `job_priority` and `max_runs`, are set with the [SET\\_ATTRIBUTE Procedure](#).

To create multiple jobs efficiently, use the `CREATE_JOBS` procedure.

---

---

**Note:** The Scheduler runs event-based jobs for each occurrence of an event that matches the event condition of the job. However, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job.

---

---

## CREATE\_JOB\_CLASS Procedure

This procedure creates a job class. Job classes are created in the SYS schema.

### Syntax

```
DBMS_SCHEDULER.CREATE_JOB_CLASS (
  job_class_name          IN VARCHAR2,
  resource_consumer_group IN VARCHAR2 DEFAULT NULL,
  service                 IN VARCHAR2 DEFAULT NULL,
  logging_level           IN PLS_INTEGER
                          DEFAULT DBMS_SCHEDULER.LOGGING_RUNS,
  log_history             IN PLS_INTEGER DEFAULT NULL,
  comments                IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–28 CREATE\_JOB\_CLASS Procedure Parameters**

Parameter	Description
job_class_name	<p>The name to assign to the job class. Job classes can only be created in the SYS schema.</p> <p>This attribute specifies the name of the job class and uniquely identifies the job class. The name must be unique in the SQL namespace. For example, a job class cannot have the same name as a table in a schema.</p>
resource_consumer_group	<p>This attribute specifies the resource consumer group that this class is associated with. A resource consumer group is a set of synchronous or asynchronous sessions that are grouped together based on their processing needs. A job class has a many-to-one relationship with a resource consumer group. The resource consumer group that the job class associates with determines the resources that are allocated to the job class.</p> <p>If a resource consumer group is dropped, job classes associated with it are then associated with the default resource consumer group.</p> <p>If no resource consumer group is specified, job classes are associated with the default resource consumer group.</p> <p>If the specified resource consumer group does not exist when creating the job class, an error occurs.</p>
service	<p>This attribute specifies the database service that the jobs in this class have affinity to. In an Oracle RAC environment, this means that the jobs in this class only run on those database instances that are assigned to the specific service.</p> <p>Note that a service can be mapped to a resource consumer group, so you can also control resources allocated to jobs by specifying a service. See <code>DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING</code> for details. If both the <code>resource_consumer_group</code> and <code>service</code> attributes are specified, and if the service is mapped to a resource consumer group, the <code>resource_consumer_group</code> attribute takes precedence.</p> <p>If no service is specified, the job class belongs to the default service, which means it has no service affinity and any one of the database instances within the cluster might run the job. If the service that a job class belongs to is dropped, the job class will then belong to the default service.</p> <p>If the specified service does not exist when creating the job class, then an error occurs.</p>

**Table 128–28 (Cont.) CREATE\_JOB\_CLASS Procedure Parameters**

Parameter	Description
logging_level	<p>This attribute specifies how much information is logged. The possible options are:</p> <ul style="list-style-type: none"> <li>■ <code>DBMS_SCHEDULER.LOGGING_OFF</code> No logging is performed for any jobs in this class.</li> <li>■ <code>DBMS_SCHEDULER.LOGGING_RUNS</code> The Scheduler writes detailed information to the job log for all runs of each job in this class. This is the default.</li> <li>■ <code>DBMS_SCHEDULER.LOGGING_FAILED_RUNS</code> The Scheduler logs only jobs that failed in this class.</li> <li>■ <code>DBMS_SCHEDULER.LOGGING_FULL</code> In addition to recording every run of a job, the Scheduler records all operations performed on all jobs in this class. Every time a job is created, enabled, disabled, altered (with <code>SET_ATTRIBUTE</code>), stopped, and so, an entry is recorded in the log.</li> </ul>
log_history	<p>This attribute controls the number of days that job log entries for jobs in this class are retained. It helps prevent the job log from growing indiscriminately.</p> <p>The range of valid values is 0 through 1000000. If set to 0, no history is kept. If <code>NULL</code> (the default), retention days are set by the <code>log_history</code> Scheduler attribute (set with <code>SET_SCHEDULER_ATTRIBUTE</code>).</p>
comments	<p>This attribute is for an optional comment about the job class. By default, this attribute is <code>NULL</code>.</p>

## Usage Notes

For users to create jobs that belong to a job class, the job owner must have `EXECUTE` privileges on the job class. Therefore, after the job class has been created, `EXECUTE` privileges must be granted on the job class so that users create jobs belonging to that class. You can also grant the `EXECUTE` privilege to a role.

Creating a job class requires the `MANAGE SCHEDULER` system privilege.

## CREATE\_JOBS Procedure

This procedure creates multiple jobs (regular or lightweight) and sets the values of their arguments in a single call.

### Syntax

```
DBMS_SCHEDULER.CREATE_JOBS (
  jobdef_array      IN SYS.JOB_DEFINITION_ARRAY,
  commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

### Parameters

**Table 128–29 CREATE\_JOBS Procedure Parameters**

Parameter	Description
jobdef_array	The array of job definitions. See "Data Structures" on page 128-2 for a description of the JOB_DEFINITION_ARRAY and JOB_DEFINITION data types.
commit_semantics	The commit semantics. The following types are supported: <ul style="list-style-type: none"> <li>▪ STOP_ON_FIRST_ERROR returns on the first error. Previous successfully created jobs are committed to disk. This is the default.</li> <li>▪ TRANSACTIONAL returns on the first error and everything that happened before that error is rolled back.</li> <li>▪ ABSORB_ERRORS tries to absorb any errors and attempts to create the rest of the jobs on the list. It commits all successfully created jobs. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details.</li> </ul>

### Usage Notes

This procedure creates many jobs in the context of a single transaction. To realize the desired performance gains, the jobs being created must be grouped in batches of sufficient size. Calling CREATE\_JOBS with a small array size may not be much faster than calling CREATE\_JOB once for each job.

You cannot use this procedure to create multiple-destination jobs. That is, the destination attribute of the job\_definition object cannot reference a destination group.

### Examples

See *Oracle Database Administrator's Guide*.

## CREATE\_PROGRAM Procedure

This procedure creates a program.

### Syntax

```
DBMS_SCHEDULER.CREATE_PROGRAM (
    program_name          IN VARCHAR2,
    program_type          IN VARCHAR2,
    program_action        IN VARCHAR2,
    number_of_arguments  IN PLS_INTEGER DEFAULT 0,
    enabled                IN BOOLEAN DEFAULT FALSE,
    comments              IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–30 CREATE\_PROGRAM Procedure Parameters**

Parameter	Description
program_name	The name to assign to the program. The name must be unique in the SQL namespace. For example, a program cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
program_type	This attribute specifies the type of program you are creating. If it is not specified then you get an error. There are three supported values for program_type: <ul style="list-style-type: none"> <li>▪ 'PLSQL_BLOCK' This specifies that the program is a PL/SQL block. Job or program arguments are not supported when the job or program type is PLSQL_BLOCK. In this case, the number of arguments must be 0.</li> <li>▪ 'STORED_PROCEDURE' This specifies that the program is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported. PL/SQL procedures with INOUT or OUT arguments are not supported.</li> <li>▪ 'EXECUTABLE' This specifies that the program is external to the database. External programs imply anything that can be executed from the operating system command line. AnyData arguments are not supported with job or program type EXECUTABLE.</li> </ul>

**Table 128–30 (Cont.) CREATE\_PROGRAM Procedure Parameters**

Parameter	Description
<code>program_action</code>	<p>This attribute specifies the action of the program. The following actions are possible:</p> <ul style="list-style-type: none"> <li>For a PL/SQL block, the action is to execute PL/SQL code. These blocks must end with a semicolon.  For example, <code>my_proc();</code> or <code>BEGIN my_proc(); END;</code> or <code>DECLARE arg pls_integer := 10; BEGIN my_proc2(arg); END;</code>.</li> </ul> <p>Note that the Scheduler wraps <code>job_action</code> in its own block and passes the following to PL/SQL for execution: <code>DECLARE ... BEGIN job_action END;</code> This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except <code>event_message</code> in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value. See <a href="#">Table 128–39</a> on page 128-81 for details on available metadata attributes.</p> <p>If it is an anonymous block, special Scheduler metadata may be accessed using the following variable names: <code>job_name</code>, <code>job_owner</code>, <code>job_start</code>, <code>window_start</code>, <code>window_end</code>. For more information, see the <a href="#">"DEFINE_METADATA_ARGUMENT Procedure"</a> on page 128-81.</p> <ul style="list-style-type: none"> <li>For a stored procedure, the action is the name of the stored procedure. You have to specify the schema if the procedure resides in a schema other than the job.  If case sensitivity is needed, enclose the schema name and the store procedure name in double quotes. For example, <code>program_action=&gt;' "Schema" . "Procedure" '</code>.</li> <li>For an executable, the action is the name of the external executable, including the full path name, but excluding any command-line arguments. If the action starts with a single question mark (?), the question mark is replaced by the path to the Oracle home directory for a local job or to the Scheduler agent home for a remote job. If the action contains an at sign (@) and the job is local, the at sign is replaced with the SID of the current Oracle instance.</li> </ul> <p>If <code>program_action</code> is not specified, an error is generated</p>
<code>number_of_arguments</code>	<p>This attribute specifies the number of arguments the program takes. If this parameter is not specified, then the default is 0. A program can have a maximum of 255 arguments.</p> <p>If the <code>program_type</code> is <code>PLSQL_BLOCK</code>, then this parameter is ignored.</p>
<code>enabled</code>	<p>This flag specifies whether the program should be created as enabled or not. If the flag is set to <code>TRUE</code>, then validity checks are made and the program is created as <code>ENABLED</code> if all the checks be successful. By default, this flag is set to <code>FALSE</code>, meaning not created enabled. You can also call the <code>ENABLE</code> procedure to enable the program before it can be used.</p>
<code>comments</code>	<p>A comment about the program. By default, this attribute is <code>NULL</code>.</p>

## Usage Notes

To create a program in their own schema, users need the `CREATE JOB` privilege. A user with the `CREATE ANY JOB` privilege can create a program in any schema. A program is created in a disabled state by default (unless the `enabled` parameter is set to `TRUE`). It cannot be executed by a job until it is enabled.

To use your programs, other users must have `EXECUTE` privileges, therefore once a program has been created, you have to grant `EXECUTE` privileges on it.



**See Also:** ["DEFINE\\_PROGRAM\\_ARGUMENT Procedure"](#) on page 128-83

## CREATE\_SCHEDULE Procedure

This procedure creates a schedule.

### Syntax

```
DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name      IN VARCHAR2,
  start_date         IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
  repeat_interval    IN VARCHAR2,
  end_date           IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
  comments           IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–31 CREATE\_SCHEDULE Procedure Parameters**

Parameter	Description
schedule_name	The name to assign to the schedule. The name must be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
start_date	This attribute specifies the first date and time on which this schedule becomes valid. For a repeating schedule, the value for start_date is a reference date. In this case, the start of the schedule is not the start_date; it depends on the repeat interval specified. start_date is used to determine the first instance of the schedule.  If start_date is specified in the past and no value for repeat_interval is specified, the schedule is invalid. For a repeating job or window, start_date can be derived from the repeat_interval if it is not specified.  If start_date is null, then the date that the job or window is enabled is used. start_date and repeat_interval cannot both be null.
repeat_interval	This attribute specifies how often the schedule repeats. It is expressed using calendaring syntax. See " <a href="#">Calendaring Syntax</a> " on page 128-20 for further information. PL/SQL expressions are not allowed as repeat intervals for named schedules.
end_date	The date and time after which jobs will not run and windows will not open.  A non-repeating schedule that has no end_date is valid forever.  end_date has to be after the start_date. If this is not the case, then an error is generated when the schedule is created.
comments	This attribute specifies an optional comment about the schedule. By default, this attribute is NULL.

### Usage Notes

This procedure requires the CREATE JOB privilege to create a schedule in your own schema or the CREATE ANY JOB privilege to create a schedule in someone else's schema by specifying schema . schedule\_name. Once a schedule has been created, it can be used by other users. The schedule is created with access to PUBLIC. Therefore, there is no need to explicitly grant access to the schedule.

## CREATE\_WINDOW Procedure

This procedure creates a recurring time window and associates it with a resource plan. You can then use the window to schedule jobs that run under the associated resource plan. Windows are created in the `SYS` schema.

The procedure is overloaded.

### Syntax

Creates a window using a named schedule object:

```
DBMS_SCHEDULER.CREATE_WINDOW (
    window_name          IN VARCHAR2,
    resource_plan        IN VARCHAR2,
    schedule_name       IN VARCHAR2,
    duration             IN INTERVAL DAY TO SECOND,
    window_priority     IN VARCHAR2 DEFAULT 'LOW',
    comments            IN VARCHAR2 DEFAULT NULL);
```

Creates a window using an inlined schedule:

```
DBMS_SCHEDULER.CREATE_WINDOW (
    window_name          IN VARCHAR2,
    resource_plan        IN VARCHAR2,
    start_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    repeat_interval     IN VARCHAR2,
    end_date            IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    duration            IN INTERVAL DAY TO SECOND,
    window_priority     IN VARCHAR2 DEFAULT 'LOW',
    comments            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–32 CREATE\_WINDOW Procedure Parameters**

Parameter	Description
<code>window_name</code>	The name to assign to the window. The name must be unique in the SQL namespace. All windows are in the <code>SYS</code> schema, so the preface 'SYS' is optional.
<code>resource_plan</code>	<p>This attribute specifies the resource plan that automatically activates when the window opens. When the window closes, the system switches to the appropriate resource plan, which is usually the plan that was in effect before the window opened, but can also be the plan of a different window.</p> <p>Only one resource plan can be associated with a window. It may be <code>NULL</code> or the empty string (''). When it is <code>NULL</code>, the resource plan in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window.</p> <p>If the window is open and the resource plan is dropped, then the resource allocation for the duration of the window is not affected.</p>

**Table 128–32 (Cont.) CREATE\_WINDOW Procedure Parameters**

Parameter	Description
start_date	<p>This attribute specifies the first date and time on which this window is scheduled to open. If the value for start_date specified is in the past or is not specified, the window opens as soon as it is created.</p> <p>For repeating windows that use a calendaring expression to specify the repeat interval, the value for start_date is a reference date. The first time the window opens depends on the repeat interval specified and the value for start_date.</p>
duration	<p>This attribute specifies how long the window stays open. For example, 'interval '5' hour' for five hours. There is no default value for this attribute. Therefore, if no value is specified when the window is created, an error occurs. The duration is of type interval day to seconds and ranges from one minute to 99 days.</p>
schedule_name	<p>This attribute specifies the name of the schedule associated with the window.</p>
repeat_interval	<p>This attribute specifies how often the window repeats. It is expressed using the Scheduler calendaring syntax. See <a href="#">"Calendaring Syntax"</a> on page 128-20 for more information.</p> <p>A PL/SQL expression cannot be used to specify the repeat interval for a window.</p> <p>The expression specified is evaluated to determine the next time the window opens. If no repeat_interval is specified, the window opens only once at the specified start date.</p>
end_date	<p>This attribute specifies the date and time after which the window no longer opens. When the value for end_date is reached, the window is disabled. In the *_SCHEDULER_WINDOWS views, the enabled flag of the window is set to FALSE.</p> <p>A non-repeating window that has no value for end_date opens only once for the duration of the window. For a repeating window, if no end_date is specified, then the window keeps repeating forever.</p> <p>The end_date must be after the start_date. If it is not, then an error is generated when the window is created.</p>
window_priority	<p>This attribute is only relevant when two windows overlap. Because only one window can be in effect at one time, the window priority determines which window opens. The two possible values for this attribute are 'HIGH' and 'LOW'. A high priority window has precedence over a low priority window, therefore, the low priority window does not open if it overlaps a high priority window. By default, windows are created with priority 'LOW'.</p>
comments	<p>This attribute specifies an optional comment about the window. By default, this attribute is NULL.</p>

## Usage Notes

Creating a window requires the `MANAGE SCHEDULER` privilege.

Scheduler windows are the principal mechanism used to automatically switch resource plans according to a schedule. You can also manually activate a resource plan by using the `ALTER SYSTEM SET RESOURCE_MANAGER_PLAN` statement or the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure. Note that either of these manual methods can also disable resource plan switching by Scheduler

windows. For more information, see *Oracle Database Administrator's Guide* and "[SWITCH\\_PLAN Procedure](#)" on page 119-37.

## CREATE\_WINDOW\_GROUP Procedure

**\*\*\* Deprecated in Oracle Database 11g Release 2 (11.2). Use CREATE\_GROUP instead.**

This procedure creates a new window group. A window group is defined by a list of Scheduler windows. You can assign a window group as a job schedule. The job then runs when any of the windows in the group become active.

Window groups are created in the SYS schema.

### Syntax

```
DBMS_SCHEDULER.CREATE_WINDOW_GROUP (
  group_name          IN VARCHAR2,
  window_list        IN VARCHAR2 DEFAULT NULL,
  comments            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–33 CREATE\_WINDOW\_GROUP Procedure Parameters**

Parameter	Description
group_name	The name to assign to the window group
window_list	A list of the windows assigned to the window group. If a window that does not exist is specified, an error is generated and the window group is not created.  Windows can also be added using the ADD_WINDOW_GROUP_MEMBER procedure. A window group cannot be a member of another window group.  Can be NULL.
comments	A comment about the window group

### Usage Notes

Creating a window group requires the `MANAGE SCHEDULER` privilege. Window groups, like windows, are created with access to `PUBLIC`, therefore, no privileges are required to access window groups.

A window group cannot contain another window group.

## DEFINE\_ANYDATA\_ARGUMENT Procedure

This procedure defines a name or default value for a program argument that is of a complex type and must be encapsulated within an ANYDATA object. A job that references the program can override the default value.

### Syntax

```
DBMS_SCHEDULER.DEFINE_ANYDATA_ARGUMENT (
  program_name          IN VARCHAR2,
  argument_position     IN PLS_INTEGER,
  argument_name         IN VARCHAR2 DEFAULT NULL,
  argument_type         IN VARCHAR2,
  default_value         IN SYS.ANYDATA,
  out_argument          IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–34** DEFINE\_ANYDATA\_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_position	The position of the argument as it is passed to the executable. Argument numbers go from one to the <code>number_of_arguments</code> specified for the program. This must be unique, so it can replace any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures, including the <a href="#">SET_JOB_ANYDATA_VALUE Procedure</a> .
argument_type	The data type of the argument being defined. This is not verified or used by the Scheduler. It is only used by the user of the program when deciding what value to assign to the argument.
default_value	The default value to be assigned to the argument encapsulated within an AnyData object. This is optional.
out_argument	This parameter is reserved for future use. It must be set to FALSE.

### Usage Notes

All program arguments from one to the `number_of_arguments` value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

#### See Also:

- ["DEFINE\\_PROGRAM\\_ARGUMENT Procedure"](#) on page 128-83
- ["SET\\_JOB\\_ANYDATA\\_VALUE Procedure"](#) on page 128-146

## DEFINE\_CHAIN\_EVENT\_STEP Procedure

This procedure adds or replaces a chain step and associates it with an event schedule or an inline event. Once started in a running chain, this step does not complete until the specified event has occurred. Every step in a chain must be defined before the chain can be enabled and used. Defining a step gives it a name and specifies what happens during the step. If a step already exists with this name, the new step replaces the old one.

### Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
    chain_name          IN VARCHAR2,
    step_name           IN VARCHAR2,
    event_schedule_name IN VARCHAR2,
    timeout             IN INTERVAL DAY TO SECOND DEFAULT NULL);
```

```
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
    chain_name          IN VARCHAR2,
    step_name           IN VARCHAR2,
    event_condition     IN VARCHAR2,
    queue_spec          IN VARCHAR2,
    timeout             IN INTERVAL DAY TO SECOND DEFAULT NULL);
```

### Parameters

**Table 128–35** DEFINE\_CHAIN\_EVENT\_STEP Procedure Parameters

Parameter	Description
chain_name	The name of the chain that the step is in
step_name	The name of the step
event_schedule_name	The name of the event schedule that the step waits for
timeout	This parameter is reserved for future use
event_condition	See the <a href="#">CREATE_EVENT_SCHEDULE Procedure</a>
queue_spec	See the <a href="#">CREATE_EVENT_SCHEDULE Procedure</a>

### Usage Notes

Defining a chain step requires ALTER privileges on the chain either as the owner of the chain, or as a user with the ALTER object privilege on the chain or the CREATE ANY JOB system privilege.

You can base a chain step on a file watcher as well. To do this, provide the file watcher name directly in the queue\_spec parameter, or use a file watcher schedule for the event\_schedule\_name parameter.

**See Also:** ["DEFINE\\_CHAIN\\_STEP Procedure"](#) on page 128-80



## DEFINE\_CHAIN\_RULE Procedure

This procedure adds a new rule to an existing chain, specified as a condition-action pair. The condition is expressed using either SQL or the Scheduler chain condition syntax and indicates the prerequisites for the action to occur. The action is a result of the condition being met.

An actual rule object is created to store the rule in the schema where the chain resides. If a rule name is given, this name is used for the rule object. If an existing rule name in the schema of the chain is given, the existing rule is altered. (A schema different than the schema of the chain cannot be specified). If no rule name is given, one is generated in the form `SCHED_RULE${N}`.

### Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
  chain_name          IN VARCHAR2,
  condition           IN VARCHAR2,
  action             IN VARCHAR2,
  rule_name          IN VARCHAR2 DEFAULT NULL,
  comments           IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–36** *DEFINE\_CHAIN\_RULE Procedure Parameters*

Parameter	Description
chain_name	The name of the chain to alter
condition	<p>A boolean expression which must evaluate to TRUE for the action to be performed. Every chain must have a rule that evaluates to TRUE to start the chain. For this purpose, you can use a rule that has 'TRUE' as its condition if you are using Scheduler chain condition syntax, or '1=1' as its condition if you are using SQL syntax.</p> <ul style="list-style-type: none"> <li>■ Scheduler Chain Condition Syntax           <p>See "<a href="#">Scheduler Chain Condition Syntax</a>" on page 128-78 for details</p> </li> <li>■ SQL WHERE Clause Syntax           <p>Conditions expressed with SQL must use the syntax of a SELECT statement WHERE clause.</p> <p>You can refer to chain step attributes by using the chain step name as a bind variable.</p> <p>The bind variable syntax is <code>:step_name.attribute</code>. (<code>step_name</code> refers to a typed object.) Possible attributes are: <code>completed</code>, <code>state</code>, <code>start_date</code>, <code>end_date</code>, <code>error_code</code>, and <code>duration</code>.</p> <p>Possible values for the state attribute include: 'NOT_STARTED', 'SCHEDULED', 'RUNNING', 'PAUSED', 'STALLED', 'SUCCEEDED', 'FAILED', and 'STOPPED'. If a step is in the state 'SUCCEEDED', 'FAILED', or 'STOPPED', its <code>completed</code> attribute is set to 'TRUE', otherwise <code>completed</code> is 'FALSE'.</p> </li> </ul>

**Table 128–36 (Cont.) DEFINE\_CHAIN\_RULE Procedure Parameters**

Parameter	Description
<code>action</code>	<p>The action to be performed when the rule evaluates to TRUE. The action must consist of at least one keyword with an optional value and an optional delay clause.</p> <p>Possible actions include:</p> <ul style="list-style-type: none"> <li>▪ <code>[AFTER <i>delay_interval</i>] START <i>step_1</i> [, <i>step_2</i> ...]</code></li> <li>▪ <code>STOP <i>step_1</i> [, <i>step_2</i> ...]</code></li> <li>▪ <code>END [{<i>end_value</i>   <i>step_name.error_code</i>}]</code></li> </ul> <p>At the beginning of the START action, a delay clause can specify a delay interval before performing the action. <i>delay_interval</i> is a formatted datetime interval of the form HH:MM:SS.</p> <p>The END action ends the chain with an error code equal to either the supplied <i>end_value</i> or the error code that <i>step_name</i> completes with. The default error code is 0, indicating a successful chain run.</p>
<code>rule_name</code>	<p>The name of the rule being created. If no <code>rule_name</code> is given, one is generated in the form <code>SCHED_RULE\$_{N}</code>.</p>
<code>comments</code>	<p>An optional comment describing the rule. This is stored in the rule object created.</p>

### Scheduler Chain Condition Syntax

The Scheduler chain condition syntax provides an easy way to construct a condition using the states and error codes of steps in the current chain. The following are the available constructs, which are all boolean expressions:

```
TRUE
FALSE
stepname [NOT] SUCCEEDED
stepname [NOT] FAILED
stepname [NOT] STOPPED
stepname [NOT] COMPLETED
stepname ERROR_CODE IN (integer, integer, integer ...)
stepname ERROR_CODE NOT IN (integer, integer, integer ...)
stepname ERROR_CODE = integer
stepname ERROR_CODE != integer
stepname ERROR_CODE <> integer
stepname ERROR_CODE > integer
stepname ERROR_CODE >= integer
stepname ERROR_CODE < integer
stepname ERROR_CODE <= integer
```

These boolean operators are available to create more complex conditions:

```
expression AND expression
expression OR expression
NOT (expression)
```

*integer* can be positive or negative. Parentheses may be used for clarity or to enforce ordering. You must use parentheses with the NOT operator.

PL/SQL code that runs as part of a step can set the value of `ERROR_CODE` for that step with the `RAISE_APPLICATION_ERROR` statement.

## Usage Notes

Defining a chain rule requires ALTER privileges on the chain (either as the owner, or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

You must define at least one rule that starts the chain and at least one that ends it. See the section "Adding Rules to a Chain" in *Oracle Database Administrator's Guide* for more information.

## Examples

The following are examples of using rule conditions and rule actions.

### Rule Conditions Using Scheduler Chain Condition Syntax

```
'step1 completed'
-- satisfied when step step1 has completed. (step1 completed is also TRUE when any
-- of the following are TRUE: step1 succeeded, step1 failed, step1 stopped.)

'step1 succeeded and step2 succeeded'
-- satisfied when steps step1 and step2 have both succeeded

'step1 error_code > 100'
-- satisfied when step step1 has failed with an error_code greater than 100

'step1 error_code IN (1, 3, 5, 7) '
-- satisfied when step step1 has failed with an error_code of 1, 3, 5, or 7
```

### Rule Conditions Using SQL Syntax

```
':step1.completed = ''TRUE'' AND :step1.end_date >SYSDATE-1/24'
--satisfied when step step1 completed less than an hour ago

':step1.duration > interval '5' minute'
-- satisfied when step step1 has completed and took longer than 5 minutes to
complete
```

### Rule Actions

```
'AFTER 01:00:00 START step1, step2'
--After an hour start steps step1 and step2

'STOP step1'
--Stop step step1

END step4.error_code'
--End the chain with the error code that step step4 finished with. If step4 has
not completed, the chain will be ended unsuccessfully with error code 27435.

'END' or 'END 0'
--End the chain successfully (with error_code 0)

'END 100'
--End the chain unsuccessfully with error code 100.
```

## DEFINE\_CHAIN\_STEP Procedure

This procedure adds or replaces a chain step and associates it with a program or a nested chain. When the chain step is started, the specified program or chain is run. If a step already exists with the name supplied in the `chain_name` argument, the new step replaces the old one.

The chain owner must have EXECUTE privileges on the program or chain associated with the step. Only one program or chain can run during a step.

You cannot set all possible step attributes with this procedure. Use the ALTER\_CHAIN procedure to set additional chain step attributes, such as `credential_name` and `destination_name`.

### Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
    chain_name          IN VARCHAR2,
    step_name          IN VARCHAR2,
    program_name       IN VARCHAR2);
```

### Parameters

**Table 128-37** DEFINE\_CHAIN\_STEP Procedure Parameters

Parameter	Description
<code>chain_name</code>	The name of the chain to alter.
<code>step_name</code>	The name of the step being defined. If a step already exists with this name, the new step replaces the old one.
<code>program_name</code>	The name of a program or chain to run during this step. The chain owner must have EXECUTE privileges on this program or chain.

### Usage Notes

Defining a chain step requires ALTER privileges on the chain (either as the owner, or a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

**See Also:**

- ["ALTER\\_CHAIN Procedure"](#) on page 128-40
- ["DEFINE\\_CHAIN\\_EVENT\\_STEP Procedure"](#) on page 128-76

## DEFINE\_METADATA\_ARGUMENT Procedure

This procedure defines a special metadata argument for the program. The Scheduler can pass Scheduler metadata through this argument to your stored procedure or other executable. You cannot set values for jobs using this argument.

### Syntax

```
DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT (
  program_name          IN VARCHAR2,
  metadata_attribute   IN VARCHAR2,
  argument_position    IN PLS_INTEGER,
  argument_name        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–38** DEFINE\_METADATA\_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered
metadata_attribute	The metadata to be passed. Valid metadata attributes are: 'job_name', 'job_subname', 'job_owner', 'job_start', 'window_start', 'window_end', and 'event_message'. <a href="#">Table 128–39</a> describes these attributes in detail.
argument_position	The position of the argument as it is passed to the executable. The position cannot be greater than the number_of_arguments specified for the program. It must be unique, so it replaces any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures.

**Table 128–39** Metadata Attributes

Metadata Attribute	Data Type	Description
job_name	VARCHAR2	Name of the currently running job
job_subname	VARCHAR2	Subname of the currently running job. The name + subname form a unique identifier for a job that is running a chain step. NULL if the job is not part of a chain.
job_owner	VARCHAR2	Owner of the currently running job
job_scheduled_start	TIMESTAMP WITH TIME ZONE	When the currently running job was scheduled to start
job_start	TIMESTAMP WITH TIME ZONE	When the currently running job started

**Table 128–39 (Cont.) Metadata Attributes**

Metadata Attribute	Data Type	Description
window_start	TIMESTAMP WITH TIME ZONE	If the job was started by a window, the time that the window opened
window_end	TIMESTAMP WITH TIME ZONE	If the job was started by a window, the time that the window is scheduled to close
event_message	(See Description)	For an event-based job, the message content of the event that started the job. The data type of this attribute depends on the queue used for the event. It has the same type as the USER_DATA column of the queue table. In the case of a file arrival event, event_message is of type SYS.SCHEDULER_FILEWATCHER_RESULT. See <a href="#">"SCHEDULER_FILEWATCHER_RESULT Object Type"</a> on page 15.

### Usage Notes

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

All metadata attributes except event\_message can be used in PL/SQL blocks that you enter into the job\_action or program\_action attributes of jobs or programs, respectively. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value.

## DEFINE\_PROGRAM\_ARGUMENT Procedure

This procedure defines a name or default value for a program argument. If no default value is defined for a program argument, the job that references the program must supply an argument value. (The job can also override a default value.)

This procedure is overloaded.

### Syntax

Defines a program argument without a default value:

```
PROCEDURE define_program_argument (
    program_name          IN VARCHAR2,
    argument_position     IN PLS_INTEGER,
    argument_name         IN VARCHAR2 DEFAULT NULL,
    argument_type         IN VARCHAR2,
    out_argument          IN BOOLEAN DEFAULT FALSE);
```

Defines a program argument with a default value:

```
PROCEDURE define_program_argument (
    program_name          IN VARCHAR2,
    argument_position     IN PLS_INTEGER,
    argument_name         IN VARCHAR2 DEFAULT NULL,
    argument_type         IN VARCHAR2,
    default_value         IN VARCHAR2,
    out_argument          IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–40** *DEFINE\_PROGRAM\_ARGUMENT Procedure Parameters*

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_position	The position of the argument as it is passed to the executable. Argument numbers go from one to the <code>number_of_arguments</code> specified for the program. This must be unique so it replaces any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if specified. If you assign a name, the name can then be used by other package procedures, including the <a href="#">SET_JOB_ARGUMENT_VALUE Procedure</a> .
argument_type	The data type of the argument being defined. This is not verified or used by the Scheduler. The program user uses <code>argument_type</code> when deciding what value to assign to the argument. Any valid SQL data type is allowed.
default_value	The default value to be assigned to the argument if none is specified by the job.
out_argument	This parameter is reserved for future use. It must be set to <code>FALSE</code> .

### Usage Notes

All program arguments from 1 to the `number_of_arguments` value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

DEFINE\_PROGRAM\_ARGUMENT only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

**See Also:**

- ["DEFINE\\_ANYDATA\\_ARGUMENT Procedure"](#) on page 128-75
- ["SET\\_JOB\\_ARGUMENT\\_VALUE Procedure"](#) on page 128-147



## DISABLE Procedure

This procedure disables a program, job, chain, window, database destination, external destination, file watcher, or group. When an object is disabled, its `enabled` attribute is set to `FALSE`.

### Syntax

```
DBMS_SCHEDULER.DISABLE (
    name           IN VARCHAR2,
    force          IN BOOLEAN DEFAULT FALSE,
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

### Parameters

**Table 128–41** *DISABLE Procedure Parameters*

Parameter	Description
<code>name</code>	<p>The name of the object being disabled. Can be a comma-delimited list.</p> <p>If a job class name is specified, then all the jobs in the job class are disabled. The job class is not disabled.</p> <p>If a group name is specified, then the group is disabled, but the enabled state of the group members is unaffected.</p>
<code>force</code>	<p>If <code>TRUE</code>, objects are disabled even if other objects depend on them. See the usage notes for more information.</p>
<code>commit_semantics</code>	<p>The commit semantics. The following types are supported:</p> <ul style="list-style-type: none"> <li> <p>■ <code>STOP_ON_FIRST_ERROR</code>: The procedure returns on the first error and the previous disable operations that were successful are committed to disk.</p> <p>This is the default.</p> </li> <li> <p>■ <code>TRANSACTIONAL</code>: The procedure returns on the first error and everything that happened before that error is rolled back.</p> <p>This type is only supported when disabling a job or a list of jobs. In addition, this type is not supported when <code>force</code> is set to <code>TRUE</code>.</p> </li> <li> <p>■ <code>ABSORB_ERRORS</code>: The procedure tries to absorb any errors and disable the rest of the jobs and commits all the disable operations that were successful. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details.</p> <p>This type is only supported when disabling a job or a list of jobs.</p> </li> </ul>

### Usage Notes

Windows must be preceded by `SYS`.

Disabling an object that is already disabled does not generate an error.

The purpose of the `force` option is to point out dependencies. No dependent objects are altered.

To run `DISABLE` for a window or a group of type `WINDOW`, you must have the `MANAGE SCHEDULER` privilege.

You can use `DISABLE` with any schema except the `SYS` schema.

### Jobs

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator will not pick up these jobs for processing. When a job is disabled, its `state` in the job queue is changed to `disabled`.

If `force` is set to `FALSE` and the job is currently running, an error is returned.

If `force` is set to `TRUE`, the job is disabled, but the currently running instance is allowed to finish.

For jobs with multiple destinations, you cannot disable a child job at a specific destination. Instead, you can disable the destination.

### Programs

When a program is disabled, the status is changed to `disabled`. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

If `force` is set to `FALSE`, the program must not be referenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`, those jobs that point to the program will not be disabled, however, they will fail at runtime because their program will not be valid.

Running jobs that point to the program are not affected by the `DISABLE` call and are allowed to continue.

No arguments that pertain to the program are affected when the program is disabled.

### File Watchers

If `force` is set to `FALSE`, the file watcher must not be referenced by any job, otherwise an error will occur. If you force disabling a file watcher, jobs that depend on it become disabled.

### Windows

This means that the window will not open, however, the metadata of the window is still there, so it can be reenabled.

If `force` is set to `FALSE`, the window must not be open or referenced by any job otherwise an error occurs.

If `force` is set to `TRUE`, disabling a window that is open will succeed but the window will not be closed. It will prevent the window from opening in the future until it is reenabled.

When the window is disabled, those jobs that have the window as their schedule will not be disabled.

### Window Groups

When a group of type `WINDOW` is disabled, jobs (other than a running job) that have the window group as their schedule will not run when the member windows open. However, a job that has one of the window group members as its schedule still runs.

The metadata of the window group is still there, so it can be reenabled. Note that the members of the window group will still open.

If `force` is set to `FALSE`, the window group must not have any members that are open or referenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`:

- The window group is disabled and the open window will be not closed or disabled. It will be allowed to continue to its end.
- The window group is disabled but those jobs that have the window group as their schedule will not be disabled.

### **Job Chains**

When a chain is disabled, the metadata for the chain is still there, but jobs that point to it will not be able to be run. This allows changes to the chain to be made safely without the risk of having an incompletely specified chain run.

If `force` is set to `FALSE`, the chain must not be referenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`, those jobs that point to the chain will not be disabled, however, they will fail at runtime.

Running jobs that point to this chain are not affected by the `DISABLE` call and are allowed to complete.

### **Database Destinations**

When you disable a database destination:

- The destination is skipped when a multiple destination job runs.
- If all destinations are disabled for a job, the Scheduler generates an error when it attempts to run the job.
- The `REFS_ENABLED` column in `*_SCHEDULER_JOB_DESTS` is set to `FALSE` for all jobs that reference the database destination.

### **External Destinations**

When you disable an external destination:

- Dependent database destinations remain enabled, but the Scheduler generates an error when it attempts to run a job with a database destination that depends on the external destination.
- The `REFS_ENABLED` column in `*_SCHEDULER_JOB_DESTS` is set to `FALSE` for all external jobs that reference the external destination and for all database jobs with a database destination that depends on the external destination.

### **Groups**

If you disable an external destination group or database destination group, the Scheduler generates an error when it attempts to run a job that names the group as its destination.

## DROP\_AGENT\_DESTINATION Procedure

This procedure drops one or more external destinations, also known as agent destinations. It should be used only when the preferred method of dropping an external destination—using the `schagent` utility to unregister a Scheduler agent with a database—is unavailable due to failures.

This procedure can be called only by the `SYS` user or a user with the `MANAGE SCHEDULER` privilege.

---

**Note:** External destinations are created on a source database only implicitly by registering an agent with the database. There is no user-callable `CREATE_AGENT_DESTINATION` procedure.

---

### Syntax

```
DBMS_SCHEDULER.DROP_AGENT_DESTINATION (
    destination_name          IN VARCHAR2);
```

### Parameters

**Table 128–42 DROP\_AGENT\_DESTINATION Procedure Parameters**

Parameter	Description
<code>destination_name</code>	<p>A comma-separated list of external destinations to drop. Because user <code>SYS</code> owns all external destinations, do not prefix them with a schema name.</p> <p>The procedure stops processing if it encounters an external destination that does not exist. All external destinations processed before the error are dropped.</p> <p>Cannot be <code>NULL</code>.</p>

### Usage Notes

When an external destination is dropped:

- All database destinations that refer to the external destination are disabled and their `agent` attribute is set to `NULL`.
- Members of external destination groups that refer to the destination are removed from the group.
- All job instances in the `*_SCHEDULER_JOB_DESTS` views that refer to the external destination are also dropped.
- Jobs running against the destination are stopped.

## DROP\_CHAIN Procedure

This procedure drops an existing chain.

### Syntax

```
DBMS_SCHEDULER.DROP_CHAIN (
    chain_name          IN VARCHAR2,
    force               IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–43** *DROP\_CHAIN Procedure Parameters*

Parameter	Description
chain_name	The name of the chain to drop. Can also be a comma-delimited list of chains.
force	<p>If <i>force</i> is set to <code>FALSE</code>, the chain must not be referenced by any job, otherwise an error will occur.</p> <p>If <i>force</i> is set to <code>TRUE</code>, all jobs pointing to the chain are disabled before the chain is dropped.</p> <p>Running jobs that point to this chain are stopped before the chain is dropped.</p>

### Usage Notes

Dropping a chain requires alter privileges on the chain (either as the owner, or a user with `ALTER` privileges on the chain or the `CREATE ANY JOB` system privilege).

All steps associated with the chain are dropped. If no rule set was specified when the chain was created, then the automatically created rule set and evaluation context associated with the chain are also dropped, so the user must have the privileges required to do this. See the `DBMS_RULE_ADM.DROP_RULE_SET` and `DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT` procedures for more information.

If *force* is `FALSE`, no jobs may be using this chain. If *force* is `TRUE`, any jobs that use this chain are disabled before the chain is dropped (and any of these jobs that are running will be stopped).

## DROP\_CHAIN\_RULE Procedure

This procedure removes a rule from an existing chain. The rule object corresponding to this rule will also be dropped. The chain will not be disabled. If dropping this rule makes the chain invalid, the user should first disable the chain to ensure that it does not run.

### Syntax

```
DBMS_SCHEDULER.DROP_CHAIN_RULE (  
    chain_name          IN VARCHAR2,  
    rule_name           IN VARCHAR2,  
    force               IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–44** DROP\_CHAIN\_RULE Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
rule_name	The name of the rule to drop
force	If <i>force</i> is set to TRUE, the drop operation proceeds even if the chain is currently running. The running chain is not stopped or interrupted. If <i>force</i> is set to FALSE and the chain is running, an error is generated.

### Usage Notes

Dropping a chain rule requires alter privileges on the chain (either as the owner or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

Dropping a chain rule also drops the underlying rule database object so you must have the privileges to drop this rule object. See the DBMS\_RULE\_ADM.DROP\_RULE procedure for more information.

## DROP\_CHAIN\_STEP Procedure

This procedure drops a chain step. If this chain step is still used in the chain rules, the chain will be disabled.

### Syntax

```
DBMS_SCHEDULER.DROP_CHAIN_STEP (
  chain_name          IN VARCHAR2,
  step_name           IN VARCHAR2,
  force               IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–45** *DROP\_CHAIN\_STEP Procedure Parameters*

Parameter	Description
chain_name	The name of the chain to alter
step_name	The name of the step being dropped. Can be a comma-separated list.
force	If <i>force</i> is set to <b>TRUE</b> , this succeeds even if this chain is currently running. The running chain will not be stopped or interrupted.  If <i>force</i> is set to <b>FALSE</b> and this chain is currently running, an error is thrown.

### Usage Notes

Dropping a chain step requires **ALTER** privileges on the chain (either as the owner or as a user with **ALTER** privileges on the chain or the **CREATE ANY JOB** system privilege).

## DROP\_CREDENTIAL Procedure

This procedure drops a credential.

### Syntax

```
DBMS_SCHEDULER.DROP_CREDENTIAL (  
    credential_name      IN VARCHAR2,  
    force                 IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–46** *DROP\_CREDENTIAL Procedure Parameters*

Parameter	Description
credential_name	The name of the credential being dropped. This can optionally be prefixed with a schema name. This cannot be set to NULL.
force	If set to <code>FALSE</code> , the credential must not be referenced by any job, or an error will occur. If set to <code>TRUE</code> , the credential is dropped whether or not there are jobs referencing it. Jobs that reference the credential will continue to point to a nonexistent credential and throw an error at runtime.

### Usage Notes

Only the owner of a credential or a user with the `CREATE ANY JOB` system privilege may drop the credential.

Running jobs that point to the credential are not affected by this procedure and are allowed to continue.

**See Also:** "[CREATE\\_CREDENTIAL Procedure](#)" on page 128-49



## DROP\_DATABASE\_DESTINATION Procedure

This procedure drops one or more database destinations.

### Syntax

```
DBMS_SCHEDULER.DROP_DATABASE_DESTINATION (
    destination_name      IN VARCHAR2);
```

### Parameters

**Table 128–47** *DROP\_DATABASE\_DESTINATION Procedure Parameters*

Parameter	Description
destination_name	<p>The name of the destination to drop. Can be a comma-separated list of database destinations to drop. Each database destination can optionally be prefixed with a schema name.</p> <p>The procedure stops processing if it encounters a database destination that does not exist. All database destinations processed before the error are dropped.</p> <p>Cannot be NULL.</p>

### Usage Notes

Only the owner or a user with the `CREATE ANY JOB` system privilege may drop the database destination.

When a database destination is dropped:

- All job instances that refer to the destination in the `*_SCHEDULER_JOB_DESTS` views are also dropped.
- Jobs running against the destination are stopped.
- Members of database destination groups that refer to the destination are removed from the group.

**See Also:** [CREATE\\_DATABASE\\_DESTINATION Procedure](#) on page 128-50

## DROP\_FILE\_WATCHER Procedure

This procedure drops one or more file watchers.

### Syntax

```
DBMS_SCHEDULER.DROP_FILE_WATCHER (  
    file_watcher_name      IN VARCHAR2,  
    force                  IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–48** *DROP\_FILE\_WATCHER Procedure Parameters*

Parameter	Description
file_watcher_name	The file watcher to drop. Can be a comma-separated list of file watchers. Each file watcher name can optionally be prefixed with a schema name.  Cannot be NULL.
force	If set to <code>FALSE</code> , the file watcher must not be referenced by any job, or an error occurs. If set to <code>TRUE</code> , the file watcher is dropped whether or not there are jobs referencing it. In this case, jobs that reference the dropped file watcher are disabled.

### Usage Notes

Only the owner of a file watcher or a user with the `CREATE ANY JOB` system privilege may drop the file watcher.

Running jobs that point to the file watcher are not affected by this procedure and are allowed to continue.

**See Also:** ["CREATE\\_FILE\\_WATCHER Procedure"](#) on page 128-53

## DROP\_GROUP Procedure

This procedure drops one or more groups.

### Syntax

```
DBMS_SCHEDULER.DROP_GROUP (
  group_name      IN VARCHAR2,
  force           IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–49** *DROP\_GROUP Procedure Parameters*

Parameter	Description
group_name	A group to drop. Can be a comma-separated list of group names. Each group name can optionally be prefixed with a schema name.  The procedure stops processing if it encounters a group that does not exist. All groups processed before the error are dropped.  Cannot be NULL.
force	If FALSE, the group must not be referenced by any job, otherwise an error occurs. If TRUE, the group is dropped whether or not there are jobs referencing it. In this case, all jobs referencing the group are disabled and all job instances that reference the group are removed from the *_SCHEDULER_JOB_DESTS views.

### Usage Notes

Only the owner or a user with the CREATE ANY JOB system privilege may drop a group. You must have the MANAGE SCHEDULER privilege to drop a group of type WINDOW.

**See Also:** ["CREATE\\_FILE\\_WATCHER Procedure"](#) on page 128-53

## DROP\_JOB Procedure

This procedure drops one or more jobs or all jobs in one or more job classes. Dropping a job also drops all argument values set for that job.

### Syntax

```
DBMS_SCHEDULER.DROP_JOB (
  job_name          IN VARCHAR2,
  force             IN BOOLEAN DEFAULT FALSE,
  defer            IN BOOLEAN DEFAULT FALSE,
  commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

### Parameters

**Table 128–50 DROP\_JOB Procedure Parameters**

Parameter	Description
job_name	The name of a job or job class. Can be a comma-delimited list. For a job class, the SYS schema should be specified.  If the name of a job class is specified, the jobs that belong to that job class are dropped, but the job class itself is not dropped.
force	If force is set to TRUE, the Scheduler first attempts to stop the running job instances (by issuing the STOP_JOB call with the force flag set to false), and then drops the jobs.
defer	If defer is set to TRUE, the Scheduler allows the running jobs to complete and then drops the jobs.
commit_semantics	The commit semantics. The following types are supported: <ul style="list-style-type: none"> <li>■ STOP_ON_FIRST_ERROR returns on the first error and previous successful drop operations are committed to disk. This is the default.</li> <li>■ TRANSACTIONAL returns on the first error. Everything that happened before that error is rolled back. This type is not supported when force is set to TRUE.</li> <li>■ ABSORB_ERRORS tries to absorb any errors and drop the rest of the jobs, and commits all the successful drops. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details.</li> </ul> <p>Only STOP_ON_FIRST_ERROR is permitted when job classes are included in the job_name list.</p>

### Usage Notes

If both force and defer are set to FALSE and a job is running at the time of the call, the attempt to drop that job fails. The entire call to DROP\_JOB may then fail, depending on the setting of commit\_semantics.

Setting both force and defer to TRUE results in an error.

Dropping a job requires ALTER privileges on the job either as the owner of the job or as a user with the ALTER object privilege on the job or the CREATE ANY JOB system privilege.

## DROP\_JOB\_CLASS Procedure

This procedure drops a job class. Dropping a job class means that all the metadata about the job class is removed from the database.

### Syntax

```
DBMS_SCHEDULER.DROP_JOB_CLASS (
  job_class_name      IN VARCHAR2,
  force               IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–51** *DROP\_JOB\_CLASS Procedure Parameters*

Parameter	Description
job_class_name	The name of the job class. Can be a comma-delimited list.
force	<p>If <i>force</i> is set to <code>FALSE</code>, a class being dropped must not be referenced by any jobs, otherwise an error occurs.</p> <p>If <i>force</i> is set to <code>TRUE</code>, jobs belonging to the class are disabled and their class is set to the default class. Only if this is successful is the class dropped.</p> <p>Running jobs that belong to the job class are not affected.</p>

### Usage Notes

Dropping a job class requires the `MANAGE SCHEDULER` system privilege.

## DROP\_PROGRAM Procedure

This procedure drops a program. Any arguments that pertain to the program are also dropped when the program is dropped.

### Syntax

```
DBMS_SCHEDULER.DROP_PROGRAM (  
    program_name          IN VARCHAR2,  
    force                 IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–52** *DROP\_PROGRAM Procedure Parameters*

Parameter	Description
program_name	The name of the program to be dropped. Can be a comma-delimited list.
force	If <i>force</i> is set to <i>FALSE</i> , the program must not be referenced by any job, otherwise an error occurs.  If <i>force</i> is set to <i>TRUE</i> , all jobs referencing the program are disabled before the program is dropped.  Running jobs that point to the program are not affected by the <i>DROP_PROGRAM</i> call and are allowed to continue.

### Usage Notes

Dropping a program requires that you be the owner of the program or have *ALTER* privileges on that program. You can also drop a program if you have the *CREATE ANY JOB* privilege.

## DROP\_PROGRAM\_ARGUMENT Procedure

This procedure drops a program argument. An argument can be specified by either name (if one has been given) or position.

The procedure is overloaded.

### Syntax

Drops a program argument by position:

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
    program_name          IN VARCHAR2,
    argument_position     IN PLS_INTEGER);
```

Drops a program argument by name:

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
    program_name          IN VARCHAR2,
    argument_name         IN VARCHAR2);
```

### Parameters

**Table 128–53** *DROP\_PROGRAM\_ARGUMENT Procedure Parameters*

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_name	The name of the argument being dropped
argument_position	The position of the argument to be dropped

### Usage Notes

Dropping a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also drop a program argument if you have the CREATE ANY JOB privilege.

## DROP\_SCHEDULE Procedure

This procedure drops a schedule.

### Syntax

```
DBMS_SCHEDULER.DROP_SCHEDULE (  
    schedule_name    IN VARCHAR2,  
    force            IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–54** *DROP\_SCHEDULE Procedure Parameters*

Parameter	Description
schedule_name	The name of the schedule. Can be a comma-delimited list.
force	If <i>force</i> is set to <code>FALSE</code> , the schedule must not be referenced by any job or window, otherwise an error will occur.  If <i>force</i> is set to <code>TRUE</code> , any jobs or windows that use this schedule are disabled before the schedule is dropped  Running jobs and open windows that point to the schedule are not affected.

### Usage Notes

You must be the owner of the schedule being dropped or have `ALTER` privileges for the schedule or the `CREATE ANY JOB` privilege.



## DROP\_WINDOW Procedure

This procedure drops a window. All metadata about the window is removed from the database. The window is removed from any groups that reference it.

### Syntax

```
DBMS_SCHEDULER.DROP_WINDOW (
    window_name          IN VARCHAR2,
    force                 IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–55** *DROP\_WINDOW Procedure Parameters*

Parameter	Description
window_name	The name of the window. Can be a comma-delimited list.
force	<p>If <code>force</code> is set to <code>FALSE</code>, the window must be not be open or referenced by any job, otherwise an error occurs.</p> <p>If <code>force</code> is set to <code>TRUE</code>, the window is dropped and those jobs that have the window as their schedule are disabled. However, jobs that have a window group, of which the dropped window is a member, as their schedule, are not disabled. If the window is open then, the Scheduler attempts to first close the window and then drop it. When the window is closed, normal close window rules apply.</p> <p>Running jobs that have the window as their schedule is allowed to continue, unless the <code>stop_on_window_close</code> flag is set to <code>TRUE</code> for the job. If this is the case, the job is stopped when the window is dropped.</p>

### Usage Notes

Dropping a window requires the `MANAGE SCHEDULER` privilege.

## DROP\_WINDOW\_GROUP Procedure

\*\*\* **Deprecated in Oracle Database 11g Release 2 (11.2). Use DROP\_GROUP instead.**

This procedure drops a window group but not the windows that are members of this window group.

### Syntax

```
DBMS_SCHEDULER.DROP_WINDOW_GROUP (  
    group_name          IN VARCHAR2  
    force               IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–56** *DROP\_WINDOW\_GROUP Procedure Parameters*

Parameter	Description
group_name	The name of the window group
force	<p>If <code>force</code> is set to <code>FALSE</code>, the window group must not be referenced by any job otherwise an error will occur.</p> <p>If <code>force</code> is set to <code>TRUE</code>, the window group is dropped and those jobs that have the window group as their schedule are disabled. Running jobs that have the window group as their schedule are allowed to continue, even if the <code>stop_on_window_close</code> flag was set to <code>TRUE</code> when for the job.</p> <p>If a member of the window group that is being dropped is open, the window group can still be dropped.</p>

### Usage Notes

If you want to drop all the windows that are members of this group but not the window group itself, you can use the `DROP_WINDOW` procedure and provide the name of the window group to the call.

To drop a window group, you must have the `MANAGE SCHEDULER` privilege.

## ENABLE Procedure

This procedure enables a program, job, chain, window, database destination, external destination, file watcher, or group. When an object is enabled, its `enabled` attribute is set to `TRUE`. By default, jobs, chains, and programs are created disabled and database destinations, external destinations, file watchers, windows, and groups are created enabled.

If a job was disabled and you enable it, the Scheduler begins to automatically run the job according to its schedule. Enabling a disabled job also resets the `JOB_RUN_COUNT`, `FAILURE_COUNT` and `RETRY_COUNT` columns in the `*_SCHEDULER_JOBS` data dictionary views.

Validity checks are performed before enabling an object. If the check fails, the object is not enabled, and an appropriate error is returned. This procedure does not return an error if the object was already enabled.

### Syntax

```
DBMS_SCHEDULER.ENABLE (
    name           IN VARCHAR2,
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

### Parameters

**Table 128–57** *ENABLE Procedure Parameters*

Parameter	Description
<code>name</code>	<p>The name of the Scheduler object being enabled. Can be a comma-delimited list of names.</p> <p>If a job class name is specified, then all the jobs in the job class are enabled.</p> <p>If a group name is specified, then the group is enabled, but the enabled state of the group members is unaffected.</p>
<code>commit_semantics</code>	<p>The commit semantics. The following types are supported:</p> <ul style="list-style-type: none"> <li>▪ <code>STOP_ON_FIRST_ERROR</code> - The procedure returns on the first error and previous successful enable operations are committed to disk. This is the default.</li> <li>▪ <code>TRANSACTIONAL</code> - The procedure returns on the first error and everything that happened before that error is rolled back. This type is only supported when enabling a job or a list of jobs.</li> <li>▪ <code>ABSORB_ERRORS</code> - The procedure tries to absorb any errors and enable the rest of the jobs. It commits all the enable operations that were successful. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details. This type is only supported when enabling a job or a list of jobs.</li> </ul>

### Usage Notes

Window names must be preceded by `SYS`.

To run `ENABLE` for a window or group of type `WINDOW`, you must have the `MANAGE_SCHEDULER` privilege. For a job of type `EXECUTABLE` (or for a job that points to a program of type `EXECUTABLE`), the job owner must have the `CREATE_EXTERNAL_JOB` system privilege before the job can be enabled or run.

To enable a file watcher, the file watcher owner must have the `EXECUTE` privilege on the designated credential.

You can use `ENABLE` with any schema except the `SYS` schema.

## END\_DETACHED\_JOB\_RUN Procedure

This procedure ends a detached job run. A detached job points to a detached program, which is a program with the `detached` attribute set to `TRUE`. A detached job run does not end until this procedure or the [STOP\\_JOB Procedure](#) is called.

### Syntax

```
DBMS_SCHEDULER.END_DETACHED_JOB_RUN (
  job_name          IN VARCHAR2,
  error_number      IN PLS_INTEGER DEFAULT 0,
  additional_info   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–58** *END\_DETACHED\_JOB\_RUN Procedure Parameters*

Parameter	Description
<code>job_name</code>	The name of the job to end. Must be a detached job that is running.
<code>error_number</code>	If zero, then the job run is logged as succeeded. If -1013, then the job run is logged as stopped. If non-zero, then the job run is logged as failed with this error number.
<code>additional_info</code>	This text is stored in the <code>additional_info</code> column of the <code>*_scheduler_job_run_details</code> views for this job run.

### Usage Notes

This procedure requires that you either own the job or have `ALTER` privileges on it. You can also end any detached job run if you have the `CREATE ANY JOB` privilege.

**See Also:** *Oracle Database Administrator's Guide* for information about detached jobs.

## EVALUATE\_CALENDAR\_STRING Procedure

You can define repeat intervals of jobs, windows or schedules using the Scheduler calendaring syntax. This procedure evaluates the calendar expression and tells you the next execution date and time of a job or window. This is very useful for testing the correct definition of the calendar string without actually scheduling the job or window.

This procedure can also get multiple steps of the repeat interval by passing the `next_run_date` returned by one invocation as the `return_date_after` argument of the next invocation.

See the calendaring syntax described in ["Operational Notes"](#) on page 128-20.

### Syntax

```
DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING (
  calendar_string      IN  VARCHAR2,
  start_date           IN  TIMESTAMP WITH TIME ZONE,
  return_date_after    IN  TIMESTAMP WITH TIME ZONE,
  next_run_date        OUT TIMESTAMP WITH TIME ZONE);
```

### Parameters

**Table 128–59** *EVALUATE\_CALENDAR\_STRING Procedure Parameters*

Parameter	Description
<code>calendar_string</code>	The calendar expression to be evaluated. The string must be in the calendaring syntax described in <a href="#">"Operational Notes"</a> on page 128-20.
<code>start_date</code>	The date and time after which the repeat interval becomes valid. It can also be used to fill in specific items that are missing from the calendar string. Can optionally be NULL.
<code>return_date_after</code>	The <code>return_date_after</code> argument helps the Scheduler determine which one of all possible matches (all valid execution dates) to return from those determined by the <code>start_date</code> and the calendar string.  When a NULL value is passed for this argument, the Scheduler automatically fills in <code>sys_timestamp</code> as its value.
<code>next_run_date</code>	The first timestamp that matches the calendar string and start date that occur after the value passed in for the <code>return_date_after</code> argument.

### Examples

The following code fragment can be used to determine the next five dates a job will run given a specific calendar string.

```
SET SERVEROUTPUT ON;
ALTER SESSION set NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
Session altered.

DECLARE
  start_date          TIMESTAMP;
  return_date_after  TIMESTAMP;
  next_run_date       TIMESTAMP;
BEGIN
  start_date :=
    to_timestamp_tz('01-JAN-2003 10:00:00', 'DD-MON-YYYY HH24:MI:SS');
  return_date_after := start_date;
  FOR i IN 1..5 LOOP
```

```
DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING(  
    'FREQ=DAILY;BYHOUR=9;BYMINUTE=30;BYDAY=MON,TUE,WED,THU,FRI',  
    start_date, return_date_after, next_run_date);  
DBMS_OUTPUT.PUT_LINE('next_run_date: ' || next_run_date);  
return_date_after := next_run_date;  
END LOOP;  
END;  
/  
  
next_run_date: 02-JAN-03 09.30.00.000000 AM  
next_run_date: 03-JAN-03 09.30.00.000000 AM  
next_run_date: 06-JAN-03 09.30.00.000000 AM  
next_run_date: 07-JAN-03 09.30.00.000000 AM  
next_run_date: 08-JAN-03 09.30.00.000000 AM  
  
PL/SQL procedure successfully completed.
```

## Usage Notes

No specific Scheduler privileges are required.

## EVALUATE\_RUNNING\_CHAIN Procedure

This procedure forces reevaluation of the rules of a running chain to trigger any rules for which the conditions have been satisfied. The job passed as an argument must point to a chain and must be running. If the job is not running, an error is thrown. (RUN\_JOB can be used to start the job.)

If any of the steps of the chain are themselves running chains, another EVALUATE\_RUNNING\_CHAIN is performed on each of the nested running chains.

### Syntax

```
DBMS_SCHEDULER.EVALUATE_RUNNING_CHAIN (
    job_name          IN VARCHAR2);
```

### Parameters

**Table 128–60** EVALUATE\_RUNNING\_CHAIN Procedure Parameter

Parameter	Description
job_name	The name of the running job (pointing to a chain) to reevaluate the rules for

### Usage Notes

Running EVALUATE\_RUNNING\_CHAIN on a job requires alter privileges on the job (either as the owner, or as a user with ALTER privileges on the job or the CREATE ANY JOB system privilege).

---

**Note:** The Scheduler automatically evaluates a chain:

- At the start of the chain job
- When a chain step completes
- When an event occurs that is associated with an event step in the chain

For most chains, this is sufficient. EVALUATE\_RUNNING\_CHAIN should be used only under the following circumstances:

- After manual intervention of a running chain with the ALTER\_RUNNING\_CHAIN procedure
- When chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler.

In these cases, EVALUATE\_RUNNING\_CHAIN may not be needed if you set the evaluation\_interval attribute when you created the chain.

---



## GENERATE\_JOB\_NAME Function

This function returns a unique name for a job. The name will be of the form {prefix}N where N is a number from a sequence. If no prefix is specified, the generated name will, by default, be JOB\$\_1, JOB\$\_2, JOB\$\_3, and so on. If 'SCOTT' is specified as the prefix, the name will be SCOTT1, SCOTT2, and so on.

### Syntax

```
DBMS_SCHEDULER.GENERATE_JOB_NAME (
    prefix          IN VARCHAR2 DEFAULT 'JOB$_') RETURN VARCHAR2;
```

### Parameters

**Table 128–61** *GENERATE\_JOB\_NAME Function Parameter*

Parameter	Description
prefix	The prefix to use when generating the job name

### Usage Notes

If the prefix is explicitly set to NULL, the name is just the sequence number. In order to successfully use such numeric names, they must be surrounded by double quotes throughout the DBMS\_SCHEDULER calls. A prefix cannot be longer than 18 characters and cannot end with a digit.

Note that, even though the GENERATE\_JOB\_NAME function never returns the same job name twice, there is a small chance that the returned name matches an already existing database object.

No specific Scheduler privileges are required to use this function.

## GET\_AGENT\_INFO Function

This function can return job information specific to an agent, such as how many are running and so on, depending on the attribute selected.

### Syntax

```
DBMS_SCHEDULER.GET_AGENT_INFO (
  agent_name      IN VARCHAR2,
  attribute       IN VARCHAR2) RETURN VARCHAR2;
```

### Parameters

**Table 128–62** GET\_AGENT\_INFO Function Parameter

Parameter	Description
agent_name	The name of an external destination where the agent is running
attribute	Possible Attributes values <ul style="list-style-type: none"> <li>■ <b>VERSION:</b> Returns the agent version number. Requires the <code>CREATE JOB</code> system privilege.</li> <li>■ <b>UPTIME:</b> Returns the time the agent has been up and running. Requires the <code>CREATE JOB</code> system privilege.</li> <li>■ <b>NUMBER_OF_RUNNING_JOBS:</b> Returns the number of jobs that the agent is currently running. Requires the <code>CREATE JOB</code> system privilege.</li> <li>■ <b>TOTAL_JOBS_RUN:</b> Returns the number of jobs run by the agent since it was started. Requires the <code>CREATE JOB</code> system privilege.</li> <li>■ <b>RUNNING_JOBS:</b> Returns a comma-separated list of the names of the jobs running currently. Requires the <code>MANAGE SCHEDULER</code> system privilege.</li> <li>■ <b>ALL:</b> Returns all the information the previous options return. It requires the <code>MANAGE SCHEDULER</code> system privilege.</li> </ul>

### Usage Notes

This function returns the same information as the `schagent` utility status option.

## GET\_AGENT\_VERSION Function

This function returns the version string of a Scheduler agent that is registered with the database and is currently running. `GET_AGENT_VERSION` throws an error if the agent is not registered with the database or if the agent is not currently running.

### Syntax

```
DBMS_SCHEDULER.GET_AGENT_VERSION (
  agent_host      IN VARCHAR2) RETURN VARCHAR2;
```

### Parameters

**Table 128–63** *GET\_AGENT\_VERSION Function Parameter*

Parameter	Description
agent_host	Either the hostname and port on which the agent is running in the form <code>hostname:port</code> or the name of the agent as shown in the <code>destination_name</code> column of the <code>ALL_SCHEDULER_EXTERNAL_DESTS</code> view which lists all Scheduler agents registered with the database.

### Usage Notes

This function requires the `CREATE EXTERNAL JOB` system privilege.

## GET\_ATTRIBUTE Procedure

This procedure retrieves the value of an attribute of a Scheduler object. It is overloaded to retrieve values of various types.

### Syntax

```
DBMS_SCHEDULER.GET_ATTRIBUTE (
  name          IN VARCHAR2,
  attribute     IN VARCHAR2,
  value        OUT {VARCHAR2|PLS_INTEGER|BOOLEAN|DATE|TIMESTAMP|
                  TIMESTAMP WITH TIME ZONE|TIMESTAMP WITH LOCAL TIME ZONE|
                  INTERVAL DAY TO SECOND});
```

```
DBMS_SCHEDULER.GET_ATTRIBUTE (
  name          IN VARCHAR2,
  attribute     IN VARCHAR2,
  value        OUT VARCHAR2,
  value2       OUT VARCHAR2);
```

### Parameters

**Table 128–64** GET\_ATTRIBUTE Procedure Parameters

Parameter	Description
name	The name of the object
attribute	The attribute being retrieved. See the <a href="#">SET_ATTRIBUTE Procedure</a> on page 128-130 for tables of attribute values.
value	The existing value of the attribute
value2	The value2 argument is for an optional second value. Most attributes have only one value associated with them, but some can have two.

### Usage Notes

To run GET\_ATTRIBUTE for a job class, you must have the MANAGE\_SCHEDULER privilege or have EXECUTE privileges on the class. For a schedule, window, or group, no privileges are necessary. Otherwise, you must be the owner of the object or have ALTER or EXECUTE privileges on that object or have the CREATE ANY JOB privilege.

See the [SET\\_ATTRIBUTE Procedure](#) on page 128-130 for tables of attribute values that you can retrieve for the various Scheduler object types.

## GET\_FILE Procedure

This procedure retrieves a file from the operating system file system of a specified host. The file is copied to a destination, or its contents are returned in a procedure output parameter.

You can also use this procedure to retrieve the standard output or error text for a run of an external job that has an associated credential.

This procedure differs from the equivalent `UTL_FILE` procedure in that it uses a credential and can retrieve files from remote hosts that have only a Scheduler agent (and not an Oracle database) installed.

### Syntax

```
DBMS_SCHEDULER.GET_FILE (
    source_file           IN VARCHAR2,
    source_host          IN VARCHAR2,
    credential_name      IN VARCHAR2,
    file_contents        IN OUT NOCOPY {BLOB|CLOB});

DBMS_SCHEDULER.GET_FILE (
    source_file           IN VARCHAR2,
    source_host          IN VARCHAR2,
    credential_name      IN VARCHAR2,
    destination_file_name IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    destination_permissions IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–65** *GET\_FILE Procedure Parameters*

Parameter	Description
<code>source_file</code>	<p>Fully qualified path name of the file to retrieve from the operating system. The file name is case-sensitive and is not converted to uppercase. If the file name starts with a question mark ('?'), the question mark is replaced by the path to the Oracle home if getting a file from the local host, or to the Scheduler agent home if getting a file from a remote host.</p> <p>If the format of this parameter is <code>external_log_id_stdout</code>, then the stdout from the designated external job run is returned.</p> <p>If the format of this parameter is <code>external_log_id_stderr</code>, the error text from the designated external job run is returned.</p> <p>You obtain the value of <code>external_log_id</code> from the <code>ADDITIONAL_INFO</code> column of the <code>*_SCHEDULER_JOB_RUN_DETAILS</code> views. This column contains a set of name/value pairs in an indeterminate order, so you must parse this column for the <code>external_log_id</code> name/value pair, and then append either <code>"_stdout"</code> or <code>"_stderr"</code> to its value.</p> <p>The external job must have an associated credential. The <code>credential_name</code> parameter of <code>GET_FILE</code> must name the same credential that is used by the job, and the <code>source_host</code> parameter must be the same as the <code>destination</code> attribute of the job.</p>

**Table 128–65 (Cont.) GET\_FILE Procedure Parameters**

Parameter	Description
source_host	<p>If the file is to be retrieved from a remote host, then this parameter must be a valid an external destination name. (An external destination is created when you register a remote Scheduler agent with the database. You can view external destination names in the views *_SCHEDULER_EXTERNAL_DESTS.)</p> <p>If source_host is NULL or set to 'localhost', then the file is retrieved from the file system of the local host. To determine the port number of a Scheduler agent, view the schagent.conf file, which is located in the Scheduler agent home directory on the remote host.</p>
credential_name	The name of the credential to use for accessing the file system.
file_contents	The variable from which the file contents is read.
destination_file_name	The file to which the file contents is written.
destination_directory_object	The directory object that specifies the path to the destination file, when destination_file_name is used. The caller must have the necessary privileges on the directory object.
destination_permissions	Reserved for future use

## Usage Notes

The caller must have the CREATE EXTERNAL JOB system privilege and have EXECUTE privileges on the credential.

## GET\_SCHEDULER\_ATTRIBUTE Procedure

This procedure retrieves the value of a Scheduler attribute.

### Syntax

```
DBMS_SCHEDULER.GET_SCHEDULER_ATTRIBUTE (
  attribute    IN VARCHAR2,
  value       OUT VARCHAR2);
```

### Parameters

**Table 128–66** GET\_SCHEDULER\_ATTRIBUTE Procedure Parameters

Parameter	Description
attribute	The name of the attribute
value	The existing value of the attribute

### Usage Notes

To run GET\_SCHEDULER\_ATTRIBUTE, you must have the MANAGE\_SCHEDULER privilege.

[Table 128–67](#) lists the Scheduler attributes that you can retrieve. For more detail on these attributes, see [Table 128–97](#) on page 128-151 and the section "Configuring the Scheduler" in *Oracle Database Administrator's Guide*.

**Table 128–67** Scheduler Attributes Retrievable with GET\_SCHEDULER\_ATTRIBUTE

Scheduler Attribute	Description
current_open_window	Name of the currently open window
default_timezone	Default time zone used by the Scheduler for repeat intervals and windows
email_sender	The default e-mail address of the sender for job state e-mail notifications
email_server	The SMTP server address that the Scheduler uses to send e-mail notifications for job state events. E-mail notifications cannot be sent if this attribute is NULL.
event_expiry_time	Time in seconds before an event generated by the Scheduler and enqueued onto the Scheduler event queue expires. May be NULL.
log_history	Retention period in days for job and window logs. The range of valid values is 0 through 1000000.
max_job_slave_processes	This Scheduler attribute is not used.

## OPEN\_WINDOW Procedure

This procedure opens a window independent of its schedule. This window opens and the resource plan associated with it takes effect immediately for the duration specified or for the normal duration of the window if no duration is given. Only an enabled window can be manually opened.

### Syntax

```
DBMS_SCHEDULER.OPEN_WINDOW (
    window_name          IN VARCHAR2,
    duration              IN INTERVAL DAY TO SECOND,
    force                 IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 128–68 OPEN\_WINDOW Procedure Parameters**

Parameter	Description
<code>window_name</code>	The name of the window
<code>duration</code>	The duration of the window. It is of type interval day to second. If it is <code>NULL</code> , then the window opens for the regular duration as specified in the window metadata.
<code>force</code>	<p>If <code>force</code> is set to <code>FALSE</code>, then opening an already open window generates an error.</p> <p>If <code>force</code> is set to <code>TRUE</code>:</p> <p>You can open a window that is already open. The window stays open for the duration specified in the call, from the time the <code>OPEN_WINDOW</code> command was issued.</p> <p>For example: <code>window1</code> was created with a duration of four hours. It has now been open for two hours. If, at this point, you reopen <code>window1</code> using the <code>OPEN_WINDOW</code> call and do not specify a duration, then <code>window1</code> stays open for four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes.</p> <p>The Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority.</p>

### Usage Notes

Opening a window manually has no impact on regular scheduled runs of the window. The next open time of the window is not updated and is determined by the regular scheduled opening.

When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

If there are jobs running when the window opens, the resources allocated to them might change if there is a switch in resource plan.

If a window fails to switch resource plans because the designated resource plan no longer exists or because resource plan switching by windows is disabled (for example, by using the `ALTER SYSTEM` statement with the `force` option), the failure to switch resource plans is recorded in the window log.



Opening a window requires the `MANAGE SCHEDULER` privilege.

## PURGE\_LOG Procedure

By default, the Scheduler automatically purges all rows in the job log and window log that are older than 30 days. The `PURGE_LOG` procedure is used to purge additional rows from the job and window log.

Rows in the job log table pertaining to the steps of a chain are purged only when the entry for the main chain job is purged (either manually or automatically).

### Syntax

```
DBMS_SCHEDULER.PURGE_LOG (
    log_history          IN PLS_INTEGER  DEFAULT 0,
    which_log           IN VARCHAR2     DEFAULT 'JOB_AND_WINDOW_LOG',
    job_name            IN VARCHAR2     DEFAULT NULL);
```

### Parameters

**Table 128–69** *PURGE\_LOG Procedure Parameters*

Parameter	Description
<code>log_history</code>	This specifies how much history (in days) to keep. The valid range is 0 - 1000000. If set to 0, no history is kept.
<code>which_log</code>	This specifies the log type. Valid values are: <code>job_log</code> , <code>window_log</code> , and <code>job_and_window_log</code> .
<code>job_name</code>	This specifies which job-specific entries must be purged from the job log. This can be a comma-delimited list of job names and job classes. Whenever <code>job_name</code> has a value other than <code>NULL</code> , the <code>which_log</code> argument implicitly includes the job log.

### Usage Notes

This procedure requires the `MANAGE SCHEDULER` privilege.

### Examples

The following completely purges all rows from both the job log and the window log:

```
DBMS_SCHEDULER.PURGE_LOG();
```

The following purges all rows from the window log that are older than 5 days:

```
DBMS_SCHEDULER.PURGE_LOG(5, 'window_log');
```

The following purges all rows from the window log that are older than 1 day and all rows from the job log that are related to jobs in `jobclass1` and older than 1 day:

```
DBMS_SCHEDULER.PURGE_LOG(1, 'job_and_window_log', 'sys.jobclass1');
```

## PUT\_FILE Procedure

This procedure saves a file to the operating system file system of a specified remote host or of the local computer. It differs from the equivalent UTL\_FILE procedure in that it uses a credential and can save files to a remote host that has only a Scheduler agent (and not an Oracle Database) installed.

### Syntax

```
DBMS_SCHEDULER.PUT_FILE (
  destination_file      IN VARCHAR2,
  destination_host      IN VARCHAR2,
  credential_name       IN VARCHAR2,
  file_contents         IN {BLOB|CLOB},
  destination_permissions IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_SCHEDULER.PUT_FILE (
  destination_file      IN VARCHAR2,
  destination_host      IN VARCHAR2,
  credential_name       IN VARCHAR2,
  source_file_name      IN VARCHAR2,
  source_directory_object IN VARCHAR2,
  destination_permissions IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–70 PUT\_FILE Procedure Parameters**

Parameter	Description
destination_file	Fully qualified path name of the file to save to the operating system file system. The file name is case-sensitive. If the file name starts with a question mark ('?'), the question mark is replaced by the path to the Oracle home if saving to the local host, or to the Scheduler agent home if saving to a remote host.
destination_host	If NULL or set to 'localhost', the file is saved to the file system of the local computer.  To save to a remote host, this parameter must be a valid external destination name. (An external destination is created when you register a remote Scheduler agent with the database. You can view external destination names in the views *_SCHEDULER_EXTERNAL_DESTS.)
credential_name	The name of the credential to use for accessing the destination file system.
file_contents	The variable from which the file contents is read.
source_file_name	The file from which the file contents is written
source_directory_object	The directory object that specifies the path to the source file, when source_file_name is used. The caller must have the necessary privileges on the directory object.
destination_permissions	Reserved for future use

### Usage Notes

The caller must have the CREATE EXTERNAL JOB system privilege and have EXECUTE privileges on the credential.

## REMOVE\_EVENT\_QUEUE\_SUBSCRIBER Procedure

This procedure unsubscribes a user from the Scheduler event queue SYS.SCHEDULER\$\_EVENT\_QUEUE.

### Syntax

```
DBMS_SCHEDULER.REMOVE_EVENT_QUEUE_SUBSCRIBER (  
    subscriber_name          IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–71 REMOVE\_EVENT\_QUEUE\_SUBSCRIBER Procedure Parameters**

Parameter	Description
subscriber_name	Name of the Oracle Streams Advanced Queuing (AQ) agent to remove the subscription from. If NULL, the user name of the calling user is used.

### Usage Notes

After the agent is unsubscribed, it is deleted. If the agent does not exist or is not currently subscribed to the Scheduler event queue, an error is raised.

## REMOVE\_GROUP\_MEMBER Procedure

This procedure removes one or more members from an existing group.

### Syntax

```
DBMS_SCHEDULER.REMOVE_GROUP_MEMBER (
  group_name      IN VARCHAR2,
  member         IN VARCHAR2);
```

### Parameters

**Table 128-72 REMOVE\_GROUP\_MEMBER Procedure Parameters**

Parameter	Description
group_name	The name of the group.
member_name	<p>The name of the member to remove from group. Comma-separated list of members to remove. An error is returned if any of the members is not part of the group.</p> <p>A group of the same type can be named as a member. The Scheduler immediately expands the included group name into its list of members.</p> <p>If the member is a destination, any job instances that run on this destination are removed from the *_SCHEDULER_JOB_DESTS views.</p>

### Usage Notes

The following users may remove members from a group:

- The group owner
- A user that has been granted the ALTER object privilege on the group
- A user with the CREATE ANY JOB system privilege

You must have the MANAGE SCHEDULER privilege to remove a member from a group of type WINDOW.

**See Also:** ["CREATE\\_GROUP Procedure"](#) on page 128-55

## REMOVE\_JOB\_EMAIL\_NOTIFICATION Procedure

This procedure removes e-mail notifications for a job. You can remove all e-mail notifications or remove notifications only for specified recipients or specified events.

### Syntax

```
DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION (
    job_name          IN VARCHAR2,
    recipients        IN VARCHAR2 DEFAULT NULL,
    events            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128-73** ADD\_JOB\_EMAIL\_NOTIFICATION Procedure Parameters

Parameter	Description
job_name	Name of the job to remove e-mail notifications for. Cannot be NULL.
recipients	E-mail address to remove e-mail notification for. Comma-separated list of e-mail addresses.
events	Job state event to remove e-mail notification for. Comma-separated list of job state events.

### Usage Notes

When you specify multiple recipients and multiple events, the notification for each specified event is removed for each specified recipient. The procedure ignores any recipients or events that are specified but that were not previously added.

If `recipients` is NULL, e-mail notifications for the specified events are removed for all existing recipients. If `events` is NULL, notifications for all events are removed for the specified recipients. If both `recipients` and `events` are NULL, all e-mail notifications are removed for the job.

For example, if `recipients` is 'jsmith@example.com,rjones@example.com' and `events` is 'JOB\_FAILED,JOB\_BROKEN', then notifications for both the JOB\_FAILED and JOB\_BROKEN events are removed for both jsmith and rjones. If `recipients` is NULL, then notifications for both the JOB\_FAILED and JOB\_BROKEN events are removed for jsmith, rjones, and any other previously defined recipients for these events.

To call this procedure, you must be the job owner or a user with the CREATE ANY JOB system privilege or ALTER object privilege on the job.

**See Also:** ["ADD\\_JOB\\_EMAIL\\_NOTIFICATION Procedure"](#) on page 128-36

## REMOVE\_WINDOW\_GROUP\_MEMBER Procedure

**\*\*\* Deprecated in Oracle Database 11g Release 2 (11.2). Use REMOVE\_GROUP\_MEMBER instead.**

This procedure removes one or more windows from an existing window group.

### Syntax

```
DBMS_SCHEDULER.REMOVE_WINDOW_GROUP_MEMBER (
    group_name          IN VARCHAR2,
    window_list        IN VARCHAR2);
```

### Parameters

**Table 128–74 REMOVE\_WINDOW\_GROUP\_MEMBER Procedure Parameters**

Parameter	Description
group_name	The name of the window group.
window_list	The name of the window or windows.

### Usage Notes

If any of the windows specified is invalid, does not exist, or is not a member of the given group, the call fails. Removing a window from a group requires the `MANAGE SCHEDULER` privilege.

Dropping an open window from a window group has no impact on any running jobs that has the window as its schedule because the jobs only stop when a window closes.

## RESET\_JOB\_ARGUMENT\_VALUE Procedure

This procedure resets (clears) the value previously set to an argument for a job.

RESET\_JOB\_ARGUMENT\_VALUE is overloaded.

### Syntax

Clears a previously set job argument value by argument position:

```
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (
  job_name          IN VARCHAR2,
  argument_position IN PLS_INTEGER);
```

Clears a previously set job argument value by argument name:

```
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (
  job_name          IN VARCHAR2,
  argument_name     IN VARCHAR2);
```

### Parameters

**Table 128–75** RESET\_JOB\_ARGUMENT\_VALUE Procedure Parameters

Parameter	Description
job_name	The name of the job being altered
argument_position	The position of the program argument being reset
argument_name	The name of the program argument being reset

### Usage Notes

If the corresponding program argument has no default value, the job is disabled. Resetting a program argument of a job belonging to another user requires ALTER privileges on that job. Arguments can be specified by position or by name.

RESET\_JOB\_ARGUMENT\_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also reset a job argument value if you have the CREATE ANY JOB privilege.

RESET\_JOB\_ARGUMENT\_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.



## RUN\_CHAIN Procedure

This procedure immediately runs a chain or part of a chain by creating a run-once job with the job name given. If no `job_name` is given, one is generated of the form `RUN_CHAIN$_chainnameN`, where `chainname` is the first 8 characters of the chain name and `N` is an integer.

If a list of start steps is given, only those steps are started when the chain begins running. Steps not in the list that would normally have started are skipped and paused (so that they or the steps after them do not run). If `start_steps` is `NULL`, then the chain starts normally—that is, it performs an initial evaluation to see which steps to start running).

If a list of initial step states is given, the newly created chain job sets every listed step to the state specified for that step before evaluating the chain rules to see which steps to start. (Steps in the list are not started.)

### Syntax

Runs a chain, with a list of start steps.

```
DBMS_SCHEDULER.RUN_CHAIN (
    chain_name          IN VARCHAR2,
    start_steps         IN VARCHAR2,
    job_name            IN VARCHAR2 DEFAULT NULL);
```

Runs a chain, with a list of initial step states.

```
DBMS_SCHEDULER.RUN_CHAIN (
    chain_name          IN VARCHAR2,
    step_state_list    IN SYS.SCHEDULER$_STEP_TYPE_LIST,
    job_name            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–76** *RUN\_CHAIN Procedure Parameters*

Parameter	Description				
<code>chain_name</code>	The name of the chain to run				
<code>job_name</code>	The name of the job to create to run the chain				
<code>start_steps</code>	Comma-separated list of the steps to start when the chain starts running				
<code>step_state_list</code>	List of chain steps with an initial state ( <code>SUCCEEDED</code> or <code>FAILED</code> ) to set for each.  Set the attributes of <code>sys.scheduler\$_step_type</code> as follows: <table border="0" style="margin-left: 20px;"> <tr> <td><code>step_name</code></td> <td>The name of the step</td> </tr> <tr> <td><code>step_type</code></td> <td>'SUCCEEDED' or 'FAILED <i>error_number</i>'</td> </tr> </table> where <i>error_number</i> is a positive or negative integer.	<code>step_name</code>	The name of the step	<code>step_type</code>	'SUCCEEDED' or 'FAILED <i>error_number</i> '
<code>step_name</code>	The name of the step				
<code>step_type</code>	'SUCCEEDED' or 'FAILED <i>error_number</i> '				

### Usage Notes

Running a chain requires `CREATE JOB` if the job is being created in the user's schema, or `CREATE ANY JOB` otherwise. In addition, the owner of the job being created needs execute privileges on the chain (as the owner of the chain, or as a user with the `EXECUTE` privilege on the chain or the `EXECUTE ANY PROGRAM` system privilege).

## Examples

The following example illustrates how to start a chain in the middle by providing the initial state of some chain steps.

```
declare
  initial_step_states sys.scheduler$_step_type_list;
begin
  initial_step_states := sys.scheduler$_step_type_list(
    sys.scheduler$_step_type('step1', 'SUCCEEDED'),
    sys.scheduler$_step_type('step2', 'FAILED 27486'),
    sys.scheduler$_step_type('step3', 'SUCCEEDED'),
    sys.scheduler$_step_type('step5', 'SUCCEEDED'));
  dbms_scheduler.run_chain('my_chain', initial_step_states);
end;
/
```

## RUN\_JOB Procedure

This procedure runs a job immediately.

If a job is enabled, the Scheduler runs it automatically. It is not necessary to call RUN\_JOB to run a job according to its schedule. Use RUN\_JOB to run a job outside of its normal schedule.

### Syntax

```
DBMS_SCHEDULER.RUN_JOB (
    job_name          IN VARCHAR2,
    use_current_session IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 128–77** RUN\_JOB Procedure Parameters

Parameter	Description
job_name	<p>A job name or a comma-separated list of entries, where each is the name of an existing job, optionally preceded by a schema name and dot separator.</p> <p>If you specify a multiple-destination job, the job runs on all destinations. In this case, the use_current_session argument must be FALSE.</p>
use_current_session	<p>This specifies whether or not the job run should occur in the same session that the procedure was invoked from.</p> <p>When use_current_session is set to TRUE:</p> <ul style="list-style-type: none"> <li>■ The job runs as the user who called RUN_JOB, or in the case of a local external job with a credential, the user named in the credential.</li> <li>■ You can test a job and see any possible errors on the command line.</li> <li>■ run_count, last_start_date, last_run_duration, and failure_count are not updated.</li> <li>■ RUN_JOB can be run in parallel with a regularly scheduled job run.</li> </ul> <p>When use_current_session is set to FALSE:</p> <ul style="list-style-type: none"> <li>■ The job runs as the user who is the job owner.</li> <li>■ You need to check the job log to find error information.</li> <li>■ run_count, last_start_date, last_run_duration, and failure_count are updated.</li> <li>■ RUN_JOB fails if a regularly scheduled job is running.</li> </ul> <p>For jobs that have a specified destination or destination group, or point to chains or programs with the detached attribute set to TRUE, use_current_session must be FALSE</p>

### Usage Notes

The job does not have to be enabled. If the job is disabled, the following validity checks are performed before running it:

- The job points to a valid job class.
- The job owner has EXECUTE privileges on the job class.

- If a program or chain is referenced, the program/chain exists.
- If a program or chain is referenced, the job owner has privileges to execute the program/chain.
- All argument values have been set (or have defaults).
- The job owner has the `CREATE EXTERNAL JOB` privilege if this is an external job.

A `TRUE` value for `use_current_session` is not permitted for the following types of jobs:

- Jobs that specify a destination or destination group in the `destination_name` attribute
- Jobs that point to chains (chain jobs)
- Jobs that make use of detached programs (detached jobs).

When `use_current_session` is `TRUE`, the call to `RUN_JOB` blocks until the job completes. Any errors that occur during the execution of the job are returned as errors to the `RUN_JOB` procedure. When `use_current_session` is `FALSE`, `RUN_JOB` returns immediately and the job is picked up by the job coordinator and passed on to a job slave for execution. The Scheduler views and logs must be queried for the outcome of the job.

Multiple user sessions can use `RUN_JOB` in their sessions simultaneously when `use_current_session` is set to `TRUE`.

`RUN_JOB` requires that you own the job or have `ALTER` privileges on that job. You can also run a job if you have the `CREATE ANY JOB` privilege.

## Example

The following is an example of using `RUN_JOB`.

```
BEGIN
  DBMS_SCHEDULER.RUN_JOB (
    JOB_NAME          => 'EODJOB, DSS.ETLJOB',
    USE_CURRENT_SESSION => FALSE);
END;
```

## SET\_AGENT\_REGISTRATION\_PASS Procedure

This procedure sets the agent registration password for a database. A Scheduler agent must register with the database before the database can submit jobs to the agent. The agent must provide this password when registering.

### Syntax

```
DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS (
  registration_password  IN VARCHAR2,
  expiration_date       IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  max_uses              IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 128–78 SET\_AGENT\_REGISTRATION\_PASS Procedure Parameters**

Parameter	Description
registration_password	This is the password that remote agents must specify in order to successfully register with the database. If this is NULL, then no agents will be able to register with the database.
expiration_date	If this is set to a non-NULL value, then the registration_password is not valid after this date. After this date, no agents can register with the database. This cannot be set to a date in the past.
max_uses	This is the maximum number of successful registrations that can be performed with this password. After the number of successful registrations has been performed with this password, then no agents can register with the database. This cannot be set to 0 or a negative value. If this is set to NULL, then there will be no limit on the number of successful registrations.

### Usage Notes

To prevent abuse, this password can be set to expire after a given date or a maximum number of successful registrations. This procedure will overwrite any password already set. This requires the `MANAGE_SCHEDULER` system privilege.

By default, `max_uses` is set to NULL, which means that there is no limit to the number of successful registrations.

Oracle recommends that an agent registration password be reset after every agent registration or every known set of agent registrations. Furthermore, Oracle recommends that this password be set to NULL if no new agents are being registered.

## SET\_ATTRIBUTE Procedure

This procedure modifies an attribute of a Scheduler object. It is overloaded to accept values of various types. To set an attribute to NULL, use the `SET_ATTRIBUTE_NULL` procedure. The attributes that can be set depend on the object being altered. All object attributes can be changed, except the object name.

### Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE (
    name          IN VARCHAR2,
    attribute     IN VARCHAR2,
    value        IN {BOOLEAN|DATE|TIMESTAMP|
                    TIMESTAMP WITH TIME ZONE|TIMESTAMP WITH LOCAL TIME ZONE|
                    INTERVAL DAY TO SECOND});
```

```
DBMS_SCHEDULER.SET_ATTRIBUTE (
    name          IN VARCHAR2,
    attribute     IN VARCHAR2,
    value        IN VARCHAR2,
    value2       IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 128–79 SET\_ATTRIBUTE Procedure Parameters**

Parameter	Description
name	The name of the object
attribute	See <a href="#">Table 128–81</a> through <a href="#">Table 128–91</a> .
value	The new value being set for the attribute. This cannot be NULL. To set an attribute value to NULL, use the <code>SET_ATTRIBUTE_NULL</code> procedure.
value2	The <code>value2</code> argument is for an optional second value. Most attributes have only one value associated with them, but some can have two.

[Table 128–80](#) is a directory of Scheduler object types and tables of attributes for the object types.

These object types can be viewed with Scheduler Data Dictionary Views, listed in *Oracle Database Administrator's Guide*.

**Table 128–80 Attribute Tables for Scheduler Object Types**

Scheduler Object Type	Table of Attributes
Job	<a href="#">Table 128–81</a> on page 128-132
Program	<a href="#">Table 128–83</a> on page 128-139
Schedule	<a href="#">Table 128–84</a> on page 128-139
File Watcher	<a href="#">Table 128–85</a> on page 128-139
Job Class	<a href="#">Table 128–86</a> on page 128-140
Window	<a href="#">Table 128–87</a> on page 128-141
Chain	<a href="#">Table 128–88</a> on page 128-141
Database Destination	<a href="#">Table 128–89</a> on page 128-142

**Table 128–80 (Cont.) Attribute Tables for Scheduler Object Types**

Scheduler Object Type	Table of Attributes
External Destination	<a href="#">Table 128–90</a> on page 128-143
Group	<a href="#">Table 128–91</a> on page 128-143
Credential	<a href="#">Table 128–92</a> on page 128-143

## Usage Notes

If an object is altered and it was in the enabled state, the Scheduler first disables it, then makes the change and reenables it. If any errors are encountered during the enable process, the object is not reenabled and an error is generated.

If an object is altered and it was in the disabled state, it remains disabled after it is altered.

To run `SET_ATTRIBUTE` for a window, a group of type `WINDOW`, or job class, you must have the `MANAGE_SCHEDULER` privilege. Otherwise, you must be the owner of the object being altered or have `ALTER` privileges on that object or have the `CREATE ANY JOB` privilege.

### Job

If there is a running instance of the job when the `SET_ATTRIBUTE` call is made, it is not affected by the call. The change is only affects future runs of the job.

If any of the schedule attributes of a job are altered while the job is running, the time of the next job run is scheduled using the new schedule attributes. Schedule attributes of a job include `schedule_name`, `start_date`, `end_date`, and `repeat_interval`.

If any of the program attributes of a job are altered while the job is running, the new program attributes take effect the next time the job runs. Program attributes of a job include `program_name`, `job_action`, `job_type`, and `number_of_arguments`.

If any job argument values are altered while the job is running, the new values take effect the next time the job runs.

Granting the `ALTER` privilege on a job lets a user alter all attributes of that job except its program attributes (`program_name`, `job_type`, `job_action`, `program_action`, and `number_of_arguments`) and does not allow a user to use a PL/SQL expression to specify the schedule for a job.

Oracle recommends that you not alter a job that was automatically created for you by the database. Jobs that were created by the database have the column `SYSTEM` set to `TRUE` in job views.

### Program

If any currently running jobs use the program that was altered, they continue to run with the program definition prior to the alter. The job runs with the new program definition the next time the job executes.

### Schedule

If a schedule is altered, the change does not affect running jobs and open windows that use this schedule. The change only goes into effect the next time the jobs runs or the window opens.

**File Watcher**

If a file watcher is altered, any currently running event-based jobs started by the file arrival event are not affected. On the local system, the new file watcher attributes take effect the next time that the file watcher checks for the arrival of the file (every ten minutes by default). On remote systems, there may be an additional delay before the new file watcher attributes take effect.

**Job Class**

With the exception of the default job class, all job classes can be altered. To alter a job class, you must have the `MANAGE_SCHEDULER` privilege.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet.

**Window**

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

If there is no current resource plan, when a window with a designated resource plan opens, the Resource Manager activates with that plan.

**Job Attribute Values**

Table 128–81 lists attribute values for jobs.

**Note:** See the `CREATE_JOB` procedure and the `CREATE_JOBS` procedure for more complete descriptions of the attributes in this table.

**Table 128–81 Job Attribute Values**

Name	Description
<code>allow_runs_in_restricted_mode</code>	If <code>TRUE</code> , the job is permitted to run when the database is in restricted mode, provided that the job owner is permitted to log in during this mode. <code>FALSE</code> by default.
<code>auto_drop</code>	This attribute, if <code>TRUE</code> , causes a job to be automatically dropped after it completes or is automatically disabled. A job is considered completed if: <ul style="list-style-type: none"> <li>■ Its end date (or the end date of the schedule) has passed.</li> <li>■ It has run <code>max_runs</code> number of times. <code>max_runs</code> must be set with <code>SET_ATTRIBUTE</code>.</li> <li>■ It is not a repeating job and has run once.</li> </ul> A job is automatically disabled when it has failed <code>max_failures</code> times. <code>max_failures</code> is also set with <code>SET_ATTRIBUTE</code> . If this attribute is set to <code>FALSE</code> , the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the <code>DROP_JOB</code> procedure. By default, jobs are created with <code>auto_drop</code> set to <code>TRUE</code> .
<code>comments</code>	An optional comment.



**Table 128–81 (Cont.) Job Attribute Values**

Name	Description
credential_ name	<p>This attribute specifies the name of the Scheduler credential object (credential) to use for a remote database job, a remote external job, a local external job, or an event-based job that processes a file arrival event. For local external jobs only, if this attribute is NULL (the default), then a preferred (default) credential is selected. See <i>Oracle Database Administrator's Guide</i> for information about preferred credentials for local external jobs.</p>
database_role	<p>This attribute applies when the database participates in an Oracle Data Guard environment. If this attribute is set to 'PRIMARY', the job runs only when the database is in the role of the primary database. If set to 'LOGICAL STANDBY', the job runs only when the database is in the role of a logical standby. The default is 'PRIMARY' when the database is the primary database, and 'LOGICAL STANDBY' when the database is a logical standby.</p>
destination	<p>Note: If you want a job to run for all database roles on a particular host, you must create two copies of the job on that host: one with a database_role of 'PRIMARY', and the other with a database_role of 'LOGICAL STANDBY'.</p> <p><b>*** Deprecated in Oracle Database 11g Release 2 (11.2). Use <code>destination_name</code> instead.</b></p> <p>This attribute specifies a host on which to run a remote external job. It must be set to the host name or IP address of the destination host. It can optionally be followed by a port number, in the following format:</p> <p><i>hostname:port</i></p> <p>This attribute is set to NULL by default.</p>
destination_ name	<p>The database destination or external destination for the job. Use for remote database jobs and remote external jobs only. For jobs running on the local database or for local external jobs (executables), must be NULL.</p> <p>See <a href="#">Table 128–27</a> on page 128-59 for details about this attribute.</p>
end_date	<p>Specifies the date and time after which the job expires and is no longer run. After the end_date, if auto_drop is TRUE, the job is dropped. If auto_drop is FALSE, the job is disabled and the STATE of the job is set to COMPLETED.</p> <p>If no value for end_date is specified, the job repeats forever unless max_runs or max_failures is set, in which case the job stops when either value is reached.</p> <p>The value for end_date must be after the value for start_date. If it is not, an error is generated when the job is enabled.</p>
event_spec	<p>This attribute takes two values: the value argument specifies the event condition and the value2 argument specifies the queue specification. For more details, see the descriptions for the event_condition and queue_spec arguments in the <a href="#">"CREATE_JOB Procedure"</a> on page 128-57.</p>

**Table 128–81 (Cont.) Job Attribute Values**

Name	Description
follow_ default_ timezone	<p>If TRUE and if the job <code>start_date</code> is null, then when the <code>default_timezone</code> scheduler attribute is changed, the Scheduler recomputes the next run date and time for this job so that it is in accordance with the new time zone.</p> <p>For example, if the job was set to run at 02:00 in the previous time zone, it will run at 02:00 in the new time zone.</p> <p>If the job <code>start_date</code> is not null, then the time zone for the run date and time for the job is always specified by the time zone of the <code>start_date</code>.</p> <p>If FALSE, the next start date and time for the job is not recomputed when the <code>default_timezone</code> scheduler attribute is changed. In this case, if the old time zone is three hours earlier than the new time zone, then a job scheduled to run at 02:00 in the old time zone runs at 05:00 in the new time zone.</p> <p>Summer and winter transitions do not change the default time zone name.</p>
instance_id	Valid only in an Oracle Real Application Clusters environment. Indicates the instance on which the job is to be run.
instance_ stickiness	<p>This attribute should only be used for a database running in an Oracle Real Application Clusters (Oracle RAC) environment. By default, it is set to TRUE. If you set <code>instance_stickiness</code> to TRUE, jobs start running on the instance with the lightest load and the Scheduler thereafter attempts to run on the instance that it last ran on. If that instance is either down or so overloaded that it does not start new jobs for a significant period of time, another instance runs the job. If the interval between runs is large, <code>instance_stickiness</code> is ignored and the job is handled as if it were a non-sticky job.</p> <p>If <code>instance_stickiness</code> is set to FALSE, each instance of the job runs on the first instance available.</p> <p>For environments other than Oracle RAC, this attribute is not useful because there is only one instance.</p>
job_action	The action that the job performs, depending on the <code>job_type</code> attribute. For example, if <code>job_type</code> is 'STORED_PROCEDURE', <code>job_action</code> contains the name of the stored procedure.
job_class	The class this job is associated with.
job_priority	<p>This attribute specifies the priority of this job relative to other jobs in the same class as this job. If multiple jobs within a class are scheduled to be executed at the same time, the job priority determines the order in which jobs from that class are picked up for execution by the job coordinator. It can be a value from 1 through 5, with 1 being the first to be picked up for job execution.</p> <p>If no job priority is specified when creating a job, the default priority of 3 is assigned to it.</p>
job_type	<p>The type of this job. Valid values are: 'PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', and 'CHAIN'.</p> <p>If this is set, <code>program_name</code> must be NULL.</p>
job_weight	<p>*** Deprecated in Oracle Database 11g Release 2 (11.2). Do not change the value of this attribute from the default, which is 1.</p> <p>Weight of the job for parallel execution.</p>

**Table 128–81 (Cont.) Job Attribute Values**

Name	Description
logging_level	<p>This attribute specifies how much information is logged. The possible options are:</p> <p>DBMS_SCHEDULER.LOGGING_OFF</p> <p>(The default) No logging is performed for this job. However, the logging level of the job class takes precedence and job logging may occur.</p> <p>DBMS_SCHEDULER.LOGGING_FAILED_RUNS</p> <p>The Scheduler logs only jobs that failed, with the reason for failure. If the job class has a higher logging level, then the higher logging level takes precedence.</p> <p>DBMS_SCHEDULER.LOGGING_RUNS</p> <p>The Scheduler writes detailed information to the job log for all runs of each job in this class. If the job class has a higher logging level, then the higher logging level takes precedence.</p> <p>DBMS_SCHEDULER.LOGGING_FULL</p> <p>In addition to recording every run of a job, the Scheduler records all operations performed on the job, including create, enable, disable, alter (with SET_ATTRIBUTE), stop, and so on.</p>
max_failures	<p>This attribute specifies the number of times a job can fail on consecutive scheduled runs before it is automatically disabled. Once a job is disabled, it is no longer executed and its STATE is set to BROKEN in the *_SCHEDULER_JOB views.</p> <p>max_failures can be an integer between 1 to 1,000,000. By default, it is set to NULL, which indicates that new instances of the job are started regardless of how many previous instances have failed.</p>
max_run_duration	<p>This attribute specifies the maximum amount of time that the job should be allowed to run. Its data type is INTERVAL DAY TO SECOND. If this attribute is set to a non-zero and non-NULL value, and job duration exceeds this value, the Scheduler raises an event of type JOB_OVER_MAX_DUR. It is then up to your event handler to decide whether or not to allow the job to continue.</p>
max_runs	<p>This attribute specifies the maximum number of consecutive scheduled runs of the job. Once max_runs is reached, the job is disabled and its state is changed to COMPLETED.</p> <p>max_runs can be an integer between 1 and 1,000,000. By default, it is set to NULL, which means that it repeats forever or until end_date or max_failures is reached.</p>
number_of_arguments	<p>The number of arguments if the program is inlined. If this is set, program_name should be NULL.</p>

**Table 128–81 (Cont.) Job Attribute Values**

Name	Description
parallel_instances	<p>This is a boolean attribute that can be set only for event-based jobs.</p> <p>If FALSE (the default), then if an event is raised and the event-based job that processes that event is already running, the new event is ignored.</p> <p>If TRUE, then an instance of the job is started for every instance of the event, and each job instance is a lightweight job so multiple instances of the same event-based job can run in parallel. Each lightweight job takes its attributes (such as action, maximum run duration, and so on) from the definition of the event-based job (its <i>parent job</i>). After the lightweight job completes, it is dropped. There is no explicit limit to the number of lightweight jobs that can run simultaneously to process multiple instances of the event. However, limitations may be imposed by available system resources.</p> <p>The lightweight jobs are not visible in any of the *_SCHEDULER_JOBS views. However, they are visible in the *_SCHEDULER_RUNNING_JOBS views. The name of each lightweight job is the same as that of the parent job, and a subname is automatically generated to distinguish each lightweight job from its parent and from its siblings.</p>
program_name	<p>The name of a program object to use with this job. If this is set, job_action, job_type and number_of_arguments should be NULL.</p>
raise_events	<p>This attribute tells the Scheduler at what stages of the job execution to raise events. It is a bit vector in which zero or more of the following bits can be set. Each bit has a package constant corresponding to it.</p> <ul style="list-style-type: none"> <li>▪ job_started CONSTANT PLS_INTEGER := 1</li> <li>▪ job_succeeded CONSTANT PLS_INTEGER := 2</li> <li>▪ job_failed CONSTANT PLS_INTEGER :=4</li> <li>▪ job_broken CONSTANT PLS_INTEGER :=8</li> <li>▪ job_completed CONSTANT PLS_INTEGER :=16</li> <li>▪ job_stopped CONSTANT PLS_INTEGER :=32</li> <li>▪ job_sch_lim_reached CONSTANT PLS_INTEGER :=64</li> <li>▪ job_disabled CONSTANT PLS_INTEGER :=128</li> <li>▪ job_chain_stalled CONSTANT PLS_INTEGER :=256</li> <li>▪ job_all_events CONSTANT PLS_INTEGER := 511</li> <li>▪ job_run_completed CONSTANT PLS_INTEGER := job_succeeded + job_failed + job_stopped</li> </ul> <p><a href="#">Table 128–82</a> describes these event types in detail.</p>
repeat_interval	<p>Either a PL/SQL function returning the next date and time on which to run, or calendaring syntax expression. If this is set, schedule_name should be NULL. See "<a href="#">Calendaring Syntax</a>" on page 128-20 for more information.</p>

**Table 128–81 (Cont.) Job Attribute Values**

Name	Description
restartable	<p>This attribute specifies whether or not a job can be restarted in case of failure. By default, jobs are not restartable and this attribute is set to <code>FALSE</code>. Setting this to <code>TRUE</code> means that if a job fails while running, it is restarted from the beginning point of the job.</p> <p>In the case of a chain job, if this attribute is <code>TRUE</code>, the chain is restarted from the beginning after an application failure. If this attribute is <code>FALSE</code>, or if there has been a database failure, the chain is restarted at the last running step. The <code>restart_on_recovery</code> attribute of that step then determines if the step is restarted or marked as stopped. (If marked as stopped, the chain evaluates rules and continues.)</p> <p>Note that setting this attribute to <code>TRUE</code> might lead to data inconsistencies in some situations, for example, if data is committed within a job.</p> <p>Retries on errors are not counted as regular runs. The run count or failure count is not incremented until the job succeeds or has failed all its six retries.</p> <p>The restartable attribute is used by the Scheduler to determine whether to retry the job not only on regular application errors, but after a database malfunction as well. The Scheduler retries the job a maximum of six times. The first time, it waits for one second and multiplies this wait time with a factor of 10 each time thereafter.</p> <p>Both the run count and failure count are incremented by 1 if the job has failed all its six retries. If the job immediately succeeds, or it succeeds on one of its retries, run count is incremented by 1.</p> <p>The Scheduler stops retrying a job when:</p> <ul style="list-style-type: none"> <li>■ One of the retries succeeds.</li> <li>■ All of its six retries have failed.</li> <li>■ The next retry would occur after the next regularly scheduled run of the job.</li> </ul> <p>The Scheduler no longer retries the job if the next scheduled retry is past the next regularly scheduled run for repeating jobs.</p>
schedule_limit	<p>In heavily loaded systems, jobs are not always started at their scheduled time. This attribute enables you to have the Scheduler not start a job at all if the delay in starting the job is larger than the interval specified. It can be a value of 1 minute to 99 days. For example, if a job was supposed to start at noon and the schedule limit is set to 60 minutes, the job will not be run if it has not started to run by 1:00 p.m.</p> <p>If <code>schedule_limit</code> is not specified, the job is executed at some later date as soon as there are resources available to run it. By default, this attribute is set to null, which indicates that the job can be run at any time after its scheduled time. A scheduled job run that is skipped because of this attribute does not count against the number of runs and failures of the job. An entry in the job log reflects the skipped run.</p>
schedule_name	<p>The name of a schedule, window, or group of type <code>WINDOW</code> to use as the schedule for this job. If this is set, <code>end_date</code>, <code>start_date</code> and <code>repeat_interval</code> should all be <code>NULL</code>.</p>
start_date	<p>The original date and time on which this job started or is scheduled to start. If this is set, <code>schedule_name</code> should be <code>NULL</code>.</p>

**Table 128–81 (Cont.) Job Attribute Values**

Name	Description
stop_on_window_close	<p>This attribute only applies if the schedule of a job is a window or a window group. Setting this attribute to TRUE implies that the job should stop once the associated window is closed. The job is stopped using the stop_job procedure with force set to FALSE.</p> <p>By default, stop_on_window_close is set to FALSE. Therefore, if you do not set this attribute, the job continues after the window closes.</p> <p>Note that, although the job is allowed to continue, its resource allocation will probably change because closing a window generally also implies a change in resource plans.</p>

The following event types are valid values for the raise\_events attribute in [Table 128–81, "Job Attribute Values"](#).

**Table 128–82 Event Types Raised by the Scheduler**

Event Type	Description
job_all_events	Not an event, but a constant that provides an easy way for you to enable all events
job_broken	The job has been disabled and has changed to the BROKEN state because it exceeded the number of failures defined by the max_failures job attribute
job_chain_stalled	A job running a chain is in the CHAIN_STALLED state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain evaluation_interval is set to NULL. No progress is made in the chain unless there is manual intervention.
job_completed	The job completed because it reached its max_runs or end_date
job_disabled	The job was disabled by the Scheduler or by a call to SET_ATTRIBUTE
job_failed	The job failed, either due to an error or an abnormal termination.
job_over_max_dur	The job exceeded the maximum run duration specified by its max_run_duration attribute. (Note: you do not need to enable this event with the raise_events job attribute; it is always enabled.)
job_run_completed	A job run either failed, succeeded, or was stopped
job_sch_lim_reached	The schedule limit of the job was reached. The job was not started because the delay in starting the job exceeded the value of the schedule_limit job attribute.
job_started	The job started
job_stopped	The job was stopped by a call to STOP_JOB
job_succeeded	The job completed successfully

### Program Attribute Values

[Table 128–83](#) lists program attribute values.

**Note:** See the CREATE\_PROGRAM procedure for more complete descriptions of the attributes in this table.

**Table 128–83 Program Attribute Values**

Name	Description
comments	An optional comment. This can describe what the program does or give usage details.
detached	If TRUE, the program is a detached program. See <i>Oracle Database Administrator's Guide</i> for information about detached jobs and detached programs.
number_of_arguments	The number of arguments required by the stored procedure or other executable that the program invokes
program_action	The action that the program performs, indicated by the program_type attribute. For example, if program_type is 'STORED_PROCEDURE', program_action contains the name of the stored procedure.
program_type	The type of program. This must be one of these supported program types: 'PLSQL_BLOCK', 'STORED_PROCEDURE', and 'EXECUTABLE'.

### Schedule Attribute Values

[Table 128–84](#) lists schedule attribute values.

**Note:** See the CREATE\_SCHEDULE and CREATE\_CALENDAR\_SCHEDULE procedures for more complete descriptions of the attributes in this table.

**Table 128–84 Schedule Attribute Values**

Name	Description
comments	An optional comment.
end_date	The cutoff date and time after which the schedule does not specify any dates.
event_spec	This attribute takes two values: the value argument should contain the event condition and the value2 argument should contain the queue specification. For more details, see the descriptions for the event_condition and queue_spec arguments to the <a href="#">"CREATE_JOB Procedure"</a> on page 128-57.
repeat_interval	An attribute specifying how often the schedule should repeat, using the calendaring syntax. See <a href="#">"Calendaring Syntax"</a> on page 128-20 for more information.
start_date	The start or reference date and time used by the calendaring syntax.

### File Watcher Attribute Values

[Table 128–85](#) lists file watcher attribute values.

**Table 128–85 File Watcher Attribute Values**

Parameter	Description
destination	Remote host name or IP address where the file is expected to arrive. If NULL, destination is the local host.
directory_path	Directory in which the file is expected to arrive. The single wildcard '?' at the beginning of the path denotes the Oracle home path. For example, '?/rdbms/log' denotes the rdbms/log subdirectory of the Oracle home directory.

**Table 128–85 (Cont.) File Watcher Attribute Values**

Parameter	Description
file_name	Name of the file being looked for. Two wildcards are permitted anywhere in the file name: '?' denotes any single character, and '*' denotes zero or more characters. This attribute cannot be NULL.
credential_name	Name of a valid Scheduler credential object. The file watcher uses the credential to authenticate itself with the host operating system to access the watched-for file. The file watcher owner must have the EXECUTE privilege on the credential. Cannot be NULL.
min_file_size	Minimum file size in bytes before the file watcher considers the file found. Default is 0.
steady_state_duration	Minimum time interval that the file must remain unchanged before the file watcher considers the file found. If NULL, an internal value is used.
comments	Optional comment.

**Job Class Attribute Values**

Table 128–86 lists job class attribute values.

**Note:** See the CREATE\_JOB\_CLASS procedure for more complete descriptions of the attributes in this table.

**Table 128–86 Job Class Attribute Values**

Name	Description
comments	An optional comment about the class.
log_history	This attribute controls the number of days that job log entries for jobs in this class are retained. It helps prevent the job log from growing indiscriminately.  The range of valid values is 0 through 1000000. If set to 0, no history is kept. If NULL, retention days are set by the log_history Scheduler attribute (set with SET_SCHEDULER_ATTRIBUTE).
logging_level	This attribute specifies how much information is logged. The valid values are: <ul style="list-style-type: none"> <li>■ DBMS_SCHEDULER.LOGGING_OFF No logging is performed for any jobs in this class.</li> <li>■ DBMS_SCHEDULER.LOGGING_FAILED_RUNS The Scheduler logs only jobs in the class that failed, with the reason for failure.</li> <li>■ DBMS_SCHEDULER.LOGGING_RUNS The Scheduler writes detailed information to the job log for all runs of each job in this class. This is the default.</li> <li>■ DBMS_SCHEDULER.LOGGING_FULL The Scheduler records all operations performed on all jobs in this class, in addition to recording every run of a job. Every time a job is created, enabled, disabled, altered (with SET_ATTRIBUTE), stopped, and so on, an entry is recorded in the log.</li> </ul>
resource_consumer_group	The resource consumer group that a class is associated with. All jobs in the class run under this resource consumer group. See <i>Oracle Database Administrator's Guide</i> for a description of resource consumer groups and the Database Resource Manager.



**Table 128–86 (Cont.) Job Class Attribute Values**

Name	Description
service	The database service that the jobs in the job class have affinity to. If both the <code>resource_consumer_group</code> and <code>service</code> attributes are set for a job class, and if the service is mapped to a resource consumer group, the <code>resource_consumer_group</code> attribute takes precedence.

**Window Attribute Values**

[Table 128–87](#) lists window attribute values.

**Note:** See the `CREATE_WINDOW` procedure for more complete descriptions of the attributes in this table.

**Table 128–87 Window Attribute Values**

Name	Description
comments	An optional comment about the window.
duration	The duration of the window.
end_date	The date after which the window no longer opens. If this is set, <code>schedule_name</code> must be NULL.
repeat_interval	An attribute specifying how often the schedule should repeat, using the calendaring syntax. PL/SQL date functions are not allowed. If this is set, <code>schedule_name</code> must be NULL. See " <a href="#">Calendaring Syntax</a> " on page 128-20 for more information.
resource_plan	The resource plan to be associated with a window. When the window opens, the system switches to this resource plan. When the window closes, the original resource plan is restored. If a resource plan has been made active with the <code>force</code> option, no resource plan switch occurs.  Only one resource plan can be associated with a window. It may be NULL or the empty string (""). When it is NULL, the resource plan that is in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window.
schedule_name	The name of a schedule to use with this window. If this is set, <code>start_date</code> , <code>end_date</code> , and <code>repeat_interval</code> must all be NULL.
start_date	The next date and time on which this window is scheduled to open. If this is set, <code>schedule_name</code> must be NULL.
window_priority	The priority of the window. Must be either 'LOW' (default) or 'HIGH'.

**Chain Attribute Values**

[Table 128–88](#) lists chain attribute values.

**Note:** See the `CREATE_CHAIN` procedure for more complete descriptions of the attributes in this table.

**Table 128–88 Chain Attribute Values**

Name	Description
comments	An optional comment describing the purpose of the chain.

**Table 128–88 (Cont.) Chain Attribute Values**

Name	Description
evaluation_interval	<p>If not NULL, provides an additional evaluation of the chain at this interval, as well as at normal evaluation times (when the job starts, when a step completes, or when an event that is associated with an event step arrives)</p> <p>This attribute should only to be used when chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler, because the extra interval is CPU intensive. For most chains, the normal evaluation times are sufficient.</p>
rule_set_name	<p>In the normal case, no rule set should be passed in. The Scheduler automatically creates a rule set and associated empty evaluation context. You then use DEFINE_CHAIN_RULE to add rules and DROP_CHAIN_RULE to remove them.</p> <p>Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created is of the form SCHED_RULESET\$ {N} and the evaluation context created is of the form SCHED_EVCTX\$ {N}</p> <p>See <i>Oracle Streams Concepts and Administration</i> for information on rules and rule sets.</p>

### Database Destination Attribute Values

Table 128–89 lists database destination attribute values.

**Note:** See the CREATE\_DATABASE\_DESTINATION procedure for more complete descriptions of the attributes in this table.

**Table 128–89 Database Destination Attribute Values**

Name	Description
agent	<p>The name of the external destination (also known as agent destination) that is used to connect to the remote database.</p> <p>You can obtain valid external destination names from the view ALL_SCHEDULER_EXTERNAL_DESTS.</p>
connect_info	<p>The TNS connect descriptor that identifies the remote database to connect to, or the net service name (alias) in tnsnames.ora that resolves to the connect descriptor.</p> <p>Note: This corresponds to the tns_name argument of CREATE_DATABASE_DESTINATION.</p>
enabled	If TRUE, the database destination is enabled.
comments	An optional comment about the database destination.

### External Destination Attribute Values

Table 128–90 lists external destination attribute values.

**Note:** External destinations are created only implicitly by registering a remote Scheduler agent with the local database.

**Table 128–90 External Destination Attribute Values**

Name	Description
hostname	(GET_ATTRIBUTE only) The fully qualified host name (including domain) or IP address of the computer on which the Scheduler agent resides.
port	(GET_ATTRIBUTE only) The TCP port number on which the agent listens.
ip_address	(GET_ATTRIBUTE only) The IP address of the host on which the agent resides.
enabled	If TRUE, the external destination is enabled.
comments	An optional comment about the external destination.

**Group Attribute Values**

[Table 128–91](#) lists group attribute values.

---



---

**Note:** See the CREATE\_GROUP procedure for more complete descriptions of the attributes in this table.

---



---

**Table 128–91 Group Attribute Values**

Name	Description
group_type	(GET_ATTRIBUTE only) The group type (either WINDOW, DB_DEST, or EXTERNAL_DEST).
member_name	Comma-separated list of members. Replaces the existing list of members. To add one or more members to the existing list, use ADD_GROUP_MEMBER.  Note: this attribute corresponds to the member argument of CREATE_GROUP.
enabled	If TRUE, the group is enabled.
comments	An optional comment about the group.
number_of_members	(GET_ATTRIBUTE only) The number of members in the group.

**Credential Attribute Values**

[Table 128–92](#) lists credential attribute values.

---



---

**Note:** See the CREATE\_CREDENTIAL procedure for more complete descriptions of the attributes in this table.

---



---

**Table 128–92 Credential Attribute Values**

Name	Description
username	The user name for logging into to the host operating system or remote Oracle database. Maximum length is 64.
password	The password for the user name. Maximum length is 128.
comments	A description of the credential. Maximum length is 240.
windows_domain	For a Windows remote executable target, this is the domain that the specified user belongs to. Maximum length is 64.

**Table 128–92 (Cont.) Credential Attribute Values**

Name	Description
database_role	The value of the database_role attribute is used as the system privilege for logging into a remote database to run a remote database job.  Valid values are: SYSDBA and SYSOPER.

## SET\_ATTRIBUTE\_NULL Procedure

This procedure sets an attribute of an object to NULL. The attributes that can be set depend on the object being altered. If the object is enabled, it is disabled before being altered and reenabled afterward. If the object cannot be reenabled, an error is generated and the object is left in a disabled state.

### Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE_NULL (
    name           IN VARCHAR2,
    attribute      IN VARCHAR2);
```

### Parameters

**Table 128–93** SET\_ATTRIBUTE\_NULL Procedure Parameters

Parameter	Description
name	The name of the object
attribute	The attribute being changed

### Usage Notes

To run SET\_ATTRIBUTE\_NULL for a window, group of type WINDOW, or job class, you must have the MANAGE\_SCHEDULER privilege. Otherwise, you must be the owner of the object being altered or have ALTER privileges on that object or have the CREATE ANY JOB privilege.

## SET\_JOB\_ANYDATA\_VALUE Procedure

This procedure sets the value for an argument of the associated program for a job, encapsulated in an `AnyData` object. It overrides any default value set for the program argument. `NULL` is a valid assignment for a program argument. The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the [DEFINE\\_ANYDATA\\_ARGUMENT Procedure](#)

Scheduler does no type checking of the argument at any time.

`SET_JOB_ANYDATA_VALUE` is overloaded.

### Syntax

Sets a program argument by its position.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (
    job_name           IN VARCHAR2,
    argument_position  IN PLS_INTEGER,
    argument_value     IN SYS.ANYDATA);
```

Sets a program argument by its name.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (
    job_name           IN VARCHAR2,
    argument_name      IN VARCHAR2,
    argument_value     IN SYS.ANYDATA);
```

### Parameters

**Table 128–94** SET\_JOB\_ANYDATA\_VALUE Procedure Parameters

Parameter	Description
<code>job_name</code>	The name of the job to be altered
<code>argument_name</code>	The name of the program argument being set
<code>argument_position</code>	The position of the program argument being set
<code>argument_value</code>	The new value to be assigned to the program argument, encapsulated in an <code>AnyData</code> object

### Usage Notes

`SET_JOB_ANYDATA_VALUE` requires that you own the job or have `ALTER` privileges on that job. You can also set a job argument value if you have the `CREATE ANY JOB` privilege.

`SET_JOB_ANYDATA_VALUE` does not apply to lightweight jobs because lightweight jobs cannot take `AnyData` arguments.

#### See Also:

- ["SET\\_JOB\\_ARGUMENT\\_VALUE Procedure"](#) on page 128-147
- ["DEFINE\\_ANYDATA\\_ARGUMENT Procedure"](#) on page 128-75

## SET\_JOB\_ARGUMENT\_VALUE Procedure

This procedure sets the value of an argument in a program associated with a job. It overrides any default value set for the program argument. NULL is a valid assignment for a program argument. The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the [DEFINE\\_PROGRAM\\_ARGUMENT Procedure](#) or the [DEFINE\\_METADATA\\_ARGUMENT Procedure](#)

Scheduler does no type checking of the argument at any time.

SET\_JOB\_ARGUMENT\_VALUE is overloaded.

### Syntax

Sets an argument value by position:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
  job_name           IN VARCHAR2,
  argument_position  IN PLS_INTEGER,
  argument_value     IN VARCHAR2);
```

Sets an argument value by name:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
  job_name           IN VARCHAR2,
  argument_name      IN VARCHAR2,
  argument_value     IN VARCHAR2);
```

### Parameters

**Table 128–95 SET\_JOB\_ARGUMENT\_VALUE Procedure Parameters**

Parameter	Description
job_name	The name of the job to be altered
argument_name	The name of the program argument being set
argument_position	The position of the program argument being set
argument_value	The new value to be set for the program argument. To set a non-VARCHAR value, use the SET_JOB_ANYDATA_VALUE procedure.

### Usage Notes

SET\_JOB\_ARGUMENT\_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also set a job argument value if you have the CREATE ANY JOB privilege.

SET\_JOB\_ARGUMENT\_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

SET\_JOB\_ARGUMENT\_VALUE can be used to set arguments of lightweight jobs but only if the argument is of type VARCHAR2.

**See Also:**

- ["SET\\_JOB\\_ANYDATA\\_VALUE Procedure"](#) on page 128-146
- ["DEFINE\\_PROGRAM\\_ARGUMENT Procedure"](#) on page 128-83



## SET\_JOB\_ATTRIBUTES Procedure

This procedure changes an attribute of a job.

### Syntax

```
DBMS_SCHEDULER.SET_JOB_ATTRIBUTES (
  jobattr_array      IN JOBATTR_ARRAY,
  commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

### Parameters

**Table 128–96** SET\_JOB\_ATTRIBUTES Procedure Parameters

Parameter	Description
jobattr_array	The array of job attribute changes.
commit_semantics	The commit semantics. The following types are supported: <ul style="list-style-type: none"> <li>▪ STOP_ON_FIRST_ERROR returns on the first error and commits previous successful attribute changes to disk. This is the default.</li> <li>▪ TRANSACTIONAL returns on the first error and rolls back everything that happened before that error.</li> <li>▪ ABSORB_ERRORS tries to absorb any errors and complete the rest of the job attribute changes on the list. It commits all the successful changes. If errors occur, you can query the view SCHEDULER_BATCH_ERRORS for details.</li> </ul>

### Usage Notes

Calling SET\_ATTRIBUTE on an enabled job disables the job, changes the attribute value, and reenables the job. SET\_JOB\_ATTRIBUTES changes the attribute values in the context of a single transaction.

## SET\_SCHEDULER\_ATTRIBUTE Procedure

This procedure sets the value of a Scheduler attribute. This takes effect immediately but the resulting changes may not be seen immediately.

[Table 128–97](#) provides short attribute descriptions for the SET\_SCHEDULER\_ATTRIBUTE procedure. For complete descriptions, see section "Setting Scheduler Preferences" in *Oracle Database Administrator's Guide*.

### Syntax

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE (  
    attribute      IN VARCHAR2,  
    value          IN VARCHAR2);
```

## Parameters

**Table 128–97 SET\_SCHEDULER\_ATTRIBUTE Procedure Parameters**

Parameter	Description
attribute	<p>The name of the Scheduler attribute. Possible values are:</p> <ul style="list-style-type: none"> <li>▪ 'default_timezone': Repeating jobs and windows that use the calendaring syntax retrieve the time zone from this attribute when start_date is not specified. See "Calendaring Syntax" on page 128-20 for more information.</li> <li>▪ 'email_server': The SMTP server address that the Scheduler uses to send e-mail notifications for job state events. E-mail notifications cannot be sent if this attribute is NULL.</li> <li>▪ 'email_sender': The default e-mail address of the sender of job state e-mail notifications.</li> <li>▪ 'email_server_credential': The schema and name of an existing credential object that SYS has execute object privileges on. Default is NULL. The username and password stored in this credential are used to authenticate with the e-mail server when sending e-mail notifications.</li> </ul> <p>This functionality is available with Oracle Database 11g Release 2 (11.2.0.2).</p> <ul style="list-style-type: none"> <li>▪ 'email_server_encryption': This attribute indicates whether or not encryption is enabled for this email server connection, and if so, at what point encryption starts, and with which protocol. Values are: <ul style="list-style-type: none"> <li>– NONE: the default, indicating no encryption used</li> <li>– SSL_TLS: indicating that either SSL or TLS are used, from the beginning of the connection</li> <li>– STARTTLS: indicating that the connection starts unencrypted, but the command STARTTLS is sent to the e-mail server and starts encryption</li> </ul> <p>This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).</p> </li> <li>▪ 'event_expiry_time': The time, in seconds, before a job state event generated by the Scheduler expires from the Scheduler event queue. If NULL, job state events expire after 24 hours.</li> <li>▪ 'log_history': The number of days that log entries for both the job log and the window log are retained. Default is 30 and the range of valid values is 0 through 1000000.</li> <li>▪ 'max_job_slave_processes': This Scheduler attribute is not used.</li> </ul>
value	The new value of the attribute

## Usage Notes

To run SET\_SCHEDULER\_ATTRIBUTE, you must have the MANAGE SCHEDULER privilege.

**See Also:** *Oracle Database Administrator's Guide* for more detailed descriptions of Scheduler attributes

## STOP\_JOB Procedure

This procedure stops currently running jobs or all jobs in a job class. After stopping the job, the state of a one-time job is set to STOPPED, whereas the state of a repeating job is set to SCHEDULED or COMPLETED, depending on whether the next run of the job is scheduled.

If a job pointing to a chain is stopped, all running steps of the running chain are stopped.

If a job has multiple destinations, the database attempts to stop the job at all destinations.

For external jobs, STOP\_JOB stops only the external process that was directly started by the job action. It does not stop child processes of external jobs.

### Syntax

```
DBMS_SCHEDULER.STOP_JOB (
  job_name          IN VARCHAR2
  force             IN BOOLEAN DEFAULT FALSE
  commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

### Parameters

**Table 128–98 STOP\_JOB Procedure Parameters**

Parameter	Description
job_name	<p>Name of a job to stop. Can be a comma-separated list of jobs, where each entry can be one of the following:</p> <ul style="list-style-type: none"> <li>▪ Job name: the name of an existing job, optionally preceded by a schema name and dot separator.</li> <li>▪ Job destination ID: a number, obtained from the JOB_DEST_ID column of the *_SCHEDULER_JOB_DESTS views, that represents the unique combination of a job, a credential, and a destination.</li> <li>▪ Job class: the name of a job class. Must be preceded by the SYS schema name and a dot separator.</li> </ul> <p>If you specify a job class, all jobs that belong to that job class are stopped. If you specify a job that was created with a destination group as its destination_name attribute, all job instances on all destinations are stopped.</p>
force	<p>If force is set to FALSE, the Scheduler tries to gracefully stop the job using an interrupt mechanism. This method gives control back to the slave process, which can update the status of the job in the job queue to stopped. If this fails, an error is returned.</p> <p>If force is set to TRUE, the Scheduler immediately terminates the job slave. Oracle recommends that STOP_JOB with force set to TRUE be used only after a STOP_JOB with force set to FALSE has failed.</p> <p>Use of the force option requires the MANAGE_SCHEDULER system privilege.</p>

**Table 128–98 (Cont.) STOP\_JOB Procedure Parameters**

Parameter	Description
<code>commit_semantics</code>	<p>The commit semantics. The following two types are supported:</p> <ul style="list-style-type: none"> <li>▪ <code>STOP_ON_FIRST_ERROR</code>: The procedure returns on the first error and commits previous successful stop operations to disk. This is the default.</li> <li>▪ <code>ABSORB_ERRORS</code>: The procedure tries to absorb any errors, stops the rest of the jobs, and commits all the successful stop operations. This type is available only if no job classes are specified in the <code>job_name</code> list. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details.</li> </ul>

## Usage Notes

`STOP_JOB` without the `force` option requires that you be the owner of the job or have `ALTER` privileges on that job. You can also stop a job if you have the `CREATE ANY JOB` or `MANAGE SCHEDULER` privilege.

`STOP_JOB` with the `force` option requires that you have the `MANAGE SCHEDULER` privilege.

## Example

The following is an example of using `STOP_JOB`.

```
BEGIN
  DBMS_SCHEDULER.STOP_JOB('DSS.ETLJOB', 984, 1223, SYS.ETL_JOBCLASS');
END;
```

---

---

## DBMS\_SERVER\_ALERT

The `DBMS_SERVER_ALERT` package enables you to configure the Oracle Database server to issue an alert when a threshold for a specified server metric has been violated. You can configure both warning and critical thresholds for a large number of predefined metrics.

If a warning threshold is reached, the server generates a severity level 5 alert. If a critical threshold is reached, the server generates a severity level 1 alert.

The chapter contains the following topics:

- [Using DBMS\\_SERVER\\_ALERT](#)
  - Security Model
  - Object Types
  - Relational Operators
  - Supported Metrics
- [Summary of DBMS\\_SERVER\\_ALERT Subprograms](#)

---

## Using DBMS\_SERVER\_ALERT

This section contains topics which relate to using the DBMS\_SERVER\_ALERT package. The following topics define constants used in package procedures.

- [Security Model](#)
- [Object Types](#)
- [Relational Operators](#)
- [Supported Metrics](#)



## Security Model

The user needs DBA or IMP\_FULL\_DATABASE roles to use the DBMS\_SERVER\_ALERT package.

## Object Types

You qualify the metric by an individual object for the following object types.

**Table 129–1 Object Types Defined as Constants**

Constant	Description
OBJECT_TYPE_SYSTEM	Metrics collected on the system level for each instance.
OBJECT_TYPE_FILE	Metrics collected on the file level. These are used for AVG_FILE_READ_TIME and AVG_FILE_WRITE_TIME metrics.
OBJECT_TYPE_SERVICE	Metrics collected on the service level. Currently ELAPSED_TIME_PER_CALL and CPU_TIME_PER_CALL are collected.
OBJECT_TYPE_TABLESPACE	Metrics collected on the tablespace level. <b>Note:</b> Dictionary managed tablespaces are not supported.
OBJECT_TYPE_EVENT_CLASS	Metrics collected on wait event class level. Currently supported metrics are AVG_USERS_WAITING and DB_TIME_WAITING.
OBJECT_TYPE_SESSION	Metrics collected on the session level. Currently only BLOCKED_USERS is collected. The threshold can only be set at the instance level, which means that no object name should be specified when setting the threshold for this type of metric.

## Relational Operators

You can specify a relational comparison operator to determine whether or not a given metric's value violates the threshold setting. The server supports the following operators.

**Table 129–2 Relational Operators Defined as Constants**

Constant	Description
OPERATOR_CONTAINS	A metric value matching an entry in a list of threshold values is considered a violation.
OPERATOR_DO_NOT_CHECK	The metric value is not compared to the threshold value, and no alerts are generated. Use this operator to disable alerts for a metric.
OPERATOR_EQ	A metric value equal to the threshold value is considered a violation.
OPERATOR_GE	A metric value greater than or equal to the threshold value is considered a violation.
OPERATOR_GT	A metric value greater than the threshold value is considered a violation.
OPERATOR_LE	A metric value less than or equal to the threshold value is considered a violation.
OPERATOR_LT	A metric value less than the threshold value is considered a violation.
OPERATOR_NE	A metric value not equal to the threshold value is considered a violation.

## Supported Metrics

The following metrics are supported. All internal metric names are supplied as package constants.

**Table 129–3 List of Supported Metrics**

<b>Metric Name (Internal)</b>	<b>Metric Name (External)</b>	<b>Units</b>
AVG_FILE_READ_TIME	Average File Read Time	Microseconds
AVG_FILE_WRITE_TIME	Average File Write Time	Microseconds
AVG_USERS_WAITING	Average Number of Users Waiting on a Class of Wait Events	Count of sessions
BLOCKED_USERS	Number of Users blocked by some Session	Number of Users
BRANCH_NODE_SPLITS_SEC	Branch Node Splits (for each second)	Splits for each Second
BRANCH_NODE_SPLITS_TXN	Branch Node Splits (for each transaction)	Splits for each Transaction
BUFFER_CACHE_HIT	Buffer Cache Hit (%)	% of cache accesses
CLUSTER_MSG_WAIT_SCT	Cluster Messaging Wait (by session count)	Count of sessions
CLUSTER_MSG_WAIT_TIME	Cluster Messaging Wait (by time)	Microseconds
CONSISTENT_CHANGES_SEC	Consistent Changes (for each second)	Changes for each Second
CONSISTENT_CHANGES_TXN	Consistent Changes (for each transaction)	Changes for each Transaction
CONSISTENT_GETS_SEC	Consistent Gets (for each second)	Gets for each Second
CONSISTENT_GETS_TXN	Consistent Gets (for each transaction)	Gets for each Transaction
CONTENTION_WAIT_SCT	Internal Contention Wait (by session count)	Count of sessions
CONTENTION_WAIT_TIME	Internal Contention Wait (by time)	Microseconds
CPU_TIME_PER_CALL	CPU time for each user call for each service	Microseconds for each call
CR_BLOCKS_CREATED_SEC	CR Blocks Created (for each second)	Blocks for each Second
CR_BLOCKS_CREATED_TXN	CR Blocks Created (for each transaction)	Blocks for each Transaction
CR_RECORDS_APPLIED_SEC	CR Undo Records Applied (for each second)	Records for each Second
CR_RECORDS_APPLIED_TXN	CR Undo Records Applied (for each transaction)	Records for each Transaction
CURSOR_CACHE_HIT	Cursor Cache Hit (%)	% of soft parses
DATABASE_WAIT_TIME	Database Wait Time (%)	% of all database time
DATABASE_CPU_TIME	Database CPU Time (%)	% of all database time
DATA_DICT_HIT	Data Dictionary Hit (%)	% of dictionary accesses
DATA_DICT_MISS	Data Dictionary Miss (%)	% of dictionary accesses

**Table 129-3 (Cont.) List of Supported Metrics**

<b>Metric Name (Internal)</b>	<b>Metric Name (External)</b>	<b>Units</b>
DB_BLKGETS_SEC	DB Block Gets (for each second)	Gets for each Second
DB_BLKGETS_TXN	DB Block Gets (for each transaction)	Gets for each Transaction
DB_TIME_WAITING	Percent of Database Time Spent Waiting on a Class of Wait Events	% of Database Time
DBR_IO_LIMIT_WAIT_SCT	Resource Mgr I/O Limit Wait (by session count)	Count of sessions
DBR_IO_LIMIT_WAIT_TIME	Resource Mgr I/O Limit Wait (by time)	Microseconds
DBR_CPU_LIMIT_WAIT_SCT	Resource Mgr CPU Limit Wait (by session count)	Count of sessions
DBR_CPU_LIMIT_WAIT_TIME	Resource Mgr CPU Limit Wait (by time)	Microseconds
DBR_USR_LIMIT_WAIT_SCT	Resource Mgr User Limit Wait (by session count)	Count of sessions
DBR_USR_LIMIT_WAIT_TIME	Resource Mgr User Limit Wait (by time)	Microseconds
DBWR_CKPT_SEC	DBWR Checkpoints (for each second)	Checkpoints for each Second
DISK_IO	Disk I/O	Milliseconds
DISK_IO_WAIT_SCT	Disk I/O Wait (by session count)	Count of sessions
DISK_SORT_SEC	Sorts to Disk (for each second)	Sorts for each Second
DISK_SORT_TXN	Sorts to Disk (for each transaction)	Sorts for each Transaction
ELAPSED_TIME_PER_CALL	Elapsed time for each user call for each service	Microseconds for each call
ENQUEUE_DEADLOCKS_SEC	Enqueue Deadlocks (for each second)	Deadlocks for each Second
ENQUEUE_DEADLOCKS_TXN	Enqueue Deadlocks (for each transaction)	Deadlocks for each Transaction
ENQUEUE_REQUESTS_SEC	Enqueue Requests (for each second)	Requests for each Second
ENQUEUE_REQUESTS_TXN	Enqueue Requests (for each transaction)	Requests for each Transaction
ENQUEUE_TIMEOUTS_SEC	Enqueue Timeouts (for each second)	Timeouts for each Second
ENQUEUE_TIMEOUTS_TXN	Enqueue Timeouts (for each transaction)	Timeouts for each Transaction
ENQUEUE_WAITS_SEC	Enqueue Waits (for each second)	Waits for each Second
ENQUEUE_WAITS_TXN	Enqueue Waits (for each transaction)	Waits for each Transaction
EXECUTE_WITHOUT_PARSE	Executes Performed Without Parsing	% of all executes
FULL_INDEX_SCANS_SEC	Fast Full Index Scans (for each second)	Scans for each Second
FULL_INDEX_SCANS_TXN	Fast Full Index Scans (for each transaction)	Scans for each Transaction

**Table 129–3 (Cont.) List of Supported Metrics**

<b>Metric Name (Internal)</b>	<b>Metric Name (External)</b>	<b>Units</b>
GC_AVG_CR_GET_TIME	Global Cache CR Request	Milliseconds
GC_AVG_CUR_GET_TIME	Global Cache Current Request	Milliseconds
GC_BLOCKS_CORRUPT	Global Cache Blocks Corrupt	Blocks
GC_BLOCKS_LOST	Global Cache Blocks Lost	Blocks
HARD_PARSSES_SEC	Hard Parses (for each second)	Parses for each Second
HARD_PARSSES_TXN	Hard Parses (for each transaction)	Parses for each Transaction
LEAF_NODE_SPLITS_SEC	Leaf Node Splits (for each second)	Splits for each Second
LEAF_NODE_SPLITS_TXN	Leaf Node Splits (for each transaction)	Splits for each Transaction
LIBRARY_CACHE_HIT	Library Cache Hit (%)	% of cache accesses
LIBRARY_CACHE_MISS	Library Cache Miss (%)	% of cache accesses
LOG_SWITCH_SEC	Background Checkpoints (for each second)	Checkpoints for each Second
LOGONS_CURRENT	Current Number of Logons	Number of Logons
LOGONS_SEC	Cumulative Logons (for each second)	Logons for each Second
LOGONS_TXN	Cumulative Logons (for each transaction)	Logons for each Transaction
LONG_TABLE_SCANS_SEC	Scans on Long Tables (for each second)	Scans for each Second
LONG_TABLE_SCANS_TXN	Scans on Long Tables (for each transaction)	Scans for each Transaction
OPEN_CURSORS_SEC	Cumulative Open Cursors (for each second)	Cursors for each Second
MEMORY_SORTS_PCT	Sorts in Memory (%)	% of sorts
NETWORK_BYTES_SEC	Network Bytes, for each second	Bytes for each Second
NETWORK_MSG_WAIT_SCT	Network Message Wait (by session count)	Count of sessions
NETWORK_MSG_WAIT_TIME	Network Message Wait (by time)	Microseconds
OPEN_CURSORS_CURRENT	Current Number of Cursors	Number of Cursors
OPEN_CURSORS_TXN	Cumulative Open Cursors (for each transaction)	Cursors for each Transaction
OS_SCHED_CPU_WAIT_SCT	Operating System Scheduler CPU Wait (by session count)	Count of sessions
OS_SCHED_CPU__WAIT_TIME	Operating System Scheduler CPU Wait (by time)	Microseconds
OS_SERVICE_WAIT_SCT	Operating System Service Wait (by session count)	Count of sessions
OS_SERVICE_WAIT_TIME	Operating System Service Wait (by time)	Microseconds
OTHER_WAIT_SCT	Other Waits (by session count)	Count of sessions
OTHER_WAIT_TIME	Other Waits (by time)	Microseconds

**Table 129-3 (Cont.) List of Supported Metrics**

<b>Metric Name (Internal)</b>	<b>Metric Name (External)</b>	<b>Units</b>
PARSE_FAILURES_SEC	Parse Failures (for each second)	Parses for each Second
PARSE_FAILURES_TXN	Parse Failures (for each transaction)	Parses for each Transaction
PGA_CACHE_HIT	PGA Cache Hit (%)	% bytes processed in PGA
PHYS_DESGN_WAIT_SCT	Physical Design Wait (by session count)	Count of sessions
PHYS_DESGN_WAIT_TIME	Physical Design Wait (by time)	Microseconds
PHYSICAL_READS_SEC	Physical Reads (for each second)	Reads for each Second
PHYSICAL_READS_TXN	Physical Reads (for each transaction)	Reads for each Transaction
PHYSICAL_WRITES_SEC	Physical Writes (for each second)	Writes for each Second
PHYSICAL_WRITES_TXN	Physical Writes (for each transaction)	Writes for each Transaction
PHYSICAL_READS_DIR_SEC	Direct Physical Reads (for each second)	Reads for each Second
PHYSICAL_READS_DIR_TXN	Direct Physical Reads (for each transaction)	Reads for each Transaction
PHYSICAL_WRITES_DIR_SEC	Direct Physical Writes (for each second)	Writes for each Second
PHYSICAL_WRITES_DIR_TXN	Direct Physical Writes (for each transaction)	Writes for each Transaction
PHYSICAL_READS_LOB_SEC	Direct LOB Physical Reads (for each second)	Reads for each Second
PHYSICAL_READS_LOB_TXN	Direct LOB Physical Reads (for each transaction)	Reads for each Transaction
PHYSICAL_WRITES_LOB_SEC	Direct LOB Physical Writes (for each second)	Writes for each Second
PHYSICAL_WRITES_LOB_TXN	Direct LOB Physical Writes (for each transaction)	Writes for each Transaction
PROCESS_LIMIT_PCT	Process Limit Usage (%)	% of maximum value
PSERVICE_WAIT_SCT	Process Service Wait (by session count)	Count of sessions
PSERVICE_WAIT_TIME	Process Service Wait (by time)	Microseconds
PX_DOWNGRADED_SEC	Downgraded Parallel Operations (for each second)	Operations for each Second
PX_DOWNGRADED_25_SEC	Downgraded to 25% and more (for each second)	Operations for each Second
PX_DOWNGRADED_50_SEC	Downgraded to 50% and more (for each second)	Operations for each Second
PX_DOWNGRADED_75_SEC	Downgraded to 75% and more (for each second)	Operations for each Second
PX_DOWNGRADED_SER_SEC	Downgraded to serial (for each second)	Operations for each Second
RB_RECORDS_APPLIED_SEC	Rollback Undo Records Applied (for each second)	Records for each Second

**Table 129–3 (Cont.) List of Supported Metrics**

<b>Metric Name (Internal)</b>	<b>Metric Name (External)</b>	<b>Units</b>
RB_RECORDS_APPLIED_TXN	Rollback Undo Records Applied (for each transaction)	Records for each Transaction
REDO_ALLOCATION_HIT	Redo Log Allocation Hit	% of redo allocations
REDO_GENERATED_SEC	Redo Generated (for each second)	Redo Bytes for each Second
REDO_GENERATED_TXN	Redo Generated (for each transaction)	Redo Bytes for each Transaction
REDO_WRITES_SEC	Redo Writes (for each second)	Writes for each Second
REDO_WRITES_TXN	Redo Writes (for each transaction)	Writes for each Transaction
RECURSIVE_CALLS_SEC	Recursive Calls (for each second)	Calls for each Second
RECURSIVE_CALLS_TXN	Recursive Calls (for each transaction)	Calls for each Transaction
RESPONSE_TXN	Response (for each transaction)	Seconds for each Transaction
ROWS_PER_SORT	Rows Processed for each Sort	Rows for each Sort
SESS_LOGICAL_READS_SEC	Session Logical Reads (for each second)	Reads for each Second
SESS_LOGICAL_READS_TXN	Session Logical Reads (for each transaction)	Reads for each Transaction
SESSION_CPU_SEC	Database CPU (for each second)	Microseconds for each Second
SESSION_CPU_TXN	Database CPU (for each transaction)	Microseconds for each Transaction
SESSION_LIMIT_PCT	Session Limit Usage (%)	% of maximum value
SHARED_POOL_FREE_PCT	Shared Pool Free(%)	% of shared pool
SOFT_PARSE_PCT	Soft Parse (%)	% of all parses
SQL_SRV_RESPONSE_TIME	Service Response (for each execution)	Seconds
TABLESPACE_PCT_FULL	Tablespace space usage	% full
TABLESPACE_BYT_FREE	Tablespace bytes space usage	Kilobytes free
TOTAL_TABLE_SCANS_SEC	Total Table Scans (for each second)	Scans for each Second
TOTAL_TABLE_SCANS_TXN	Total Table Scans (for each transaction)	Scans for each Transaction
TOTAL_INDEX_SCANS_SEC	Total Index Scans (for each second)	Scans for each Second
TOTAL_INDEX_SCANS_TXN	Total Index Scans (for each transaction)	Scans for each Transaction
TOTAL_PARSSES_SEC	Total Parses (for each second)	Parses for each Second
TOTAL_PARSSES_TXN	Total Parses (for each transaction)	Parses for each Transaction
TRANSACTION_RATE	Number of Transactions (for each second)	Transactions for each Second
TXN_COMMITTED_PCT	Transactions Committed (%)	% of all transactions
USER_COMMITS_SEC	User Commits (for each second)	Commits for each Second
USER_COMMITS_TXN	User Commits (for each transaction)	Commits for each Transaction
USER_ROLLBACKS_SEC	User Rollbacks (for each second)	Rollbacks for each Second



**Table 129-3 (Cont.) List of Supported Metrics**

<b>Metric Name (Internal)</b>	<b>Metric Name (External)</b>	<b>Units</b>
USER_ROLLBACKS_TXN	User Rollbacks (for each transaction)	Rollbacks for each Transaction
USER_CALLS_SEC	User Calls (for each second)	Calls for each Second
USER_CALLS_TXN	User Calls (for each transaction)	Calls for each Transaction
USER_CALLS_PCT	User Calls (%)	% of all calls
USER_LIMIT_PCT	User Limit Usage (%)	% of maximum value
WCR_AVG_IO_LAT	Average IO response time (for a WRC client)	Milliseconds
WCR_PCPU	Percentage of replay threads on CPU (for a WRC client)	% of total replay threads
WCR_PIO	Percentage of replay threads doing IOs (for a WRC client)	% of total replay threads

## Summary of DBMS\_SERVER\_ALERT Subprograms

**Table 129–4 DBMS\_SERVER\_ALERT Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">EXPAND_MESSAGE</a> Function on page 129-13	Expands alert messages
<a href="#">GET_THRESHOLD</a> Procedure on page 129-14	Gets the current threshold settings for a specified metric
<a href="#">SET_THRESHOLD</a> Procedure on page 129-15	Sets the warning and critical thresholds for a specified metric

## EXPAND\_MESSAGE Function

This function expands alert messages.

### Syntax

```
DBMS_SERVER_ALERT.EXPAND_MESSAGE (
  user_language      IN  VARCHAR2,
  message_id        IN  NUMBER,
  argument_1        IN  VARCHAR2,
  argument_2        IN  VARCHAR2,
  argument_3        IN  VARCHAR2,
  argument_4        IN  VARCHAR2,
  argument_5        IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 129–5** *EXPAND\_MESSAGE Function Parameters*

Parameter	Description
user_language	The language of the current session.
message_id	Id of the alert message
argument_1	The first argument in the alert message.
argument_2	The second argument in the alert message.
argument_3	The third argument in the alert message.
argument_4	The fourth argument in the alert message.
argument_5	The fifth argument in the alert message.

## GET\_THRESHOLD Procedure

This procedure gets the current threshold settings for the specified metric.

### Syntax

```
DBMS_SERVER_ALERT.GET_THRESHOLD (
  metrics_id           IN  BINARY_INTEGER,
  warning_operator     OUT BINARY_INTEGER,
  warning_value        OUT VARCHAR2,
  critical_operator    OUT BINARY_INTEGER,
  critical_value       OUT VARCHAR2,
  observation_period   OUT BINARY_INTEGER,
  consecutive_occurrences OUT BINARY_INTEGER,
  instance_name        IN  VARCHAR2,
  object_type          IN  BINARY_INTEGER,
  object_name          IN  VARCHAR2);
```

### Parameters

**Table 129–6** GET\_THRESHOLD Procedure Parameters

Parameter	Description
metrics_id	The internal name of the metric. See <a href="#">"Supported Metrics"</a> on page 129-6.
warning_operator	The operator for the comparing the actual value with the warning threshold.
warning_value	The warning threshold value.
critical_operator	The operator for the comparing the actual value with the critical threshold.
critical_value	The critical threshold value.
observation_period	The period at which the metric values are computed and verified against the threshold setting.
consecutive_occurrences	The number of observation periods the metric value should violate the threshold value before the alert is issued.
instance_name	The name of the instance for which the threshold is set. This is NULL for database-wide alerts. In cases in which this parameter is not NULL, this should be set to one of the INSTANCE_NAME values found in the GV\$INSTANCE View.
object_type	Either OBJECT_TYPE_SYSTEM or OBJECT_TYPE_SERVICE.
object_name	The name of the object.

### Usage Notes

Note that this subprogram does not check if the value of the instance\_name parameter is meaningful or valid.

## SET\_THRESHOLD Procedure

This procedure sets the warning and critical thresholds for a specified metric.

### Syntax

```
DBMS_SERVER_ALERT.SET_THRESHOLD (
    metrics_id           IN   BINARY_INTEGER,
    warning_operator     IN   BINARY_INTEGER,
    warning_value        IN   VARCHAR2,
    critical_operator     IN   BINARY_INTEGER,
    critical_value        IN   VARCHAR2,
    observation_period   IN   BINARY_INTEGER,
    consecutive_occurrences IN BINARY_INTEGER,
    instance_name        IN   VARCHAR2,
    object_type          IN   BINARY_INTEGER,
    object_name          IN   VARCHAR2);
```

### Parameters

**Table 129-7 SET\_THRESHOLD Procedure Parameters**

Parameter	Description
metrics_id	The internal name of the metric. See <a href="#">"Supported Metrics"</a> on page 129-6.
warning_operator	The operator for the comparing the actual value with the warning threshold (such as OPERATOR_GE). See <a href="#">"Relational Operators"</a> on page 129-5.
warning_value	The warning threshold value. This is NULL if no warning threshold is set. A list of values may be specified for OPERATOR_CONTAINS.
critical_operator	The operator for the comparing the actual value with the critical threshold. See <a href="#">"Relational Operators"</a> on page 129-5.
critical_value	The critical threshold value. This is NULL if not set. A list of values may be specified for OPERATOR_CONTAINS.
observation_period	The period at which the metric values are computed and verified against the threshold setting. The valid range is 1 to 60 minutes.
consecutive_occurrences	The number of observation periods the metric value should violate the threshold value before the alert is issued.
instance_name	The name of the instance for which the threshold is set. This is NULL for database-wide alerts.
object_type	See <a href="#">"Object Types"</a> on page 129-4.
object_name	The name of the object. This is NULL for SYSTEM.

### Usage Notes

Note that this subprogram does not check if the value of the `instance_name` parameter is meaningful or valid. Passing a name that does not identify a valid instance will result in a threshold that is not used by any by any instance although the threshold setting will be visible in the `DBA_THRESHOLDS` view. The exception is the lower-case string 'database\_wide' which is semantically equivalent to passing NULL for the instance name, the latter being the preferred usage.



The `DBMS_SERVICE` package lets you create, delete, activate, and deactivate services for a single instance.

The chapter contains the following topics:

- [Using DBMS\\_SERVICE](#)
  - Overview
  - Security Model
  - Constants
  - Exceptions
- [Summary of DBMS\\_SERVICE Subprograms](#)

**See Also:** *Oracle Real Application Clusters Administration and Deployment Guide* for administering services in Oracle Real Application Clusters.

---

## Using DBMS\_SERVICE

This section contains topics which relate to using the DBMS\_SERVICE package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Exceptions](#)



## Overview

DBMS\_SERVICE supports the management of services in the RDBMS for the purposes of workload measurement, management, prioritization, and XA/and distributed transaction management.

Oracle Real Application Clusters (Oracle RAC) has a functionality to manage service names across instances. This package allows the creation, deletion, starting and stopping of services in both Oracle RAC and a single instance. Additionally, it provides the ability to disconnect all sessions which connect to the instance with a service name when Oracle RAC removes that service name from the instance.

**See Also:** For more information about Oracle Real Application Clusters, *Oracle Real Application Clusters Administration and Deployment Guide*.

## Security Model

### Privileges

The client using this package must have the `ALTER SYSTEM` execution privilege and the `V$SESSION` table read privilege.

### Schemas

This package must be installed under `SYS` schema.

### Roles

The `EXECUTE` privilege of the package is granted to the `DBA` role only.

## Deprecated Subprograms

For Oracle Real Applications Clusters (Oracle RAC) databases, the following DBMS\_SERVICE procedures are deprecated in 11.2, and `srvctl` used instead:

- [CREATE\\_SERVICE Procedure](#)
- [MODIFY\\_SERVICE Procedure](#)
- [START\\_SERVICE Procedure](#)
- [STOP\\_SERVICE Procedure](#)

The same is true for single-instance databases managed by Oracle Restart or Oracle Clusterware.

This is because the service attributes are stored in CRS and those attributes overwrite those specified by DBMS\_SERVICE. The DBMS\_SERVICE procedures do not update the CRS attributes.

## Constants

The `DBMS_SERVICE` package uses the constants shown in following tables

- Constants used in calling arguments are described in [Table 130–1, "Constants used in Calling Arguments"](#)
- Constants used in connection balancing goal arguments are described in [Table 130–2, "Constants used in Connection Balancing Goal Arguments"](#)
- Constants used TAF failover attribute arguments are described in [Table 130–3, "Constants used in TAF Failover Attribute Arguments"](#)

**Table 130–1 Constants used in Calling Arguments**

Name	Type	Value	Description
GOAL_NONE	NUMBER	0	Disables Load Balancing Advisory
GOAL_SERVICE_TIME	NUMBER	1	Load Balancing Advisory is based on elapsed time for work done in the service plus available bandwidth to the service
GOAL_THROUGHPUT	NUMBER	2	Load Balancing Advisory is based on the rate that work is completed in the service plus available bandwidth to the service

**Table 130–2 Constants used in Connection Balancing Goal Arguments**

Name	Type	Value	Description
CLB_GOAL_SHORT	NUMBER	1	Connection load balancing uses Load Balancing Advisory, when Load Balancing Advisory is enabled (either <code>goal_service_time</code> or <code>goal_throughput</code> ). When <code>GOAL=NONE</code> (no load balancing advisory), connection load balancing uses an abridged advice based on CPU utilization.
CLB_GOAL_LONG	NUMBER	2	Balances the number of connections per instance using session count per service. This setting is recommended for applications with long connections such as forms. This setting can be used with Load Balancing Advisory when the connection pool is sized to accommodate gravitation within the pool itself (without adding or removing connections). The latter is the most efficient design.

**Table 130–3 Constants used in TAF Failover Attribute Arguments**

Name	Type	Value	Description
FAILOVER_METHOD_NONE	VARCHAR2	0	Server side TAF is not enabled for this service
FAILOVER_METHOD_BASIC	VARCHAR2	1	Server side TAF method is BASIC. BASIC is the only value currently supported. This means that a new connection is established at failure time. It is not possible to pre-establish a backup connection. (which is to say, <code>PRECONNECT</code> is not supported)

**Table 130–3 (Cont.) Constants used in TAF Failover Attribute Arguments**

Name	Type	Value	Description
FAILOVER_TYPE_ NONE	NUMBER		Server side TAF type is NONE
FAILOVER_TYPE_ SESSION	NUMBER		Server side TAF failover type is SESSION. At failure time, if the failover type is SESSION, TAF will re-connect to a surviving node and re-establish a vanilla database session. Customizations (for example, ALTER SESSION) must be re-executed in a failover callback.
FAILOVER_TYPE_ SELECT	NUMBER		Server side TAF failover type is SELECT
FAILOVER_RETRIES	NUMBER		Number of retries to use during a failover. Specifies the number of times for TAF to attempt the re-connect and re-authenticate pair. The value must be integral and greater than 0. The maximum value is UB4MAXVAL
FAILOVER_DELAY	NUMBER		Number of seconds delay before trying to fail over. Specifies the delay (in seconds) that TAF will incur if the re-connect / re-authentication fails. The value must be integral and greater than 0. The maximum value is UB4MAXVAL.

## Usage Notes

- If a TAF callback has been registered, then the failover retries and failover delay are ignored. If an error occurs, TAF will continue to re-attempt the connect and authentication as long as the callback returns a value of OCI\_FO\_RETRY. Any delay must be coded into the callback logic
- Server side TAF settings override client-side counterparts that might be configured in TNS connect descriptors. If TAF is not configured on the client side, then at a minimum, the failover type must be set to enable TAF. If the failover type is set on the server side, then the failover method will default to BASIC. Delay and retries are optional and may be specified independently.

## Exceptions

The following table lists the exceptions raised by `DBMS_SERVICE` package.

**Table 130–4** *DBMS\_SERVICE Exceptions*

Exception	Error Code	Description
<code>NULL_SERVICE_NAME</code>	44301	The service name argument was found to be <code>NULL</code>
<code>NULL_NETWORK_NAME</code>	44302	The network name argument was found to be <code>NULL</code>
<code>SERVICE_EXISTS</code>	44303	This service name was already in existence
<code>SERVICE_DOES_NOT_EXIST</code>	44304	The specified service was not in existence
<code>SERVICE_IN_USE</code>	44305	The specified service was running
<code>SERVICE_NAME_TOO_LONG</code>	44306	The service name was too long
<code>NETWORK_PREFIX_TOO_LONG</code>	44307	The network name, excluding the domain, was too long
<code>NOT_INITIALIZED</code>	44308	The services layer was not yet initialized
<code>GENERAL_FAILURE</code>	44309	There was an unknown failure
<code>MAX_SERVICES_EXCEEDED</code>	44310	The maximum number of services has been reached
<code>SERVICE_NOT_RUNNING</code>	44311	The specified service was not running
<code>DATABASE_CLOSED</code>	44312	The database was closed
<code>INVALID_INSTANCE</code>	44313	The instance name argument was not valid
<code>NETWORK_EXISTS</code>	44314	The network name was already in existence
<code>NULL_ATTRIBUTES</code>	44315	All attributes specified were <code>NULL</code>
<code>INVALID_ARGUMENT</code>	44316	Invalid argument supplied
<code>DATABASE_READONLY</code>	44317	The database is open read-only
<code>MAX_SN_LENGTH</code>	44318	The total length of all running service network names exceeded the maximum allowable length

---

## Summary of DBMS\_SERVICE Subprograms

**Table 130-5 DBMS\_SERVICE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CREATE_SERVICE Procedure</a> on page 130-10	Creates service
<a href="#">DELETE_SERVICE Procedure</a> on page 130-12	Deletes service
<a href="#">DISCONNECT_SESSION Procedure</a> on page 130-13	Disconnects service
<a href="#">MODIFY_SERVICE Procedure</a> on page 130-14	Modifies service
<a href="#">START_SERVICE Procedure</a> on page 130-16	Activates service
<a href="#">STOP_SERVICE Procedure</a> on page 130-17	Stops service

## CREATE\_SERVICE Procedure

This procedure creates a service name in the data dictionary. Services are also created in the data dictionary implicitly when you set the service in the `service_name` parameter or by means of the `ALTER SYSTEM SET SERVICE_NAMES` command.

---

**Note:** This procedure is deprecated in databases managed by Oracle Clusterware and Oracle Restart with Release 11.2. While the procedure remains available in the package, Oracle recommends using `srvctl` to manage services. This is because the service attributes are stored in CRS by `srvctl`, and overwrite those specified by `DBMS_SERVICE`. The `DBMS_SERVICE` procedures do not update the CRS attributes.

---



---

**Note:** The functionality associated with the `edition` argument is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```
DBMS_SERVICE.CREATE_SERVICE(
  service_name      IN VARCHAR2,
  network_name     IN VARCHAR2,
  goal              IN NUMBER DEFAULT NULL,
  dtp              IN BOOLEAN DEFAULT NULL,
  aq_ha_notifications IN BOOLEAN DEFAULT NULL,
  failover_method  IN VARCHAR2 DEFAULT NULL,
  failover_type    IN VARCHAR2 DEFAULT NULL,
  failover_retries IN NUMBER DEFAULT NULL,
  failover_delay   IN NUMBER DEFAULT NULL,
  clb_goal         IN NUMBER DEFAULT NULL,
  edition          IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 130–6 CREATE\_SERVICE Procedure Parameters**

Parameter	Description
<code>service_name</code>	Name of the service limited to 64 characters in the Data Dictionary
<code>network_name</code>	The network name of the service as used in SQLNet connect descriptors for client connections. This is limited to the NET <code>service_names</code> character set (see <i>Oracle Database Net Services Reference</i> ).
<code>goal</code>	The workload management goal directive for the service. Valid values: <ul style="list-style-type: none"> <li>■ <code>DBMS_SERVICE.GOAL_SERVICE_TIME</code></li> <li>■ <code>DBMS_SERVICE.GOAL_THROUGHPUT</code></li> <li>■ <code>DBMS_SERVICE.GOAL_NONE</code></li> </ul>
<code>dtp</code>	Declares the service to be for DTP or distributed transactions including XA transactions



**Table 130–6 (Cont.) CREATE\_SERVICE Procedure Parameters**

Parameter	Description
aq_ha_notifications	Determines whether HA events are sent through AQ for this service
failover_method	The TAF failover method for the service
failover_type	The TAF failover type for the service
failover_retries	The TAF failover retries for the service
failover_delay	The TAF failover delay for the service
clb_goal	Method used for Connection Load Balancing (see <a href="#">Table 130–2, "Constants used in Connection Balancing Goal Arguments"</a> )
edition	<p>If this argument has a non-NULL value, this provides the initial session edition for subsequent database connections using this service that do not specify an edition. If no value is specified, this argument will have no effect.</p> <p>During service creation or modification, no validation is performed on this parameter.</p> <p>At connection time, if the connecting user does not have USE privilege on the edition, or the edition does not exist, this raises the error ORA-38802 (edition does not exist).</p>

## Examples

```
DBMS_SERVICE.CREATE_SERVICE('ernie.us.oracle.com', 'ernie.us.oracle.com');
```

## DELETE\_SERVICE Procedure

This procedure deletes a service from the data dictionary.

---

---

**Note:** This procedure is deprecated in databases managed by Oracle Clusterware and Oracle Restart with Release 11.2. While the procedure remains available in the package, Oracle recommends using `srvctl` to manage services. This is because the service attributes are stored in CRS by `srvctl`, and overwrite those specified by `DBMS_SERVICE`. Unless the service is also deleted with `srvctl`, it will be re-created in the database when CRS next starts the service.

---

---

### Syntax

```
DBMS_SERVICE.DELETE_SERVICE(  
    service_name    IN VARCHAR2);
```

### Parameters

**Table 130–7** *DELETE\_SERVICE Procedure Parameters*

Parameter	Description
<code>service_name</code>	Name of the service limited to 64 characters in the Data Dictionary

### Examples

```
DBMS_SERVICE.DELETE_SERVICE('ernie.us.oracle.com');
```

## DISCONNECT\_SESSION Procedure

This procedure disconnects sessions with the named service at the current instance.

### Syntax

```
DBMS_SERVICE.DISCONNECT_SESSION(
  service_name      IN VARCHAR2,
  disconnect_option IN NUMBER DEFAULT POST_TRANSACTION);
```

### Parameters

**Table 130–8** DISCONNECT\_SESSION Procedure Parameters

Parameter	Description
service_name	Name of the service limited to 64 characters in the Data Dictionary
disconnect_option	There are two options, package constants expressed as NUMBER: <ul style="list-style-type: none"> <li>■ POST_TRANSACTION = 0 : the session will disconnect after the current transaction commits or rolls back</li> <li>■ IMMEDIATE = 1 : sessions will be disconnected immediately</li> </ul> <p><b>Note:</b> IMMEDIATE or POST_TRANSACTION will be automatically translated as 1 and 0 respectively. However, passing-in a string literal (quoted using either the ' or " characters, such as "IMMEDIATE" or 'POST_TRANSACTION') will raise an error.</p>

### Usage Notes

- This procedure can be used in the context of a single instance as well as with Oracle Real Application Clusters.
- This subprogram does not return until all corresponding sessions are disconnected. Therefore, use the DBMS\_JOB package or put the SQL session in background if the caller does not want to wait for all corresponding sessions to be disconnected.

### Examples

This disconnects sessions with service\_name 'ernie.us.oracle.com'.

```
DBMS_SERVICE.DISCONNECT_SESSION('ernie.us.oracle.com');
```

## MODIFY\_SERVICE Procedure

This procedure modifies an existing service.

---

**Note:** This procedure is deprecated in databases managed by Oracle Clusterware and Oracle Restart with Release 11.2. While the procedure remains available in the package, Oracle recommends using `srvctl` to manage services. This is because the service attributes are stored in CRS by `srvctl`, and overwrite those specified by `DBMS_SERVICE`. The `DBMS_SERVICE` procedures do not update the CRS attributes and therefore are not persistent, though they do take effect immediately, until the service is next started with `srvctl`.

---



---

**Note:** The functionality associated with the `edition` and `modify_edition` arguments is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```
DBMS_SERVICE.MODIFY_SERVICE(
  service_name      IN VARCHAR2,
  goal              IN NUMBER DEFAULT NULL,
  dtp              IN BOOLEAN DEFAULT NULL,
  aq_ha_notifications IN BOOLEAN DEFAULT NULL,
  failover_method  IN VARCHAR2 DEFAULT NULL,
  failover_type    IN VARCHAR2 DEFAULT NULL,
  failover_retries IN NUMBER DEFAULT NULL,
  failover_delay   IN NUMBER DEFAULT NULL,
  clb_goal         IN NUMBER DEFAULT NULL,
  edition          IN VARCHAR2 DEFAULT NULL,
  modify_edition   IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 130–9 MODIFY\_SERVICE Procedure Parameters**

Parameter	Description
<code>service_name</code>	Name of the service limited to 64 characters in the Data Dictionary
<code>goal</code>	The workload management goal directive for the service. Valid values: <ul style="list-style-type: none"> <li>▪ <code>DBMS_SERVICE.GOAL_SERVICE_TIME</code></li> <li>▪ <code>DBMS_SERVICE.GOAL_THROUGHPUT</code></li> <li>▪ <code>DBMS_SERVICE.GOAL_NONE</code></li> </ul>
<code>dtp</code>	Declares the service to be for DTP or distributed transactions including XA transactions
<code>aq_ha_notifications</code>	Determines whether HA events are sent through AQ for this service
<code>failover_method</code>	The TAF failover method for the service
<code>failover_type</code>	The TAF failover type for the service

**Table 130–9 (Cont.) MODIFY\_SERVICE Procedure Parameters**

Parameter	Description
failover_retries	The TAF failover retries for the service
failover_delay	The TAF failover delay for the service
clb_goal	Method used for Connection Load Balancing (see <a href="#">Table 130–2, "Constants used in Connection Balancing Goal Arguments"</a> )
edition	<p>If this argument has a non-NULL value, this provides the initial session edition for subsequent database connections using this service that do not specify an edition. If no value is specified, this argument will have no effect.</p> <p>During service creation or modification, no validation is performed on this parameter.</p> <p>At connection time, if the connecting user does not have <code>USE</code> privilege on the edition, or the edition does not exist, this raises the error <code>ORA-38802</code> (edition does not exist).</p>
modify_edition	If <code>TRUE</code> , the edition service attribute is updated to use the edition argument value. If <code>FALSE</code> or <code>NULL</code> , the edition attribute will not be updated.

## Usage Notes

- If you are using Clustered Managed Services with Oracle Clusterware, or using Oracle Restart with your single instance database, you must modify services using the `srvctl` command rather than `DBMS_SERVICE`. When the service is started by Oracle Clusterware or Oracle Restart, the service will be modified in the database to match the resource defined to either Oracle Clusterware or Oracle Restart. Any changes made with `DBMS_SERVICE` will be lost unless they are also made with the corresponding `srvctl` command. Service attribute modifications take effect immediately starting with 11.2.0.2 when the service is started or modified by `srvctl`.
- Although the edition attribute can be modified while the service is up and running, it may not be safe to do so. Users must proceed with caution since this will cause new connections to be connected at the new edition, while existing connection will not be affected. This can in turn cause mid-tier operations to connect to the wrong edition.

## START\_SERVICE Procedure

This procedure starts a service. This procedure alters the `service_name` IOP to contain this `service_name`. In Oracle RAC, implementing this option will act on the instance specified.

---



---

**Note:** This procedure is deprecated in databases managed by Oracle Clusterware and Oracle Restart with Release 11.2. While the procedure remains available in the package, Oracle recommends using `srvctl` to manage services.

---



---

### Syntax

```
DBMS_SERVICE.START_SERVICE(
    service_name IN VARCHAR2,
    instance_name IN VARCHAR2);
```

### Parameters

**Table 130–10** *START\_SERVICE Procedure Parameters*

Parameter	Description
<code>service_name</code>	Name of the service limited to 64 characters in the Data Dictionary
<code>instance_name</code>	Name of the instance where the service must be activated (optional). The instance on which to start the service. <code>NULL</code> results in starting of the service on the local instance. In single instance this can only be the current instance or <code>NULL</code> . Specify <code>DBMS_SERVICE.ALL_INSTANCES</code> to start the service on all configured instances.

### Examples

```
DBMS_SERVICE.START_SERVICE('ernie.us.oracle.com');
```

## STOP\_SERVICE Procedure

This procedure stops a service, altering the `service_name` IOP to remove this `service_name`.

---



---

**Note:** This procedure is deprecated in databases managed by Oracle Clusterware and Oracle Restart with Release 11.2. While the procedure remains available in the package, Oracle recommends using `srvctl` to manage services.

---



---

### Syntax

```
DBMS_SERVICE.STOP_SERVICE(
  service_name  IN VARCHAR2,
  instance_name IN VARCHAR2);
```

### Parameters

**Table 130–11 STOP\_SERVICE Procedure Parameters**

Parameter	Description
<code>service_name</code>	Name of the service limited to 64 characters in the Data Dictionary
<code>instance_name</code>	Name of the instance where the service must be stopped (optional). The instance on which to stop the service. <code>NULL</code> results in stopping of the service locally. In single instance this can only be the current instance or <code>NULL</code> . The default in Oracle RAC and exclusive case is <code>NULL</code> . Specify <code>DBMS_SERVICE.ALL_INSTANCES</code> to stop the service on all configured instances.

### Examples

```
DBMS_SERVICE.STOP_SERVICE('ernie.us.oracle.com');
```





This package provides access to SQL `ALTER SESSION` and `SET ROLE` statements, and other session information, from PL/SQL. You can use `DBMS_SESSION` to set preferences and security levels.

This chapter contains the following topics:

- [Using DBMS\\_SESSION](#)
  - Security Model
  - Operational Notes
- [Data Structures](#)
- [Summary of DBMS\\_SESSION Subprograms](#)

---

## Using DBMS\_SESSION

- [Security Model](#)
- [Operational Notes](#)

## Security Model

This package runs with the privileges of the calling user, rather than the package owner `SYS`.

## Operational Notes

You should not attempt to turn `close_cached_open_cursors` on or off.

## Data Structures

---

The DBMS\_SESSION package defines TABLE types.

### Table Types

- [INTEGER\\_ARRAY Table Type](#)
- [LNAME\\_ARRAY Table Type](#)

## INTEGER\_ARRAY Table Type

A table type of `BINARY_INTEGER`.

### Syntax

```
TYPE integer_array IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

## LNAME\_ARRAY Table Type

A table type of VARCHAR2.

### Syntax

```
TYPE lname_array IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

---

## Summary of DBMS\_SESSION Subprograms

**Table 131–1 DBMS\_SESSION Package Subprograms**

Subprogram	Description
<a href="#">CLEAR_ALL_CONTEXT Procedure</a> on page 131-9	Clears all context information
<a href="#">CLEAR_CONTEXT Procedure</a> on page 131-10	Clears the context
<a href="#">CLEAR_IDENTIFIER Procedure</a> on page 131-11	Clears the identifier
<a href="#">CLOSE_DATABASE_LINK Procedure</a> on page 131-12	Closes database link
<a href="#">FREE_UNUSED_USER_MEMORY Procedure</a> on page 131-13	Lets you reclaim unused memory after performing operations requiring large amounts of memory
<a href="#">GET_PACKAGE_MEMORY_UTILIZATION Procedure</a> on page 131-15	Describes static package memory usage
<a href="#">IS_ROLE_ENABLED Function</a> on page 131-16	Determines if the named role is enabled for the session.
<a href="#">IS_SESSION_ALIVE Function</a> on page 131-17	Determines if the specified session is active
<a href="#">LIST_CONTEXT Procedures</a> on page 131-18	Returns a list of active namespace and context for the current session
<a href="#">MODIFY_PACKAGE_STATE Procedure</a> on page 131-19	Used to perform various actions (as specified by the <code>action_flags</code> parameter) on the session state of all PL/SQL program units active in the session
<a href="#">SESSION_TRACE_DISABLE Procedure</a> on page 131-23	Resets the session-level SQL trace for the session from which it was called.
<a href="#">SESSION_TRACE_ENABLE Procedure</a> on page 131-24	Enables session-level SQL trace for the invoking session
<a href="#">RESET_PACKAGE Procedure</a> on page 131-25	De-instantiates all packages in the session
<a href="#">SET_CONTEXT Procedure</a> on page 131-27	Sets or resets the value of a context attribute
<a href="#">SET_EDITION_DEFERRED Procedure</a> on page 131-29	Requests a switch to the specified edition
<a href="#">SET_IDENTIFIER Procedure</a> on page 131-30	Sets the identifier
<a href="#">SET-NLS Procedure</a> on page 131-31	Sets Globalization Support (NLS)
<a href="#">SET_ROLE Procedure</a> on page 131-32	Sets role
<a href="#">SET_SQL_TRACE Procedure</a> on page 131-33	Turns tracing on or off
<a href="#">SWITCH_CURRENT_CONSUMER_GROUP Procedure</a> on page 131-34	Facilitates changing the current resource consumer group of a user's current session
<a href="#">UNIQUE_SESSION_ID Function</a> on page 131-36	Returns an identifier that is unique for all sessions currently connected to this database



## CLEAR\_ALL\_CONTEXT Procedure

### Syntax

```
DBMS_SESSION.CLEAR_ALL_CONTEXT  
namespace          VARCHAR2);
```

### Parameters

**Table 131–2 CLEAR\_ALL\_CONTEXT Procedure Parameters**

Parameter	Description
namespace	The namespace where the application context information is to be cleared. Required.

### Usage Notes

- This procedure must be invoked directly or indirectly by the trusted package.
- Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

## CLEAR\_CONTEXT Procedure

### Syntax

```
DBMS_SESSION.CLEAR_CONTEXT
  namespace      VARCHAR2,
  client_identifier VARCHAR2
  attribute      VARCHAR2);
```

### Parameters

**Table 131–3 CLEAR\_CONTEXT Procedure Parameters**

Parameter	Description
namespace	<p>Namespace in which the application context is to be cleared. Required.</p> <p>For a session-local context, namespace must be specified. If namespace is defined as Session Local Context, then client_identifier is optional since it is only associated with a globally accessed context.</p> <p>For a globally accessed context, namespace must be specified. NULL is a valid value for client_identifier because a session with no identifier set can see a context that looks like the (namespace, attribute, value, username, null) set using SET_CONTEXT.</p>
client_identifier	Applies to a global context and is optional for other types of contexts; 64-byte maximum
attribute	Specific attribute in the namespace to be cleared. Optional. the default is NULL. If you specify attribute as NULL, then (namespace, attribute, value) for that namespace are cleared from the session. If attribute is not specified, then all context information that has the namespace and client_identifier arguments is cleared.

### Usage Notes

- This procedure must be invoked directly or indirectly by the trusted package.
- Any changes in context value are reflected immediately and subsequent calls to access the value through SYS\_CONTEXT return the most recent value.

## **CLEAR\_IDENTIFIER Procedure**

This procedure removes the `set_client_id` in the session.

### **Syntax**

```
DBMS_SESSION.CLEAR_IDENTIFIER;
```

### **Usage Notes**

This procedure is executable by public.

## CLOSE\_DATABASE\_LINK Procedure

This procedure closes an open database link. It is equivalent to the following SQL statement:

```
ALTER SESSION CLOSE DATABASE LINK <name>
```

### Syntax

```
DBMS_SESSION.CLOSE_DATABASE_LINK (  
    dblink VARCHAR2);
```

### Parameters

**Table 131–4** *CLOSE\_DATABASE\_LINK Procedure Parameters*

Parameter	Description
dblink	Name of the database link to close

## FREE\_UNUSED\_USER\_MEMORY Procedure

This procedure reclaims unused memory after performing operations requiring large amounts of memory (more than 100K).

Examples of operations that use large amounts of memory include:

- Large sorting where entire `sort_area_size` is used and `sort_area_size` is hundreds of KB.
- Compiling large PL/SQL packages, procedures, or functions.
- Storing hundreds of KB of data within PL/SQL indexed tables.

You can monitor user memory by tracking the statistics "session UGA memory" and "session PGA memory" in the `v$sesstat` or `v$statname` fixed views. Monitoring these statistics also shows how much memory this procedure has freed.

---



---

**Note:** This procedure should only be used in cases where memory is at a premium. It should be used infrequently and judiciously.

---



---

### Syntax

```
DBMS_SESSION.FREE_UNUSED_USER_MEMORY;
```

### Return Values

The behavior of this procedure depends upon the configuration of the server operating on behalf of the client:

- **Dedicated server:** This returns unused PGA memory and session memory to the operating system. Session memory is allocated from the PGA in this configuration.
- **Shared server:** This returns unused session memory to the `shared_pool`. Session memory is allocated from the `shared_pool` in this configuration.

### Usage Notes

In order to free memory using this procedure, the memory must not be in use.

After an operation allocates memory, only the same type of operation can reuse the allocated memory. For example, after memory is allocated for sort, even if the sort is complete and the memory is no longer in use, only another sort can reuse the sort-allocated memory. For both sort and compilation, after the operation is complete, the memory is no longer in use, and the user can call this procedure to free the unused memory.

An indexed table implicitly allocates memory to store values assigned to the indexed table's elements. Thus, the more elements in an indexed table, the more memory the RDBMS allocates to the indexed table. As long as there are elements within the indexed table, the memory associated with an indexed table is in use.

The scope of indexed tables determines how long their memory is in use. Indexed tables declared globally are indexed tables declared in packages or package bodies. They allocate memory from session memory. For an indexed table declared globally, the memory remains in use for the lifetime of a user's login (lifetime of a user's session), and is freed after the user disconnects from ORACLE.

Indexed tables declared locally are indexed tables declared within functions, procedures, or anonymous blocks. These indexed tables allocate memory from PGA

memory. For an indexed table declared locally, the memory remains in use for as long as the user is still running the procedure, function, or anonymous block in which the indexed table is declared. After the procedure, function, or anonymous block is finished running, the memory is then available for other locally declared indexed tables to use (in other words, the memory is no longer in use).

Assigning an uninitialized, "empty" indexed table to an existing index table is a method to explicitly re-initialize the indexed table and the memory associated with the indexed table. After this operation, the memory associated with the indexed table is no longer in use, making it available to be freed by calling this procedure. This method is particularly useful on indexed tables declared globally which can grow during the lifetime of a user's session, as long as the user no longer needs the contents of the indexed table.

The memory rules associated with an indexed table's scope still apply; this method and this procedure, however, allow users to intervene and to explicitly free the memory associated with an indexed table.

## Examples

The following PL/SQL illustrates the method and the use of procedure `FREE_UNUSED_USER_MEMORY`.

```
CREATE PACKAGE foobar
  type number_idx_tbl is table of number indexed by binary_integer;

  store1_table number_idx_tbl;    -- PL/SQL indexed table
  store2_table number_idx_tbl;    -- PL/SQL indexed table
  store3_table number_idx_tbl;    -- PL/SQL indexed table
  ...
END;                                -- end of foobar

DECLARE
  ...
  empty_table number_idx_tbl;    -- uninitialized ("empty") version
BEGIN
  FOR i in 1..1000000 loop
    store1_table(i) := i;        -- load data
  END LOOP;
  ...
  store1_table := empty_table;    -- "truncate" the indexed table
  ...
  -
  dbms_session.free_unused_user_memory; -- give memory back to system

  store1_table(1) := 100;        -- index tables still declared;
  store2_table(2) := 200;        -- but truncated.
  ...
END;
```

## GET\_PACKAGE\_MEMORY\_UTILIZATION Procedure

This procedure describes static package memory usage.

The output collections describe memory usage in each instantiated package. Each package is described by its owner name, package name, used memory amount, and unused allocated memory amount. The amount of unused memory is greater than zero because of memory fragmentation and also because once used free memory chunks initially go to a free list owned by the package memory heap. They are released back to the parent heap only when the [FREE\\_UNUSED\\_USER\\_MEMORY Procedure](#) is invoked.

### Syntax

```
DBMS_SESSION.GET_PACKAGE_MEMORY_UTILIZATION (
    owner_names      OUT NOCOPY LNAME_ARRAY,
    unit_names       OUT NOCOPY LNAME_ARRAY,
    unit_types       OUT NOCOPY INTEGER_ARRAY,
    used_amounts     OUT NOCOPY INTEGER_ARRAY,
    free_amounts     OUT NOCOPY INTEGER_ARRAY);
```

### Parameters

**Table 131-5** GET\_PACKAGE\_MEMORY\_UTILIZATION Function Parameters

Parameter	Description
owner_name	Owner of package
unit_name	Name of package
unit_types	Value of the type# columns of the dictionary table obj\$
used_amounts	Amount of allocated memory specified in bytes
free_amounts	Amount of available memory specified in bytes

## IS\_ROLE\_ENABLED Function

This function determines if the named role is enabled for this session.

### Syntax

```
DBMS_SESSION.IS_ROLE_ENABLED (  
    rolename    VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 131-6 IS\_ROLE\_ENABLED Function Parameters**

Parameter	Description
rolename	Name of the role.\

### Return Values

**Table 131-7 IS\_ROLE\_ENABLED Function Return Values**

Return	Description
is_role_enabled	TRUE or FALSE, depending on whether the role is enabled



## IS\_SESSION\_ALIVE Function

This function determines if the specified session is active.

### Syntax

```
DBMS_SESSION.IS_SESSION_ALIVE (  
    uniqueid VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 131–8 IS\_SESSION\_ALIVE Function Parameters**

Parameter	Description
uniqueid	Unique ID of the session: This is the same one as returned by UNIQUE_SESSION_ID.

### Return Values

**Table 131–9 IS\_SESSION\_ALIVE Function Return Values**

Return	Description
is_session_alive	TRUE or FALSE, depending on whether the session is active

## LIST\_CONTEXT Procedures

This procedure returns a list of active namespaces and contexts for the current session.

### Syntax

```
TYPE AppCtxRecTyp IS RECORD (
    namespace VARCHAR2(30),
    attribute VARCHAR2(30),
    value      VARCHAR2(256));

TYPE AppCtxTabTyp IS TABLE OF AppCtxRecTyp INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT (
    list OUT AppCtxTabTyp,
    size OUT NUMBER);
```

### Parameters

**Table 131–10 LIST\_CONTEXT Procedure Parameters**

Parameter	Description
list	Buffer to store a list of application context set in the current session

### Return Values

**Table 131–11 LIST\_CONTEXT Procedure Return Values**

Return	Description
list	A list of (namespace, attribute, values) set in current session
size	Returns the number of entries in the buffer returned

### Usage Notes

The context information in the list appears as a series of <namespace> <attribute> <value>. Because list is a table type variable, its size is dynamically adjusted to the size of returned list.

## MODIFY\_PACKAGE\_STATE Procedure

This procedure is used to perform various actions (as specified by the `action_flags` parameter) on the session state of all PL/SQL program units active in the session. This takes effect after the PL/SQL call that made the current invocation finishes running. The procedure uses the DBMS\_SESSION constants listed in [Table 131–13](#).

### Syntax

```
DBMS_SESSION.MODIFY_PACKAGE_STATE (
    action_flags IN PLS_INTEGER);
```

### Parameters

**Table 131–12** *MODIFY\_PACKAGE\_STATE Procedure Parameters*

Parameter	Description
<code>action_flags</code>	<p>Bit flags that determine the action taken on PL/SQL program units:</p> <p><code>DBMS_SESSION.FREE_ALL_RESOURCES</code> (or 1)—frees all memory associated with each of the previously run PL/SQL programs from the session. Clears the current values of any package globals and closes cached cursors. On subsequent use, the PL/SQL program units are reinstated and package globals are reinitialized. Invoking <code>MODIFY_PACKAGE_STATE</code> with the <code>DBMS_SESSION.FREE_ALL_RESOURCES</code> parameter provides functionality identical to the <code>DBMS_SESSION.RESET_PACKAGE()</code> interface.</p> <p><code>DBMS_SESSION.REINITIALIZE</code> (or 2)—reinitializes packages without actually being freed and recreated from scratch. Instead the package memory is reused. In terms of program semantics, the <code>DBMS_SESSION.REINITIALIZE</code> flag is similar to the <code>DBMS_SESSION.FREE_ALL_RESOURCES</code> flag in that both have the effect of reinitializing all packages.</p> <p>However, <code>DBMS_SESSION.REINITIALIZE</code> should exhibit better performance than the <code>DBMS_SESSION.FREE_ALL_RESOURCES</code> option because:</p> <ul style="list-style-type: none"> <li>■ Packages are reinitialized without actually being freed and recreated from scratch. Instead the package memory gets reused.</li> <li>■ Any open cursors are closed, semantically speaking. However, the cursor resource is not actually freed. It is simply returned to the PL/SQL cursor cache. The cursor cache is not flushed. Hence, cursors corresponding to frequently accessed static SQL in PL/SQL remains cached in the PL/SQL cursor cache and the application does not incur the overhead of opening, parsing, and closing a new cursor for those statements on subsequent use.</li> <li>■ The session memory for PL/SQL modules without global state (such as types, stored-procedures) are not freed and recreated.</li> </ul>

## Usage Notes

See the parameter descriptions in [Table 131–14](#) for the differences between the flags and why `DBMS_SESSION.REINITIALIZE` exhibits better performance than `DBMS_SESSION.FREE_ALL_RESOURCES`.

**Table 131–13 Action\_flags Constants for MODIFY\_PACKAGE\_STATE**

Constant	Description
<code>FREE_ALL_RESOURCES</code>	<code>PLS_INTEGER:= 1</code>
<code>REINITIALIZE</code>	<code>PLS_INTEGER:= 2</code>

- Reinitialization refers to the process of resetting all package variables to their initial values and running the initialization block (if any) in the package bodies. Consider the package:

```

package P is
  n number;
  m number := P2.foo;
  d date := SYSDATE;
  cursor c is select * from emp;
  procedure bar;
end P;
/
package body P is
  v varchar2(20) := 'hello';
  procedure bar is
  begin
    ...
  end;
  procedure init_pkg is
  begin
    ...
  end;
begin
  -- initialization block
  init_pkg;
  ...
  ...
end P;
/

```

For the package P, reinitialization involves:

- Setting `P.n` to `NULL`
- Invoking function `P2.foo` and setting `P.m` to the value returned from `P2.foo`
- Setting `P.d` to the return value of `SYSDATE` built-in
- Closing cursor `P.c` if it was previously opened
- Setting `P.v` to 'hello'
- Running the initialization block in the package body
- The reinitialization for a package is done only if the package is actually referenced subsequently. Furthermore, the packages are reinitialized in the order in which they are referenced subsequently.

- When using `FREE_ALL_RESOURCES` or `REINITIALIZE`, make sure that resetting package variable values does not affect the application.
- Because `DBMS_SESSION.REINITIALIZE` does not actually cause all the package state to be freed, in some situations, the application could use significantly more session memory than if the `FREE_ALL_RESOURCES` flag or the `RESET_PACKAGE` procedure had been used. For instance, after performing `DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE)`, if the application does not refer to many of the packages that were previously referenced, then the session memory for those packages remains until the end of the session (or until `DBMS_SESSION.RESET_PACKAGE` is called).
- Because the client-side PL/SQL code cannot reference remote package variables or constants, you must explicitly use the values of the constants. For example, `DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE)` does not compile on the client because it uses the constant `DBMS_SESSION.REINITIALIZE`.

Instead, use `DBMS_SESSION.MODIFY_PACKAGE_STATE(2)` on the client, because the argument is explicitly provided.

## Examples

This example illustrates the use of `DBMS_SESSION.MODIFY_PACKAGE_STATE`. Consider a package `P` with some global state (a cursor `c` and a number `cnt`). When the package is first initialized, the package variable `cnt` is 0 and the cursor `c` is `CLOSED`. Then, in the session, change the value of `cnt` to 111 and also execute an `OPEN` operation on the cursor. If you call `print_status` to display the state of the package, you see that `cnt` is 111 and that the cursor is `OPEN`. Next, call `DBMS_SESSION.MODIFY_PACKAGE_STATE`. If you print the status of the package `P` again using `print_status`, you see that `cnt` is 0 again and the cursor is `CLOSED`. If the call to `DBMS_SESSION.MODIFY_PACKAGE_STATE` had not been made, then the second `print_status` would have printed 111 and `OPEN`.

```

create or replace package P is
  cnt    number := 0;
  cursor c is select * from emp;
  procedure print_status;
end P;
/
show errors;

create or replace package body P is
  procedure print_status is
  begin
    dbms_output.put_line('P.cnt = ' || cnt);
    if c%ISOPEN then
      dbms_output.put_line('P.c is OPEN');
    else
      dbms_output.put_line('P.c is CLOSED');
    end if;
  end;
end P;
/
show errors;

SQL> set serveroutput on;
SQL> begin
  2  P.cnt := 111;
  3  open p.c;

```

```
4 P.print_status;
5 end;
6 /
P.cnt = 111
P.c is OPEN
```

PL/SQL procedure successfully completed.

```
SQL> begin
2 dbms_session.modify_package_state(dbms_session.reinitialize);
3 end;
4 /
```

PL/SQL procedure successfully completed.

```
SQL> set serveroutput on;
```

```
SQL>
```

```
SQL> begin
2 P.print_status;
3 end;
4 /
```

```
P.cnt = 0
```

```
P.c is CLOSED
```

PL/SQL procedure successfully completed.

## **SESSION\_TRACE\_DISABLE Procedure**

This procedure resets the session-level SQL trace for the session from which it was called. Client ID and service/module/action traces are not affected.

### **Syntax**

```
DBMS_SESSION.SESSION_TRACE_DISABLE;
```

## SESSION\_TRACE\_ENABLE Procedure

This procedure enables session-level SQL trace for the invoking session. Invoking this procedure results in SQL tracing of every SQL statement issued by the session.

### Syntax

```
DBMS_SESSION.SESSION_TRACE_ENABLE(  
    waits      IN   BOOLEAN DEFAULT TRUE,  
    binds      IN   BOOLEAN DEFAULT FALSE,  
    plan_stat  IN   VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 131–14** SESSION\_TRACE\_ENABLE Procedure Parameters

Parameter	Description
waits	Specifies if wait information is to be traced
binds	Specifies if bind information is to be traced
plan_stat	Frequency at which we dump row source statistics. Value should be 'NEVER', 'FIRST_EXECUTION' (equivalent to NULL) or 'ALL_EXECUTIONS'.



## RESET\_PACKAGE Procedure

This procedure de-instantiates all packages in this session. It frees the package state.

---

**Note:** See "[SESSION\\_TRACE\\_ENABLE Procedure](#)" on page 131-24. The `MODIFY_PACKAGE_STATE` interface, introduced in Oracle9i, provides an equivalent of the `RESET_PACKAGE` capability. It is an efficient, lighter-weight variant for reinitializing the state of all PL/SQL packages in the session.

---

Memory used for caching the execution state is associated with all PL/SQL functions, procedures, and packages that were run in a session.

For packages, this collection of memory holds the current values of package variables and controls the cache of cursors opened by the respective PL/SQL programs. A call to `RESET_PACKAGE` frees the memory associated with each of the previously run PL/SQL programs from the session, and, consequently, clears the current values of any package globals and closes any cached cursors.

`RESET_PACKAGE` can also be used to reliably restart a failed program in a session. If a program containing package variables fails, then it is hard to determine which variables need to be reinitialized. `RESET_PACKAGE` guarantees that all package variables are reset to their initial values.

### Syntax

```
DBMS_SESSION.RESET_PACKAGE;
```

### Usage Notes

Because the amount of memory consumed by all executed PL/SQL can become large, you might use `RESET_PACKAGE` to trim down the session memory footprint at certain points in your database application. However, make sure that resetting package variable values does not affect the application. Also, remember that later execution of programs that have lost their cached memory and cursors will perform slower, because they need to re-create the freed memory and cursors.

`RESET_PACKAGE` does not free the memory, cursors, and package variables immediately when called.

---

**Note:** `RESET_PACKAGE` only frees the memory, cursors, and package variables after the PL/SQL call that made the invocation finishes running.

---

For example, PL/SQL procedure P1 calls PL/SQL procedure P2, and P2 calls `RESET_PACKAGE`. The `RESET_PACKAGE` effects do not occur until procedure P1 finishes execution (the PL/SQL call ends).

### Examples

This SQL\*Plus script runs a large program with many PL/SQL program units that may or may not use global variables, but it doesn't need them beyond this execution:

```
EXECUTE large_plsql_program1;
```

To free up PL/SQL cached session memory:

```
EXECUTE DBMS_SESSION.RESET_PACKAGE;
```

To run another large program:

```
EXECUTE large_plsql_program2;
```

## SET\_CONTEXT Procedure

This procedure sets the context, of which there are four types: session local, globally initialized, externally initialized, and globally accessed.

Of its five parameters, only the first three are required; the final two parameters are optional, used only in globally accessed contexts. Further parameter information appears in the parameter table and the usage notes.

### Syntax

```
DBMS_SESSION.SET_CONTEXT (
    namespace VARCHAR2,
    attribute  VARCHAR2,
    value      VARCHAR2,
    username   VARCHAR2,
    client_id  VARCHAR2 );
```

### Parameters

**Table 131–15 SET\_CONTEXT Procedure Parameters**

Parameter	Description
namespace	Namespace of the application context to be set, limited to 30 bytes
attribute	Attribute of the application context to be set, limited to 30 bytes
value	Value of the application context to be set, limited to 4 kilobytes.
username	Database username attribute of the application context. Default: NULL
client_id	Application-specific client_id attribute of the application context (64-byte maximum). Default: NULL

### Usage Notes

- The first three parameters are required for all types of context.
- The username parameter must be a valid SQL identifier.
- The client\_id parameter must be a string of at most 64 bytes. It is case-sensitive and must match the argument provided for set\_identifier.
- If the namespace parameter is a global context namespace, then the username parameter is matched against the current database user name in the session, and the client\_id parameter is matched against the current client\_id in the session. If these parameters are not set, NULL is assumed, enabling any user to see the context values.
- This procedure must be invoked directly or indirectly by the trusted package.
- The caller of SET\_CONTEXT must be in the calling stack of a procedure that has been associated to the context namespace through a CREATE CONTEXT statement. The checking of the calling stack does not cross a DBMS boundary.
- No limit applies to the number of attributes that can be set in a namespace. An attribute retains its value during the user's session unless it is reset by the user.

- If the value of the parameter in the namespace has been set, `SET_CONTEXT` overwrites this value.
- Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

**See Also:** *Oracle Database Security Guide* for more information about

- "Setting the username and client ID"
- "Example: Creating a Global Application Context that Uses a Client Session ID"

## SET\_EDITION\_DEFERRED Procedure

This procedure requests a switch to the specified edition. The switch takes effect at the end of the current client call.

### Syntax

```
DBMS_SESSION.SET_EDITION_DEFERRED (  
    edition    IN    VARCHAR2);
```

### Parameters

**Table 131–16** *SET\_EDITION\_DEFERRED Procedure Parameters*

Parameter	Description
edition	Name of the edition to which to switch. The contents of the string are processed as a SQL identifier; double-quotes must surround the remainder of the string if special characters or lower case characters are present in the edition's actual name and, if double-quotes are not used, the contents are set in uppercase. The caller must have USE privilege on the named edition.

## SET\_IDENTIFIER Procedure

This procedure sets the client ID in the session.

### Syntax

```
DBMS_SESSION.SET_IDENTIFIER (  
    client_id VARCHAR2);
```

### Parameters

**Table 131–17 SET\_IDENTIFIER Procedure Parameters**

Parameter	Description
<code>client_id</code>	The application-specific identifier of the current database session

### Usage Notes

Note the following:

- `SET_IDENTIFIER` initializes the current session with a client identifier to identify the associated global application context
- `client_id` is case sensitive; it must match the `client_id` parameter in the `set_context`
- This procedure is executable by public

## SET\_NLS Procedure

This procedure sets up your Globalization Support (NLS). It is equivalent to the following SQL statement:

```
ALTER SESSION SET <nls_parameter> = <value>
```

### Syntax

```
DBMS_SESSION.SET_NLS (  
    param VARCHAR2,  
    value VARCHAR2);
```

### Parameters

**Table 131–18 SET\_NLS Procedure Parameters**

Parameter	Description
param	Globalization Support parameter. The parameter name must begin with 'NLS'.
value	Parameter value. If the parameter is a text literal, then it needs embedded single-quotes. For example, "set_nls ('nls_date_format','DD-MON-YY')".

## SET\_ROLE Procedure

This procedure enables and disables roles. It is equivalent to the SET ROLE SQL statement.

### Syntax

```
DBMS_SESSION.SET_ROLE (  
    role_cmd VARCHAR2);
```

### Parameters

**Table 131–19 SET\_ROLE Procedure Parameters**

Parameter	Description
role_cmd	Text is appended to "set role" and then run as SQL

### Usage Notes

Note that the procedure creates a new transaction if it is not invoked from within an existing transaction.



## SET\_SQL\_TRACE Procedure

This procedure turns tracing on or off. It is equivalent to the following SQL statement:

```
ALTER SESSION SET SQL_TRACE ...
```

### Syntax

```
DBMS_SESSION.SET_SQL_TRACE (  
    sql_trace boolean);
```

### Parameters

**Table 131–20** *SET\_SQL\_TRACE Procedure Parameters*

Parameter	Description
sql_trace	TRUE turns tracing on, FALSE turns tracing off

## SWITCH\_CURRENT\_CONSUMER\_GROUP Procedure

This procedure changes the current resource consumer group of a user's current session.

This lets you switch to a consumer group if you have the switch privilege for that particular group. If the caller is another procedure, then this enables the user to switch to a consumer group for which the owner of that procedure has switch privilege.

### Syntax

```
DBMS_SESSION.switch_current_consumer_group (
    new_consumer_group    IN  VARCHAR2,
    old_consumer_group    OUT VARCHAR2,
    initial_group_on_error IN  BOOLEAN);
```

### Parameters

**Table 131–21 SWITCH\_CURRENT\_CONSUMER\_GROUP Procedure Parameters**

Parameter	Description
<code>new_consumer_group</code>	Name of consumer group to which you want to switch
<code>old_consumer_group</code>	Name of the consumer group from which you just switched out
<code>initial_group_on_error</code>	If TRUE, then sets the current consumer group of the caller to his/her initial consumer group in the event of an error

### Return Values

This procedure outputs the old consumer group of the user in the parameter `old_consumer_group`.

---

**Note:** You can switch back to the old consumer group later using the value returned in `old_consumer_group`.

---

### Exceptions

**Table 131–22 SWITCH\_CURRENT\_CONSUMER\_GROUP Procedure Exceptions**

Exception	Description
29368	Non-existent consumer group
1031	Insufficient privileges
29396	Cannot switch to OTHER_GROUPS consumer group

### Usage Notes

The owner of a procedure must have privileges on the group from which a user was switched (`old_consumer_group`) in order to switch them back. There is one exception: The procedure can always switch the user back to his/her initial consumer group (skipping the privilege check).

By setting `initial_group_on_error` to TRUE, SWITCH\_CURRENT\_CONSUMER\_GROUP puts the current session into the default group, if it can't put it into the group designated by `new_consumer_group`. The error associated with the attempt to move

a session into `new_consumer_group` is raised, even though the current consumer group has been changed to the initial consumer group.

## Examples

```
CREATE OR REPLACE PROCEDURE high_priority_task is
  old_group varchar2(30);
  prev_group varchar2(30);
  curr_user varchar2(30);
BEGIN
  -- switch invoker to privileged consumer group. If we fail to do so, an
  -- error is thrown, but the consumer group does not change
  -- because 'initial_group_on_error' is set to FALSE

  dbms_session.switch_current_consumer_group('tkrogrp1', old_group, FALSE);
  -- set up exception handler (in the event of an error, we do not want to
  -- return to caller while leaving the session still in the privileged
  -- group)

  BEGIN
    -- perform some operations while under privileged group

  EXCEPTION
    WHEN OTHERS THEN
      -- It is possible that the procedure owner does not have privileges
      -- on old_group. 'initial_group_on_error' is set to TRUE to make sure
      -- that the user is moved out of the privileged group in such a
      -- situation

      dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
      RAISE;
    END;

  -- we've succeeded. Now switch to old_group, or if cannot do so, switch
  -- to caller's initial consumer group

  dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
END high_priority_task;
/
```

## UNIQUE\_SESSION\_ID Function

This function returns an identifier that is unique for all sessions currently connected to this database. Multiple calls to this function during the same session always return the same result.

### Syntax

```
DBMS_SESSION.UNIQUE_SESSION_ID  
RETURN VARCHAR2;
```

### Pragmas

```
pragma restrict_references(unique_session_id,WNDS,RNDS,WNPS);
```

### Return Values

**Table 131–23** *UNIQUE\_SESSION\_ID Function Return Values*

Return	Description
unique_session_id	Returns up to 24 bytes

---

---

## DBMS\_SHARED\_POOL

The `DBMS_SHARED_POOL` package provides access to the shared pool, which is the shared memory area where cursors and PL/SQL objects are stored. `DBMS_SHARED_POOL` enables you to display the sizes of objects in the shared pool, and mark them for keeping or not-keeping in order to reduce memory fragmentation.

This chapter contains the following topics:

- [Using DBMS\\_SHARED\\_POOL](#)
  - Overview
  - Operational Notes
- [Summary of DBMS\\_SHARED\\_POOL Subprograms](#)

---

## Using DBMS\_SHARED\_POOL

- [Overview](#)
- [Operational Notes](#)

## Overview

The procedures provided here may be useful when loading large PL/SQL objects. When large PL/SQL objects are loaded, users response time is affected because of the large number of smaller objects that need to be aged out from the shared pool to make room (due to memory fragmentation). In some cases, there may be insufficient memory to load the large objects.

DBMS\_SHARED\_POOL is also useful for frequently executed triggers. You may want to keep compiled triggers on frequently used tables in the shared pool.

Additionally, DBMS\_SHARED\_POOL supports sequences. Sequence numbers are lost when a sequence is aged out of the shared pool. DBMS\_SHARED\_POOL is useful for keeping sequences in the shared pool and thus preventing the loss of sequence numbers.

## Operational Notes

To create `DBMS_SHARED_POOL`, run the `DBMSPOOL.SQL` script. The `PRVTPPOOL.PLB` script is automatically executed after `DBMSPOOL.SQL` runs. These scripts are *not* run by as part of standard database creation.



---

## Summary of DBMS\_SHARED\_POOL Subprograms

**Table 132-1 DBMS\_SHARED\_POOL Package Subprograms**

Subprogram	Description
<a href="#">ABORTED_REQUEST_THRESHOLD Procedure</a> on page 132-6	Sets the aborted request threshold for the shared pool
<a href="#">KEEP Procedure</a> on page 132-7	Keeps an object in the shared pool
<a href="#">MARKHOT Procedure</a> on page 132-9	Marks a library cache object as a hot object
<a href="#">PURGE Procedure</a> on page 132-10	Purges the named object or specified heap(s) of the object
<a href="#">SIZES Procedure</a> on page 132-12	Shows objects in the shared pool that are larger than the specified size
<a href="#">UNKEEP Procedure</a> on page 132-13	Unkeeps the named object
<a href="#">UNMARKHOT Procedure</a> on page 132-14	Unmarks a library cache object as a hot object

## ABORTED\_REQUEST\_THRESHOLD Procedure

This procedure sets the aborted request threshold for the shared pool.

### Syntax

```
DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD (  
    threshold_size NUMBER);
```

### Parameters

**Table 132–2 ABORTED\_REQUEST\_THRESHOLD Procedure Parameters**

Parameter	Description
threshold_size	Size, in bytes, of a request which does not try to free unpinned (not "unkeep-ed") memory within the shared pool. The range of threshold_size is 5000 to ~2 GB inclusive.

### Exceptions

An exception is raised if the threshold is not in the valid range.

### Usage Notes

Usually, if a request cannot be satisfied on the free list, then the RDBMS tries to reclaim memory by freeing objects from the LRU list and checking periodically to see if the request can be fulfilled. After finishing this step, the RDBMS has performed a near equivalent of an 'ALTER SYSTEM FLUSH SHARED\_POOL'.

Because this impacts all users on the system, this procedure "localizes" the impact to the process failing to find a piece of shared pool memory of size greater than threshold\_size. This user gets the 'out of memory' error without attempting to search the LRU list.

## KEEP Procedure

This procedure keeps an object in the shared pool. Once an object has been kept in the shared pool, it is not subject to aging out of the pool. This may be useful for frequently used large objects. When large objects are brought into the shared pool, several objects may need to be aged out to create a contiguous area large enough.

### Syntax

```
DBMS_SHARED_POOL.KEEP (
    name VARCHAR2,
    flag CHAR          DEFAULT 'P');
```

### Parameters

**Table 132–3 KEEP Procedure Parameters**

Parameter	Description
name	Name of the object to keep.  The value for this identifier is the concatenation of the address and hash_value columns from the v\$sqlarea view. This is displayed by the SIZES procedure.  Currently, TABLE and VIEW objects may not be kept.
flag	(Optional) If this is not specified, then the package assumes that the first parameter is the name of a package/procedure/function and resolves the name.  Set to 'P' or 'p' to fully specify that the input is the name of a package/procedure/function.  Set to 'T' or 't' to specify that the input is the name of a type.  Set to 'R' or 'r' to specify that the input is the name of a trigger.  Set to 'Q' or 'q' to specify that the input is the name of a sequence.  In case the first argument is a cursor address and hash-value, the parameter should be set to any character except 'P' or 'p' or 'Q' or 'q' or 'R' or 'r' or 'T' or 't'.

### Exceptions

An exception is raised if the named object cannot be found.

### Usage Notes

There are two kinds of objects:

- PL/SQL objects, triggers, sequences, and types which are specified by name
- SQL cursor objects which are specified by a two-part number (indicating a location in the shared pool).

For example:

```
DBMS_SHARED_POOL.KEEP('scott.hispackage')
```

This keeps package HISPACKAGE, owned by SCOTT. The names for PL/SQL objects follow SQL rules for naming objects (for example, delimited identifiers and multibyte

names are allowed). A cursor can be kept by `DBMS_SHARED_POOL.KEEP('0034CDFF, 20348871', 'C')`, `0034CDFF` being the `ADDRESS` and `20348871` the `HASH_VALUE`. Note that the complete hexadecimal address must be in the first 8 characters.

## MARKHOT Procedure

This procedure marks a library cache object as a hot object.

### Syntax

```
DBMS_SHARED_POOL.MARKHOT (
  schema      VARCHAR2,
  objname     VARCHAR2,
  namespace   NUMBER DEFAULT 1,
  global      BOOLEAN DEFAULT TRUE);
```

```
DBMS_SHARED_POOL.MARKHOT (
  hash        VARCHAR2,
  namespace   NUMBER DEFAULT 1,
  global      BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 132–4 MARKHOT Procedure Parameters**

Parameter	Description
schema	User name or the schema to which the object belongs
objname	Name of the object
namespace	Number indicating the library cache namespace in which the object is to be searched
global	If TRUE (default), mark the object hot on all OracleRAC instances
hash	16-byte hash value for the object

### Exceptions

ORA-06502: An exception is raised if the named object cannot be found due to incorrect input

ORA-04043: An exception is raised if the named object cannot be found (bad namespace, or hash input)

## PURGE Procedure

This procedure purges the named object or specified heap(s) of the object.

### Syntax

```
DBMS_SHARED_POOL.PURGE (
  name          VARCHAR2,
  flag          CHAR DEFAULT 'P',
  heaps         NUMBER DEFAULT 1);
```

```
DBMS_SHARED_POOL.PURGE (
  schema        VARCHAR2,
  objname       VARCHAR2,
  namespace     NUMBER,
  heaps         NUMBER);
```

```
DBMS_SHARED_POOL.PURGE (
  hash          VARCHAR2,
  namespace     NUMBER,
  heaps         NUMBER);
```

### Parameters

**Table 132–5 PURGE Procedure Parameters**

Parameter	Description
name	Name of the object to purge.  The value for this identifier is the concatenation of the address and hash_value columns from the v\$sqlarea view. This is displayed by the SIZES procedure.  Currently, TABLE and VIEW objects may not be purged.
flag	(Optional) If this is not specified, then the package assumes that the first parameter is the name of a package/procedure/function and resolves the name.  Set to 'P' or 'p' to fully specify that the input is the name of a package/procedure/function.  Set to 'T' or 't' to specify that the input is the name of a type.  Set to 'R' or 'r' to specify that the input is the name of a trigger.  Set to 'Q' or 'q' to specify that the input is the name of a sequence.  In case the first argument is a cursor address and hash-value, the parameter should be set to any character except 'P' or 'p' or 'Q' or 'q' or 'R' or 'r' or 'T' or 't'.
heaps	Heaps to be purged. For example, if heap 0 and heap 6 are to be purged:  1<<0   1<<6 => hex 0x41 => decimal 65, so specify heaps =>65.  Default is 1, that is, heap 0 which means the whole object would be purged
schema	User name or the schema to which the object belongs
objname	Name of the object to purge
namespace	Parameter is a number indicating the library cache namespace in which the object is to be searched

**Table 132-5 (Cont.) PURGE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
hash	16-byte hash value for the object

**Exceptions**

ORA-6570: An exception is raised if the named object cannot be found

ORA-6570: An object cannot be purged it marked as permanently kept

**Usage Notes**

All objects supported by the [KEEP Procedure](#) are supported for PURGE.

## SIZES Procedure

This procedure shows objects in the `shared_pool` that are larger than the specified size. The name of the object is also given, which can be used as an argument to either the `KEEP` or `UNKEEP` calls.

### Syntax

```
DBMS_SHARED_POOL.SIZES (  
    minsize NUMBER);
```

### Parameters

**Table 132–6 SIZES Procedure Parameters**

Parameter	Description
<code>minsize</code>	Size, in kilobytes, over which an object must be occupying in the shared pool, in order for it to be displayed.

### Usage Notes

Issue the `SQLDBA` or `SQLPLUS` `'SET SERVEROUTPUT ON SIZE XXXXX'` command prior to using this procedure so that the results are displayed.



## UNKEEP Procedure

This procedure unkeeps the named object.

### Syntax

```
DBMS_SHARED_POOL.UNKEEP (  
    name VARCHAR2,  
    flag CHAR      DEFAULT 'P');
```

---

---

**Caution:** This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

---

---

### Parameters

**Table 132–7 UNKEEP Procedure Parameters**

Parameter	Description
name	Name of the object to unkeep. See description of the name object for the KEEP procedure.
flag	See description of the flag parameter for the KEEP procedure.

### Exceptions

An exception is raised if the named object cannot be found.

## UNMARKHOT Procedure

This procedure unmarks a library cache object as a hot object.

### Syntax

```
DBMS_SHARED_POOL.UNMARKHOT (  
    schema          VARCHAR2,  
    objname         VARCHAR2,  
    namespace       NUMBER DEFAULT 1,  
    global          BOOLEAN DEFAULT TRUE);
```

```
DBMS_SHARED_POOL.UNMARKHOT (  
    hash           VARCHAR2,  
    namespace       NUMBER DEFAULT 1,  
    global          BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 132–8 UNMARKHOT Procedure Parameters**

Parameter	Description
schema	User name or the schema to which the object belongs
objname	Name of the object
namespace	Number indicating the library cache namespace in which the object is to be searched
global	If TRUE (default), unmark the object hot on all Oracle RAC instances
hash	16-byte hash value for the object

### Exceptions

ORA-06502: An exception is raised if the named object cannot be found due to incorrect input

ORA-04043: An exception is raised if the named object cannot be found (bad namespace, or hash input, or non-existent object)

The DBMS\_SPACE package enables you to analyze segment growth and space requirements.

This chapter contains the following topics:

- [Using DBMS\\_SPACE](#)
  - Security Model
- [Data Structures](#)
- [Summary of DBMS\\_SPACE Subprograms](#)

## Using DBMS\_SPACE

- [Security Model](#)
-

## Security Model

This package runs with `SYS` privileges. The execution privilege is granted to `PUBLIC`. Subprograms in this package run under the caller security. The user must have `ANALYZE` privilege on the object.

## Data Structures

The DBMS\_SPACE package defines an OBJECT type, a RECORD type, and a TABLE type.

### OBJECT Types

[CREATE\\_TABLE\\_COST\\_COLINFO](#) Object Type

### RECORD Types

[ASA\\_RECO\\_ROW](#) Record Type

### TABLE Types

[ASA\\_RECO\\_ROW\\_TB](#) Table Type

---

## CREATE\_TABLE\_COST\_COLINFO Object Type

This type describes the data type and size of a column in the table.

### Syntax

```
TYPE create_table_cost_colinfo IS OBJECT(  
  col_type    VARCHAR(200),  
  col_size    NUMBER)
```

### Attributes

**Table 133-1** CQ\_NOTIFICATION\$\_DESCRIPTOR Object Type

Attribute	Description
col_type	Column type
col_size	Column size

## ASA\_RECO\_ROW Record Type

This type contains the column type of individual columns returned by the [ASA\\_RECOMMENDATIONS Function](#).

### Syntax

```
TYPE asa_reco_row IS RECORD (
    tablespace_name    VARCHAR2 (30) ,
    segment_owner     VARCHAR2 (30) ,
    segment_name      VARCHAR2 (30) ,
    segment_type      VARCHAR2 (18) ,
    partition_name    VARCHAR2 (30) ,
    allocated_space    NUMBER,
    used_space        NUMBER,
    reclaimable_space NUMBER,
    chain_rowexcess   NUMBER,
    recommendations   VARCHAR2 (1000) ,
    c1                VARCHAR2 (1000) ,
    c2                VARCHAR2 (1000) ,
    c3                VARCHAR2 (1000) ,
    task_id           NUMBER,
    mesg_id           NUMBER) ;
```

### Attributes

**Table 133–2 ASA\_RECO\_ROW Attributes**

Field	Description
tablespace_name	Name of the tablespace containing the object
segment_owner	Name of the schema
segment_name	Name of the object
segment_type	Type of the segment 'TABLE','INDEX' and so on
partition_name	Name of the partition
allocated_space	Space allocated to the segment
used_space	Space actually used by the segment
reclaimable_space	Reclaimable free space in the segment
chain_rowexcess	Percentage of excess chain row pieces that can be eliminated
recommendations	Recommendation or finding for this segment
c1	Command associated with the recommendation
c2	Command associated with the recommendation
c3	Command associated with the recommendation
task_id	Advisor Task that processed this segment
mesg_id	Message ID corresponding to the recommendation



## ASA\_RECO\_ROW\_TB Table Type

### Syntax

```
TYPE asa_reco_row_tb IS TABLE OF asa_reco_row;
```

---

## Summary of DBMS\_SPACE Subprograms

**Table 133-3 DBMS\_SPACE Package Subprograms**

Subprogram	Description
<a href="#">ASA_RECOMMENDATIONS Function</a> on page 133-9	Returns recommendations/findings of segment advisor run automatically by the system or manually invoked by the user
<a href="#">CREATE_INDEX_COST Procedure</a> on page 133-10	Determines the cost of creating an index on an existing table
<a href="#">CREATE_TABLE_COST Procedures</a> on page 133-11	Determines the size of the table given various attributes
<a href="#">FREE_BLOCKS Procedure</a> on page 133-13	Returns information about free blocks in an object (table, index, or cluster)
<a href="#">ISDATAFILEDROPPABLE_NAME Procedure</a> on page 133-15	Checks whether a datafile is droppable
<a href="#">OBJECT_DEPENDENT_SEGMENTS Function</a> on page 133-16	Returns the list of segments that are associated with the object
<a href="#">OBJECT_GROWTH_TREND Function</a> on page 133-18	A table function where each row describes the space usage of the object at a specific point in time
<a href="#">SPACE_USAGE Procedures</a> on page 133-20	Returns information about free blocks in an auto segment space managed segment
<a href="#">UNUSED_SPACE Procedure</a> on page 133-23	Returns information about unused space in an object (table, index, or cluster)

## ASA\_RECOMMENDATIONS Function

This function returns recommendations using the stored results of the auto segment advisor. This function returns results from the latest run on any given object.

### Syntax

```
DBMS_SPACE.ASA_RECOMMENDATIONS (
    all_runs          IN    VARCHAR2 DEFAULT := TRUE,
    show_manual      IN    VARCHAR2 DEFAULT := TRUE,
    show_findings    IN    VARCHAR2 DEFAULT := FALSE)
RETURN ASA_RECO_ROW_TB PIPELINED;
```

### Parameters

**Table 133–4** *ASA\_RECOMMENDATIONS Procedure Parameters*

Parameter	Description
all_runs	If TRUE, returns recommendations/findings for all runs of auto segment advisor. If FALSE, returns the results of the LATEST run only. LATEST does not make sense for manual invocation of segment advisor. This is applicable only for auto advisor.
show_manual	If TRUE, we show the results of manual invocations only. The auto advisor results are excluded. If FALSE, results of manual invocation of segment advisor are not returned.
show_findings	Show only the findings instead of the recommendations

## CREATE\_INDEX\_COST Procedure

This procedure determines the cost of creating an index on an existing table. The input is the DDL statement that will be used to create the index. The procedure will output the storage required to create the index.

### Syntax

```
DBMS_SPACE.CREATE_INDEX_COST (
  ddl          IN   VARCHAR2,
  used_bytes   OUT  NUMBER,
  alloc_bytes  OUT  NUMBER,
  plan_table   IN   VARCHAR2 DEFAULT NULL);
```

### Pragmas

```
pragma restrict_references(create_index_cost,WNDS);
```

### Parameters

**Table 133–5 CREATE\_INDEX\_COST Procedure Parameters**

Parameter	Description
ddl	The create index DDL statement
used_bytes	The number of bytes representing the actual index data
alloc_bytes	Size of the index when created in the tablespace
plan_table	Which plan table to use, default NULL

### Usage Notes

- The table on which the index is created must already exist.
- The computation of the index size depends on statistics gathered on the segment.
- It is imperative that the table must have been analyzed recently.
- In the absence of correct statistics, the results may be inaccurate, although the procedure will not raise any errors.

## CREATE\_TABLE\_COST Procedures

This procedure is used in capacity planning to determine the size of the table given various attributes. The size of the object can vary widely based on the tablespace storage attributes, tablespace block size, and so on. There are two overloads of this procedure.

- The first version takes the column information of the table as argument and outputs the table size.
- The second version takes the average row size of the table as argument and outputs the table size.

This procedure can be used on tablespace of dictionary managed and locally managed extent management as well as manual and auto segment space management.

### Syntax

```
DBMS_SPACE.CREATE_TABLE_COST (
  tablespace_name  IN VARCHAR2,
  avg_row_size     IN NUMBER,
  row_count        IN NUMBER,
  pct_free         IN NUMBER,
  used_bytes       OUT NUMBER,
  alloc_bytes      OUT NUMBER);
```

```
DBMS_SPACE.CREATE_TABLE_COST (
  tablespace_name  IN VARCHAR2,
  colinfos         IN CREATE_TABLE_COST_COLUMNS,
  row_count        IN NUMBER,
  pct_free         IN NUMBER,
  used_bytes       OUT NUMBER,
  alloc_bytes      OUT NUMBER);
```

```
CREATE TYPE create_table_cost_colinfo IS OBJECT (
  COL_TYPE  VARCHAR(200),
  COL_SIZE  NUMBER);
```

### Parameters

**Table 133–6 CREATE\_TABLE\_COST Procedure Parameters**

Parameter	Description
tablespace_name	The tablespace in which the object will be created. The default is SYSTEM tablespace.
avg_row_size	The anticipated average row size in the table
colinfos	The description of the columns
row_count	The anticipated number of rows in the table
pct_free	The percentage of free space in each block for future expansion of existing rows due to updates
used_bytes	The space used by user data
alloc_bytes	The size of the object taking into account the tablespace extent characteristics

## Usage Notes

- The `used_bytes` represent the actual bytes used by the data. This includes the overhead due to the block metadata, `pctfree` etc.
- The `alloc_bytes` represent the size of the table when it is created in the tablespace. This takes into account, the size of the extents in the tablespace and tablespace extent management properties.

## Examples

```
-- review the parameters
SELECT argument_name, data_type, type_owner, type_name
FROM all_arguments
WHERE object_name = 'CREATE_TABLE_COST'
AND overload = 2

-- examine the input parameter type
SELECT text
FROM dba_source
WHERE name = 'CREATE_TABLE_COST_COLUMNS';

-- drill down further into the input parameter type
SELECT text
FROM dba_source
WHERE name = 'create_table_cost_colinfo';

set serveroutput on

DECLARE
  ub NUMBER;
  ab NUMBER;
  c1 sys.create_table_cost_columns;
BEGIN
  c1 := sys.create_table_cost_columns( sys.create_table_cost_colinfo('NUMBER',10),
    sys.create_table_cost_colinfo('VARCHAR2',30),
    sys.create_table_cost_colinfo('VARCHAR2',30),
    sys.create_table_cost_colinfo('DATE',NULL));

  DBMS_SPACE.CREATE_TABLE_COST('SYSTEM',c1,100000,0,ub,ab);

  DBMS_OUTPUT.PUT_LINE('Used Bytes: ' || TO_CHAR(ub));
  DBMS_OUTPUT.PUT_LINE('Alloc Bytes: ' || TO_CHAR(ab));
END;
/
```

## FREE\_BLOCKS Procedure

This procedure returns information about free blocks in an object (table, index, or cluster). See [SPACE\\_USAGE Procedures](#) for returning free block information in an auto segment space managed segment.

### Syntax

```
DBMS_SPACE.FREE_BLOCKS (
  segment_owner   IN VARCHAR2,
  segment_name    IN VARCHAR2,
  segment_type    IN VARCHAR2,
  freelist_group_id IN NUMBER,
  free_blks       OUT NUMBER,
  scan_limit      IN NUMBER DEFAULT NULL,
  partition_name  IN VARCHAR2 DEFAULT NULL);
```

### Pragmas

```
pragma restrict_references (free_blocks,WNDS);
```

### Parameters

**Table 133–7 FREE\_BLOCKS Procedure Parameters**

Parameter	Description
segment_owner	Schema name of the segment to be analyzed
segment_name	Segment name of the segment to be analyzed
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER): <ul style="list-style-type: none"> <li>■ TABLE</li> <li>■ TABLE PARTITION</li> <li>■ TABLE SUBPARTITION</li> <li>■ INDEX</li> <li>■ INDEX PARTITION</li> <li>■ INDEX SUBPARTITION</li> <li>■ CLUSTER</li> <li>■ LOB</li> <li>■ LOB PARTITION</li> <li>■ LOB SUBPARTITION</li> </ul>
freelist_group_id	Freelist group (instance) whose free list size is to be computed
free_blks	Returns count of free blocks for the specified group
scan_limit	Maximum number of free list blocks to read (optional). Use a scan limit of X you are interested only in the question, "Do I have X blocks on the free list?"
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables. The name of subpartition should be used when partitioning is composite.

**Examples**

The following uses the CLUS cluster in SCOTT schema with 4 freelist groups. It returns the number of blocks in freelist group 3 in CLUS.

```
DBMS_SPACE.FREE_BLOCKS('SCOTT', 'CLUS', 'CLUSTER', 3, :free_blocks);
```

---

---

**Note:** An error is raised if `scan_limit` is not a positive number.

---

---



## ISDATAFILEDROPPABLE\_NAME Procedure

This procedure checks whether a datafile is droppable. This procedure may be called before actually dropping the file.

### Syntax

```
DBMS_SPACE.ISDATAFILEDROPPABLE_NAME (
  filename      IN      VARCHAR2,
  retval        OUT     NUMBER);
```

### Pragmas

```
pragma restrict_references (free_blocks,WNDS);
```

### Parameters

**Table 133–8 ISDATAFILEDROPPABLE\_NAME Procedure Parameters**

Parameter	Description
filename	Name of the file
retval	Values: 0 if the file is not droppable, 1 if the file is droppable.

### Examples

```
DECLARE
  fname      VARCHAR2(100);
  retval     NUMBER;
BEGIN
  SELECT file_name
  INTO fname
  FROM dba_data_files
  WHERE file_name like '%empty%';

  DBMS_SPACE.ISDATAFILEDROPPABLE_NAME(fname, retval);
  DBMS_OUTPUT.PUT_LINE(retval);
END;
/
```

## OBJECT\_DEPENDENT\_SEGMENTS Function

This table function, given an object, returns the list of segments that are associated with the object.

### Syntax

```
DBMS_SPACE.OBJECT_DEPENDENT_SEGMENTS (
  objowner   IN   VARCHAR2,
  objname    IN   VARCHAR2,
  partname   IN   VARCHAR2,
  objtype    IN   NUMBER)
RETURN dependent_segments_table PIPELINED;
```

### Parameters

**Table 133–9** *OBJECT\_DEPENDENT\_SEGMENTS Function Parameters*

Parameter	Description
objowner	The schema containing the object
objname	The name of the object
partname	The name of the partition
objtype	Type of the object: <ul style="list-style-type: none"> <li>▪ OBJECT_TYPE_TABLE constant positive := 1;</li> <li>▪ OBJECT_TYPE_NESTED_TABLE constant positive := 2;</li> <li>▪ OBJECT_TYPE_INDEX constant positive := 3;</li> <li>▪ OBJECT_TYPE_CLUSTER constant positive := 4;</li> <li>▪ OBJECT_TYPE_TABLE_PARTITION constant positive := 7;</li> <li>▪ OBJECT_TYPE_INDEX_PARTITION constant positive := 8;</li> <li>▪ OBJECT_TYPE_TABLE_SUBPARTITION constant positive := 9;</li> <li>▪ OBJECT_TYPE_INDEX_SUBPARTITION constant positive := 10;</li> <li>▪ OBJECT_TYPE_MV constant positive := 13;</li> <li>▪ OBJECT_TYPE_MVLOG constant positive := 14;</li> </ul>

### Return Values

The content of one row of a `dependent_segments_table`:

```
TYPE object_dependent_segment IS RECORD (
  segment_owner   VARCHAR2(100),
  segment_name    VARCHAR2(100),
  segment_type    VARCHAR2(100),
  tablespace_name VARCHAR2(100),
  partition_name  VARCHAR2(100),
```

---

```
lob_column_name    VARCHAR2(100);
```

**Table 133–10** *OBJECT\_DEPENDENT\_SEGMENT Type Parameters*

<b>Parameter</b>	<b>Description</b>
segment_owner	The schema containing the segment
segment_name	The name of the segment
segment_type	The type of the segment, such as table, index or LOB
tablespace_name	The name of the tablespace
partition_name	The name of the partition, if any
lob_column_name	The name of the LOB column, if any

## OBJECT\_GROWTH\_TREND Function

This is a table function. The output will be in the form of one or more rows where each row describes the space usage of the object at a specific point in time. Either the space usage totals will be retrieved from Automatic Workload Repository Facilities (AWRF), or the current space usage will be computed and combined with space usage deltas retrieved from AWRF.

### Syntax

```
DBMS_SPACE.OBJECT_GROWTH_TREND (
  object_owner      IN   VARCHAR2,
  object_name       IN   VARCHAR2,
  object_type       IN   VARCHAR2,
  partition_name    IN   VARCHAR2 DEFAULT NULL,
  start_time        IN   TIMESTAMP DEFAULT NULL,
  end_time          IN   TIMESTAMP DEFAULT NULL,
  interval          IN   DSINTERVAL_UNCONSTRAINED DEFAULT NULL,
  skip_interpolated IN   VARCHAR2 DEFAULT 'FALSE',
  timeout_seconds   IN   NUMBER DEFAULT NULL,
  single_datapoint_flag IN VARCHAR2 DEFAULT 'TRUE')
RETURN object_growth_trend_table PIPELINED;
```

### Parameters

**Table 133–11** OBJECT\_GROWTH\_TREND Function Parameters

Parameter	Description
object_owner	The schema containing the object
object_name	The name of the object
object_type	The type of the object
partition_name	The name of the partition
start_time	Statistics generated after this time will be used in generating the growth trend
end_time	Statistics generated until this time will be used in generating the growth trend
interval	The interval at which to sample
skip_interpolated	Whether interpolation of missing values should be skipped
timeout_seconds	The time-out value for the function in seconds
single_data_point_flag	Whether in the absence of statistics the segment should be sampled

### Return Values

The `object_growth_trend_row` and `object_growth_trend_table` are used by the `OBJECT_GROWTH_TREND` table function to describe its output.

```
TYPE object_growth_trend_row IS RECORD(
  timepoint      TIMESTAMP,
  space_usage    NUMBER,
  space_alloc    NUMBER,
  quality        VARCHAR(20));
```

**Table 133-12** *OBJECT\_GROWTH\_TREND\_ROW* Type Parameters

<b>Parameter</b>	<b>Description</b>
timepoint	The time at which the statistic was recorded
space_usage	The space used by data
space_alloc	The size of the segment including overhead and unused space
quality	The quality of result: "GOOD", "INTERPOLATED", "PROJECTION"

```
TYPE object_growth_trend_table IS TABLE OF object_growth_trend_row;
```

## SPACE\_USAGE Procedures

The first form of the procedure shows the space usage of data blocks under the segment High Water Mark. You can calculate usage for LOBs, LOB PARTITIONS and LOB SUBPARTITIONS. This procedure can only be used on tablespaces that are created with auto segment space management. The bitmap blocks, segment header, and extent map blocks are not accounted for by this procedure. Note that this overload cannot be used on SECUREFILE LOBs.

The second form of the procedure returns information about SECUREFILE LOB space usage. It will return the amount of space in blocks being used by all the SECUREFILE LOBs in the LOB segment. The procedure displays the space actively used by the LOB column, freed space that has retention expired, and freed space that has retention unexpired. Note that this overload can be used only on SECUREFILE LOBs.

### Syntax

```
DBMS_SPACE.SPACE_USAGE(
    segment_owner      IN  VARCHAR2,
    segment_name       IN  VARCHAR2,
    segment_type       IN  VARCHAR2,
    unformatted_blocks OUT NUMBER,
    unformatted_bytes  OUT NUMBER,
    fs1_blocks         OUT NUMBER,
    fs1_bytes          OUT NUMBER,
    fs2_blocks         OUT NUMBER,
    fs2_bytes          OUT NUMBER,
    fs3_blocks         OUT NUMBER,
    fs3_bytes          OUT NUMBER,
    fs4_blocks         OUT NUMBER,
    fs4_bytes          OUT NUMBER,
    full_blocks        OUT NUMBER,
    full_bytes         OUT NUMBER,
    partition_name     IN  VARCHAR2 DEFAULT NULL);
```

```
DBMS_SPACE.SPACE_USAGE(
    segment_owner      IN  VARCHAR2,
    segment_name       IN  VARCHAR2,
    segment_type       IN  VARCHAR2,
    segment_size_blocks OUT NUMBER,
    segment_size_bytes OUT NUMBER,
    used_blocks        OUT NUMBER,
    used_bytes         OUT NUMBER,
    expired_blocks     OUT NUMBER,
    expired_bytes      OUT NUMBER,
    unexpired_blocks   OUT NUMBER,
    unexpired_bytes    OUT NUMBER,
    partition_name     IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 133–13** SPACE\_USAGE Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed
segment_name	Name of the segment to be analyzed
partition_name	Partition name of the segment to be analyzed

**Table 133–13 (Cont.) SPACE\_USAGE Procedure Parameters**

Parameter	Description
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER): <ul style="list-style-type: none"> <li>▪ TABLE</li> <li>▪ TABLE PARTITION</li> <li>▪ TABLE SUBPARTITION</li> <li>▪ INDEX</li> <li>▪ INDEX PARTITION</li> <li>▪ INDEX SUBPARTITION</li> <li>▪ CLUSTER</li> <li>▪ LOB</li> <li>▪ LOB PARTITION</li> <li>▪ LOB SUBPARTITION</li> </ul>
unformatted_blocks	Total number of blocks unformatted
unformatted_bytes	Total number of bytes unformatted
fs1_blocks	Number of blocks having at least 0 to 25% free space
fs1_bytes	Number of bytes having at least 0 to 25% free space
fs2_blocks	Number of blocks having at least 25 to 50% free space
fs2_bytes	Number of bytes having at least 25 to 50% free space
fs3_blocks	Number of blocks having at least 50 to 75% free space
fs3_bytes	Number of bytes having at least 50 to 75% free space
fs4_blocks	Number of blocks having at least 75 to 100% free space
fs4_bytes	Number of bytes having at least 75 to 100% free space
full_blocks	Total number of blocks full in the segment
full_bytes	Total number of bytes full in the segment
segment_size_blocks	Number of blocks allocated to the segment
segment_size_bytes	Number of bytes allocated to the segment
used_blocks	Number blocks allocated to the LOB that contains active data
used_bytes	Number bytes allocated to the LOB that contains active data
expired_blocks	Number of expired blocks used by the LOB to keep version data
expired_bytes	Number of expired bytes used by the LOB to keep version data
unexpired_blocks	Number of unexpired blocks used by the LOB to keep version data
unexpired_bytes	Number of unexpired bytes used by the LOB to keep version data
partition_name	Name of the partition (NULL if not a partition)

**Examples**

```
variable unf number;
variable unfb number;
variable fs1 number;
```

```
variable fs1b number;
variable fs2 number;
variable fs2b number;
variable fs3 number;
variable fs3b number;
variable fs4 number;
variable fs4b number;
variable full number;
variable fullb number;

begin
dbms_space.space_usage('U1','T',
                       'TABLE',
                       :unf, :unfb,
                       :fs1, :fs1b,
                       :fs2, :fs2b,
                       :fs3, :fs3b,
                       :fs4, :fs4b,
                       :full, :fullb);

end;
/
print unf ;
print unfb ;
print fs4 ;
print fs4b;
print fs3 ;
print fs3b;
print fs2 ;
print fs2b;
print fs1 ;
print fs1b;
print full;
print fullb;
```



## UNUSED\_SPACE Procedure

This procedure returns information about unused space in an object (table, index, or cluster).

### Syntax

```
DBMS_SPACE.UNUSED_SPACE (
    segment_owner      IN  VARCHAR2,
    segment_name       IN  VARCHAR2,
    segment_type       IN  VARCHAR2,
    total_blocks       OUT NUMBER,
    total_bytes        OUT NUMBER,
    unused_blocks      OUT NUMBER,
    unused_bytes       OUT NUMBER,
    last_used_extent_file_id OUT NUMBER,
    last_used_extent_block_id OUT NUMBER,
    last_used_block    OUT NUMBER,
    partition_name     IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 133–14** *UNUSED\_SPACE Procedure Parameters*

Parameter	Description
segment_owner	Schema name of the segment to be analyzed
segment_name	Segment name of the segment to be analyzed
segment_type	Type of the segment to be analyzed (TABLE, INDEX, or CLUSTER): <ul style="list-style-type: none"> <li>■ TABLE</li> <li>■ TABLE PARTITION</li> <li>■ TABLE SUBPARTITION</li> <li>■ INDEX</li> <li>■ INDEX PARTITION</li> <li>■ INDEX SUBPARTITION</li> <li>■ CLUSTER</li> <li>■ LOB</li> <li>■ LOB PARTITION</li> <li>■ LOB SUBPARTITION</li> </ul>
total_blocks	Returns total number of blocks in the segment
total_bytes	Returns total number of blocks in the segment, in bytes
unused_blocks	Returns number of blocks which are not used
unused_bytes	Returns, in bytes, number of blocks which are not used
last_used_extent_file_id	Returns the file ID of the last extent which contains data
last_used_extent_block_id	Returns the starting block ID of the last extent which contains data
last_used_block	Returns the last block within this extent which contains data

**Table 133–14 (Cont.) UNUSED\_SPACE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>partition_name</code>	Partition name of the segment to be analyzed.  This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

## Examples

The following declares the necessary bind variables and executes.

```
DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks,  
:total_bytes, :unused_blocks, :unused_bytes, :lastextf,  
:last_extb, :lastusedblock);
```

---

---

## DBMS\_SPACE\_ADMIN

The DBMS\_SPACE\_ADMIN package provides functionality for locally managed tablespaces.

**See Also:** *Oracle Database Administrator's Guide* for an example and description of using DBMS\_SPACE\_ADMIN.

This chapter contains the following topics:

- [Using DBMS\\_SPACE\\_ADMIN](#)
  - Security Model
  - Constants
  - Operational Notes
- [Summary of DBMS\\_SPACE\\_ADMIN Subprograms](#)

---

## Using DBMS\_SPACE\_ADMIN

This section contains topics which relate to using the DBMS\_SPACE\_ADMIN package.

- [Security Model](#)
- [Constants](#)
- [Operational Notes](#)

## Security Model

This package runs with *SYS* privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

## Constants

**Table 134–1 DBMS\_SPACE\_ADMIN Constants**

Constant	Type	Value	Description
SEGMENT_VERIFY_EXTENTS	POSITIVE	1	Verifies that the space owned by segment is appropriately reflected in the bitmap as used
SEGMENT_VERIFY_EXTENTS_GLOBAL	POSITIVE	2	Verifies that the space owned by segment is appropriately reflected in the bitmap as used and that no other segment claims any of this space to be used by it
SEGMENT_MARK_CORRUPT	POSITIVE	3	Marks a temporary segment as corrupt whereby facilitating its elimination from the dictionary (without space reclamation)
SEGMENT_MARK_VALID	POSITIVE	4	Marks a corrupt temporary segment as valid. It is useful when the corruption in the segment extent map or elsewhere has been resolved and the segment can be dropped normally.
SEGMENT_DUMP_EXTENT_MAP	POSITIVE	5	Dumps the extent map for a given segment
TABLESPACE_VERIFY_BITMAP	POSITIVE	6	Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent
TABLESPACE_EXTENT_MAKE_FREE	POSITIVE	7	Marks the block range (extent) as free in the bitmaps
TABLESPACE_EXTENT_MAKE_USED	POSITIVE	8	Marks the block range (extent) as used in the bitmaps
SEGMENT_VERIFY_BASIC	POSITIVE	9	Performs the basic metadata checks
SEGMENT_VERIFY_DEEP	POSITIVE	10	Performs deep verification
SEGMENT_VERIFY_SPECIFIC	POSITIVE	11	Performs a specific check for the segment
HWM_CHECK	POSITIVE	12	Checks high water mark (HWM)
BMB_CHECK	POSITIVE	13	Checks integrity among L1, L2 and L3 BMBs (Bit Map Blocks)
SEG_DICT_CHECK	POSITIVE	14	Checks consistency of segment header with corresponding SEG entry
EXTENT_TS_BITMAP_CHECK	POSITIVE	15	Checks whether the tablespace bitmaps corresponding to the extent map are marked used
DB_BACKPOINTER_CHECK	POSITIVE	16	Checks whether the L1 BMBs, L2 BMBs, L3 BMBs and data blocks point to the same parent segment
EXTENT_SEGMENT_BITMAP_CHECK	POSITIVE	17	Checks whether the bitmap blocks are consistent with the extent map

**Table 134-1 (Cont.) DBMS\_SPACE\_ADMIN Constants**

<b>Constant</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
BITMAPS_CHECK	POSITIVE	18	Checks from the datablocks that the bitmap states representing the blocks are consistent
TS_VERIFY_BITMAPS	POSITIVE	19	Checks whether the tablespace bitmaps are consistent with the extents belonging to that tablespace
TS_VERIFY_DEEP	POSITIVE	20	Performs TS_VERIFY_BITMAPS and TS_VERIFY_SEGMENTS with DEEP option
TS_VERIFY_SEGMENTS	POSITIVE	21	Performs ASSM_SEGMENT_VERIFY on all segments in the tablespace, taking either the BASIC or the DEEP option
SEGMENTS_DUMP_BITMAP_SUMMARY	POSITIVE	27	Dumps only bitmap block summaries

## Operational Notes

Before migrating the `SYSTEM` tablespace, the following conditions must be met. These conditions are enforced by the `TABLESPACE_MIGRATE_TO_LOCAL` procedure, except for the cold backup.

- The database must have a default temporary tablespace that is not `SYSTEM`.
- Dictionary-managed tablespaces cannot have any rollback segments.
- A locally managed tablespace must have at least one online rollback segment. If you are using automatic undo management, then an undo tablespace must be online.
- All tablespaces—except the tablespace containing the rollback segment or the undo tablespace—must be read-only.
- You must have a cold backup of the database.
- The system must be in restricted mode.



---

## Summary of DBMS\_SPACE\_ADMIN Subprograms

**Table 134–2 DBMS\_SPACE\_ADMIN Package Subprograms**

Subprogram	Description
<a href="#">ASSM_SEGMENT_VERIFY Procedure</a> on page 134-8	Verifies segments created in ASSM (Automatic Segment-Space Management) tablespaces
<a href="#">ASSM_TABLESPACE_VERIFY Procedure</a> on page 134-10	Verifies ASSM tablespaces
<a href="#">DROP_EMPTY_SEGMENTS Procedure</a> on page 134-11	Drops segments from empty tables or table fragments and dependent objects
<a href="#">MATERIALIZER_DEFERRED_SEGMENTS Procedure</a> on page 134-12	Materializes segments for tables and table fragments with deferred segment creation and their dependent objects
<a href="#">SEGMENT_CORRUPT Procedure</a> on page 134-13	Marks the segment corrupt or valid so that appropriate error recovery can be done
<a href="#">SEGMENT_DROP_CORRUPT Procedure</a> on page 134-14	Drops a segment currently marked corrupt (without reclaiming space)
<a href="#">SEGMENT_DUMP Procedure</a> on page 134-15	Dumps the segment header and extent maps of a given segment
<a href="#">SEGMENT_VERIFY Procedure</a> on page 134-16	Verifies the consistency of the extent map of the segment
<a href="#">TABLESPACE_FIX_BITMAPS Procedure</a> on page 134-17	Marks the appropriate block range (extent) as free or used in bitmap
<a href="#">TABLESPACE_FIX_SEGMENT_STATES Procedure</a> on page 134-18	Fixes the state of the segments in a tablespace in which migration was aborted
<a href="#">TABLESPACE_MIGRATE_FROM_LOCAL Procedure</a> on page 134-19	Migrates a locally managed tablespace to dictionary-managed tablespace
<a href="#">TABLESPACE_MIGRATE_TO_LOCAL Procedure</a> on page 134-20	Migrates a tablespace from dictionary-managed format to locally managed format
<a href="#">TABLESPACE_REBUILD_BITMAPS Procedure</a> on page 134-21	Rebuilds the appropriate bitmaps
<a href="#">TABLESPACE_REBUILD_QUOTAS Procedure</a> on page 134-22	Rebuilds quotas for given tablespace
<a href="#">TABLESPACE_RELOCATE_BITMAPS Procedure</a> on page 134-23	Relocates the bitmaps to the destination specified
<a href="#">TABLESPACE_VERIFY Procedure</a> on page 134-24	Verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized

## ASSM\_SEGMENT\_VERIFY Procedure

Given a segment definition, the procedure verifies the basic consistency of the space metadata blocks as well as consistency between space metadata and segment data blocks. This procedure verifies segments created in Automatic Segment Space Management (ASSM) tablespaces.

There is however a difference between basic verification and deep verification:

- Basic verification involves consistency checks of space metadata, such as integrity among level 1, level 2, level 3 bitmap blocks, consistency of segment extent map and level 1 bitmap ranges.
- Deep verification involves consistency checks between datablocks and space metadata blocks such as whether the datablocks point correctly to the parent level 1 bitmap blocks, and whether the freeness states in the datablocks are consistent with the freeness states of bits in level 1 bitmap blocks corresponding to the datablocks.

### Syntax

```
DBMS_SPACE_ADMIN.ASSM_SEGMENT_VERIFY (
  segment_owner  IN VARCHAR2,
  segment_name   IN VARCHAR2,
  segment_type   IN VARCHAR2,
  partition_name IN VARCHAR2,
  verify_option  IN POSITIVE  DEFAULT SEGMENT_VERIFY_BASIC,
  attrib         IN POSITIVE  DEFAULT NULL);
```

### Parameters

**Table 134–3 ASSM\_SEGMENT\_VERIFY Procedure Parameters**

Parameter	Description
segment_owner	Schema that owns the segment
segment_name	Name of the segment to be verified
segment_type	Segment namespace is one of TABLE, TABLE PARTITION, TABLE SUBPARTITION, INDEX, INDEX PARTITION, INDEX SUBPARTITION, LOB, LOB PARTITION, LOB SUBPARTITION, CLUSTER
partition_name	Name of the partition or subpartition
verify_option	One of the following options: <ul style="list-style-type: none"> <li>■ SEGMENT_VERIFY_BASIC := 9. Performs deep verification</li> <li>■ SEGMENT_VERIFY_DEEP := 10. Performs the basic metadata checks (Default)</li> <li>■ SEGMENT_VERIFY_SPECIFIC := 11. Performs a specific check for the segment</li> </ul>

**Table 134-3 (Cont.) ASSM\_SEGMENT\_VERIFY Procedure Parameters**

Parameter	Description
attrib	<p>When option <code>SEGMENT_VERIFY_SPECIFIC</code> is specified as option, <code>attrib</code> can be one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>HWM_CHECK := 12</code>. Checks whether high water mark information is accurate</li> <li>■ <code>BMB_CHECK := 13</code>. Checks whether space bitmap blocks have correct backpointers to the segment header</li> <li>■ <code>SEG_DICT_CHECK := 14</code>. Checks whether dictionary information for segment is accurate</li> <li>■ <code>EXTENT_TS_BITMAP_CHECK := 15</code>. Checks whether extent maps are consistent with file level bitmaps</li> <li>■ <code>DB_BACKPOINTER_CHECK := 16</code>. Checks whether datablocks have correct backpointers to the space metadata blocks</li> <li>■ <code>EXTENT_SEGMENT_BITMAP_CHECK := 17</code>. Checks whether extent map in the segment matches the bitmaps in the segment</li> <li>■ <code>BITMAPS_CHECK := 18</code>. Checks whether space bitmap blocks are accurate</li> </ul>

### Usage Notes

- Using this procedure requires `SYSDBA` privileges.
- You can determine the relative file # and header block # (`header_relative_file` and `header_block` parameters) by querying `DBA_SEGMENTS`.
- This procedure outputs a dump file named `sid_ora_process_ID.trc` to the location specified in the `USER_DUMP_DEST` initialization parameter.

## ASSM\_TABLESPACE\_VERIFY Procedure

This procedure verifies all the segments created in an ASSM tablespace. The verification for each segment performs basic consistency checks of the space metadata blocks as well as consistency checks between space metadata and segment data blocks.

### Syntax

```
DBMS_SPACE_ADMIN.ASSM_TABLESPACE_VERIFY (
  tablespace_name  IN VARCHAR2,
  ts_option        IN POSITIVE,
  segment_option   IN POSITIVE DEFAULT NULL);
```

### Parameters

**Table 134–4 ASSM\_TABLESPACE\_VERIFY Procedure Parameters**

Parameter	Description
tablespace_name	Name of the tablespace to verify. The tablespace must be an ASSM tablespace.
ts_option	<ul style="list-style-type: none"> <li>■ TS_VERIFY_BITMAPS := 19. The bitmaps are verified against the extents. This detects bits that are marked used or free wrongly and detects multiple allocation of extents. The file metadata is validated against file\$ and control file.</li> <li>■ TS_VERIFY_DEEP := 20. This option is used to verify the file bitmaps as well perform checks on all the segments .</li> <li>■ TS_VERIFY_SEGMENTS := 21. This option is used to invoke SEGMENT_VERIFY on all the segments in the tablespace. Optionally you can write a script that queries all the segments in the tablespace and invoke SEGMENT_VERIFY.</li> </ul>
segment_option	<p>When TS_VERIFY_SEGMENTS is specified, segment_option can be one of the following:</p> <ul style="list-style-type: none"> <li>■ SEGMENT_VERIFY_BASIC := 9</li> <li>■ SEGMENT_VERIFY_DEEP := 10</li> </ul> <p>The value of segment_option is NULL when TS_VERIFY_DEEP or TS_VERIFY_BITMAPS is specified.</p>

### Usage Notes

- Using this procedure requires SYSDBA privileges.
- This procedure outputs a dump file named `sid_ora_process_ID.trc` to the location specified in the USER\_DUMP\_DEST initialization parameter.

## DROP\_EMPTY\_SEGMENTS Procedure

This procedure drops segments from empty tables or table fragments and dependent objects.

---



---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```
DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS (
  schema_name      IN      VARCHAR2  DEFAULT NULL,
  table_name       IN      VARCHAR2  DEFAULT NULL,
  partition_name   IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 134–5** *DROP\_EMPTY\_SEGMENTS Procedure Parameters*

Parameter	Description
schema_name	Name of schema
table_name	Name of table
partition_name	Name of partition

### Usage Notes

Given a schema name, this procedure scans all tables in the schema. For each table, if the table or any of its fragments are found to be empty, and the table satisfies certain criteria (restrictions being the same as those described in "Restrictions on Deferred Segment Creation"), then the empty table fragment and associated index segments are dropped along with the corresponding LOB data and index segments. A subsequent insert creates segments with the same properties.

Optionally:

- No schema\_name is specified, in which case tables belonging to all schemas are scanned
- Both schema\_name and table\_name are specified to perform the operation on a specified table
- All three arguments are supplied, restricting the operation to the partition and its dependent objects

## MATERIALIZED\_DEFERRED\_SEGMENTS Procedure

This procedure materializes segments for tables and table fragments with deferred segment creation and their dependent objects.

---



---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```
DBMS_SPACE_ADMIN.MATERIALIZED_DEFERRED_SEGMENTS (
  schema_name      IN      VARCHAR2  DEFAULT NULL,
  table_name       IN      VARCHAR2  DEFAULT NULL,
  partition_name   IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 134–6** *MATERIALIZED\_DEFERRED\_SEGMENTS Procedure Parameters*

Parameter	Description
schema_name	Name of schema
table_name	Name of table
partition_name	Name of partition

### Usage Notes

Given a schema name, this procedure scans all tables in the schema. For each table, if the deferred or delayed segment property is set for the table or any of its fragments, then a new segment is created for those fragments and their dependent objects.

Optionally:

- No schema\_name is specified, in which case tables belonging to all schemas are scanned
- Both schema\_name and table\_name are specified to perform the operation on a specified table
- All three arguments are supplied, restricting the operation to the partition and its dependent objects

## SEGMENT\_CORRUPT Procedure

This procedure marks the segment corrupt or valid so that appropriate error recovery can be done. It cannot be used on the SYSTEM tablespace.

### Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (
  tablespace_name      IN   VARCHAR2,
  header_relative_file IN   POSITIVE,
  header_block        IN   POSITIVE,
  corrupt_option       IN   POSITIVE DEFAULT SEGMENT_MARK_CORRUPT);
```

### Parameters

**Table 134–7** SEGMENT\_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides
header_relative_file	Relative file number of segment header
header_block	Block number of segment header
corrupt_option	SEGMENT_MARK_CORRUPT (default) or SEGMENT_MARK_VALID

### Usage Notes

You can determine the relative file number and block number (header\_relative\_file and header\_block parameter) of the segment header block by querying DBA\_SEGMENTS.

### Examples

The following example marks the segment as corrupt:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, DBMS_SPACE_ADMIN.SEGMENT_MARK_CORRUPT);
```

Alternately, the next example marks a corrupt segment valid:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, DBMS_SPACE_ADMIN.SEGMENT_MARK_VALID);
```

## SEGMENT\_DROP\_CORRUPT Procedure

This procedure drops a segment currently marked corrupt (without reclaiming space). For this to work, the segment must be marked *temporary*. To mark a corrupt segment as temporary, issue a DROP command on the segment.

### Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (  
  tablespace_name      IN    VARCHAR2,  
  header_relative_file IN    POSITIVE,  
  header_block         IN    POSITIVE);
```

### Parameters

**Table 134–8** SEGMENT\_DROP\_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides
header_relative_file	Relative file number of segment header
header_block	Block number of segment header

### Usage Notes

- The space for the segment is not released, and it must be fixed by using the [TABLESPACE\\_FIX\\_BITMAPS Procedure](#) or the [TABLESPACE\\_REBUILD\\_BITMAPS Procedure](#).
- The procedure cannot be used on the SYSTEM tablespace.
- You can determine the relative file number and block number (header\_relative\_file and header\_block parameter) of the segment header block by querying DBA\_SEGMENTS.

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT('USERS', 4, 33);
```



## SEGMENT\_DUMP Procedure

This procedure dumps the segment header and bitmap blocks of a specific segment to the location specified in the `USER_DUMP_DEST` initialization parameter.

### Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DUMP (
  tablespace_name      IN   VARCHAR2,
  header_relative_file IN   POSITIVE,
  header_block         IN   POSITIVE,
  dump_option          IN   POSITIVE DEFAULT SEGMENT_DUMP_EXTENT_MAP);
```

### Parameters

**Table 134–9** *SEGMENT\_DUMP Procedure Parameters*

Parameter	Description
<code>tablespace_name</code>	Name of tablespace in which segment resides
<code>header_relative_file</code>	Relative file number of segment header
<code>header_block</code>	Block number of segment header
<code>dump_option</code>	One of the following options: <ul style="list-style-type: none"> <li>■ <code>SEGMENT_DUMP_EXTENT_MAP</code></li> <li>■ <code>SEGMENT_DUMP_BITMAP_SUMMARY</code></li> </ul>

### Usage Notes

- You can produce a slightly abbreviated dump, which includes the segment header and bitmap block summaries, without percent-free states of each block if you pass `SEGMENT_DUMP_BITMAP_SUMMARY` as the `dump_option` parameter.
- You can determine the relative file number and block number (`header_relative_file` and `header_block` parameter) of the segment header block by querying `DBA_SEGMENTS`.

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('USERS', 4, 33);
```

## SEGMENT\_VERIFY Procedure

This procedure checks the consistency of the segment extent map with the tablespace file bitmaps.

### Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (
  tablespace_name      IN    VARCHAR2,
  header_relative_file IN    POSITIVE,
  header_block         IN    POSITIVE,
  verify_option        IN    POSITIVE DEFAULT SEGMENT_VERIFY_EXTENTS);
```

### Parameters

**Table 134–10** SEGMENT\_VERIFY Procedure Parameters

Parameters	Description
tablespace_name	Name of tablespace in which segment resides
header_relative_file	Relative file number of segment header
header_block	Block number of segment header
verify_option	What kind of check to do: SEGMENT_VERIFY_EXTENTS or SEGMENT_VERIFY_EXTENTS_GLOBAL

### Usage Notes

- Anomalies are output as block range, bitmap-block, bitmap-block-range, anomaly-information, in the trace file for all block ranges found to have incorrect space representation. The kinds of problems which would be reported are free space not considered free, used space considered free, and the same space considered used by multiple segments.
- You can determine the relative file number and block number (header\_relative\_file and header\_block parameter) of the segment header block by querying DBA\_SEGMENTS.

### Examples

The following example verifies that the segment with segment header at relative file number 4, block number 33, has its extent maps and bitmaps synchronized.

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_VERIFY('USERS', 4, 33, DBMS_SPACE_ADMIN.SEGMENT_VERIFY_EXTENTS);
```

## TABLESPACE\_FIX\_BITMAPS Procedure

This procedure marks the appropriate block range (extent) as free or used in bitmap. It cannot be used on the `SYSTEM` tablespace.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS (
  tablespace_name      IN   VARCHAR2,
  dbarange_relative_file IN  POSITIVE,
  dbarange_begin_block IN  POSITIVE,
  dbarange_end_block   IN  POSITIVE,
  fix_option           IN  POSITIVE);
```

### Parameters

**Table 134–11** TABLESPACE\_FIX\_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace
dbarange_relative_file	Relative file number of block range (extent)
dbarange_begin_block	Block number of beginning of extent
dbarange_end_block	Block number (inclusive) of end of extent
fix_option	One of the following options: <ul style="list-style-type: none"> <li>■ TABLESPACE_EXTENT_MAKE_FREE</li> <li>■ TABLESPACE_EXTENT_MAKE_USED</li> </ul>

### Examples

The following example marks bits for 51 blocks for relative file number 4, beginning at block number 33 and ending at 83, as `USED` in bitmaps.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('USERS', 4, 33, 83, DBMS_SPACE_
ADMIN.EXTENT_MAKE_USED);
```

Alternatively, specifying an option of `TABLESPACE_EXTENT_MAKE_FREE` marks the bits free in bitmaps. The `BEGIN` and `END` blocks must be in extent boundary and be extent multiple; otherwise, an error is raised.

## TABLESPACE\_FIX\_SEGMENT\_STATES Procedure

This procedure fixes the state of the segments in a tablespace in which migration was aborted. During tablespace migration to or from local, the segments are put in a transient state. If migration is aborted, then the segment states are corrected by SMON when event 10906 is set. A database with segments in such a transient state cannot be downgraded. The procedure can be used to fix the state of such segments.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES (  
    tablespace_name    IN    VARCHAR);
```

### Parameters

**Table 134–12** TABLESPACE\_FIX\_SEGMENT\_STATES Procedure Parameters

Parameter Name	Description
tablespace_name	Name of the tablespace whose segments must be fixed

### Usage Notes

The tablespace must be kept online and read/write when this procedure is called.

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES ('TS1');
```

## TABLESPACE\_MIGRATE\_FROM\_LOCAL Procedure

This procedure migrates a locally managed tablespace to a dictionary-managed tablespace.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (  
    tablespace_name      IN      VARCHAR2);
```

### Parameter

**Table 134–13** TABLESPACE\_MIGRATE\_FROM\_LOCAL Procedure Parameter

Parameter	Description
tablespace_name	Name of tablespace

### Usage Notes

The tablespace must be kept online and read/write during migration. Migration of temporary tablespaces and migration of SYSTEM tablespaces are not supported.

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');
```

## TABLESPACE\_MIGRATE\_TO\_LOCAL Procedure

This procedure migrates the tablespace from a dictionary-managed format to a locally managed format. Tablespaces migrated to locally managed format are user managed.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL (
    tablespace_name    IN    VARCHAR2,
    unit_size          IN    POSITIVE DEFAULT NULL,
    rfno                IN    POSITIVE DEFAULT NULL);
```

### Parameters

**Table 134–14 TABLESPACE\_MIGRATE\_TO\_LOCAL Procedure Parameters**

Parameter Name	Description
tablespace_name	Name of the tablespace to be migrated
unit_size	Bitmap unit size (which is the size of the smallest possible chunk of space that can be allocated) in the tablespace specified in number of blocks
rfno	Relative File Number of the file where the bitmap blocks are placed

### Usage Notes

- Before you migrate the `SYSTEM` tablespace, migrate any dictionary-managed tablespaces that you want to use in read/write mode to locally managed. After the `SYSTEM` tablespace is migrated, you cannot change dictionary-managed tablespaces to read/write.

**See Also:** *Oracle Database Administrator's Guide*

- The tablespace must be kept online and read/write during migration. Note that temporary tablespaces cannot be migrated.
- Allocation Unit may be specified optionally. The default is calculated by the system based on the highest common divisor of all extents (used or free) for the tablespace. This number is further trimmed based on the `MINIMUM EXTENT` for the tablespace (5 if `MINIMUM EXTENT` is not specified). Thus, the calculated value will not be larger than the `MINIMUM EXTENT` for the tablespace. The last free extent in every file is ignored for GCD calculation. If you specify the unit size, then it must be a factor of the `unit_size` calculated by the system; otherwise an error message is returned.
- The Relative File Number parameter is used to place the bitmaps in a desired file. If space is not found in the file, then an error is issued. The data file specified must be part of the tablespace being migrated. If the dataflow is not specified, then the system chooses a dataflow in which to place the initial bitmap blocks. If space is not found for the initial bitmaps, then an error is raised.

### Examples

To migrate a tablespace 'TS1' in 2KB blocksize with minimum extent size 1MB:

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('TS1', 512, 2);
```

The bitmaps are placed in file with relative file number 2.

## TABLESPACE\_REBUILD\_BITMAPS Procedure

This procedure rebuilds the appropriate bitmaps. If no bitmap block is specified, then it rebuilds all bitmaps for the given tablespace.

The procedure cannot be used on the SYSTEM tablespace.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS (
  tablespace_name      IN   VARCHAR2,
  bitmap_relative_file IN   POSITIVE  DEFAULT NULL,
  bitmap_block        IN   POSITIVE  DEFAULT NULL);
```

### Parameters

**Table 134–15 TABLESPACE\_REBUILD\_BITMAPS Procedure Parameters**

Parameter	Description
tablespace_name	Name of tablespace
bitmap_relative_file	Relative file number of bitmap block to rebuild
bitmap_block	Block number of bitmap block to rebuild

### Usage Notes

Only full rebuild is supported.

### Examples

The following example rebuilds bitmaps for all the files in the USERS tablespace.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');
```

## TABLESPACE\_REBUILD\_QUOTAS Procedure

This procedure rebuilds quotas for the given tablespace.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS (  
    tablespace_name          IN    VARCHAR2);
```

### Parameters

**Table 134–16** *TABLESPACE\_REBUILD\_QUOTAS Procedure Parameters*

Parameter	Description
tablespace_name	Name of tablespace

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS ('USERS');
```



## TABLESPACE\_RELOCATE\_BITMAPS Procedure

This procedure relocates the bitmaps to the destination specified.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS (
    tablespace_name    IN      VARCHAR2,
    filno              IN      POSITIVE,
    blkno              IN      POSITIVE);
```

### Parameters

**Table 134–17 TABLESPACE\_RELOCATE\_BITMAPS Procedure Parameters**

Parameter Name	Description
tablespace_name	Name of tablespace
filno	Relative File Number of the destination file
blkno	Block Number of the destination range

### Usage Notes

- Migration of a tablespace from dictionary-managed to locally managed format could result in the creation of SPACE HEADER segment that contains the bitmap blocks. The SPACE HEADER segment is treated as user data. If you explicitly resize a file at or below the space header segment, then an error is issued. Use the TABLESPACE\_RELOCATE\_BITMAPS command to move the control information to a different destination and then resize the file.
- This procedure cannot be used on the SYSTEM tablespace.
- The tablespace must be kept online and read/write during relocation of bitmaps. This can be done only on migrated locally managed tablespaces.

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS('TS1', 3, 4);
```

Moves the bitmaps to file 3, block 4.

---

**Note:** The source and the destination addresses must not overlap. The destination block number is rounded down to the unit boundary. If there is user data in that location, then an error is raised.

---

## TABLESPACE\_VERIFY Procedure

This procedure verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized.

### Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY (  
    tablespace_name      IN    VARCHAR2,  
    verify_option        IN    POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

### Parameters

**Table 134–18** TABLESPACE\_VERIFY Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace
verify_option	One option is supported: TABLESPACE_VERIFY_BITMAP

### Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY ('USERS');
```

The DBMS\_SPM package supports the SQL plan management feature by providing an interface for the DBA or other user to perform controlled manipulation of plan history and SQL plan baselines maintained for various SQL statements.

**See Also:** For more information about "Using SQL Plan Management" in the *Oracle Database Performance Tuning Guide*

This chapter contains the following topics:

- [Using DBMS\\_SPM](#)
  - Overview
  - Security Model
  - Constants
  - Examples
- [Data Structures](#)
- [Summary of DBMS\\_SPM Subprograms](#)

## Using DBMS\_SPM

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Examples](#)

## Overview

The DBMS\_SPM package allows the user to manage SQL execution plans using SQL plan management. SQL plan management prevents performance regressions resulting from sudden changes to the execution plan of a SQL statement by recording and evaluating the execution plans of SQL statements over time, and builds SQL plan baselines composed of a set of existing plans known to be efficient. The SQL plan baselines are then used to preserve performance of corresponding SQL statements, regardless of changes occurring in the system. Common usage scenarios where SQL plan management can improve or preserve SQL performance include:

- A database upgrade that installs a new optimizer version usually results in plan changes for a small percentage of SQL statements, with most of the plan changes resulting in either no performance change or improvement. However, certain plan changes may cause performance regressions. The use of SQL plan baselines significantly minimizes potential performance regressions resulting from a database upgrade.
- Ongoing system and data changes can impact plans for some SQL statements, potentially causing performance regressions. The use of SQL plan baselines helps to minimize performance regressions and stabilize SQL performance.
- Deployment of new application modules means introducing new SQL statements into the system. The application software may use appropriate SQL execution plans developed under a standard test configuration for the new SQL statements. If the system production configuration differs significantly from the test configuration, SQL plan baselines can be evolved over time to produce better performance.

## Security Model

The package is owned by SYS. The EXECUTE package privilege is required to execute its procedures. Any user granted the ADMINISTER SQL MANAGEMENT OBJECT privilege is able to execute the DBMS\_SPM package.

## Constants

The DBMS\_SPM package uses the constants shown in [Table 135-1](#), "DBMS\_SPM Constants". These constants are defined as standard input for the `time_limit` parameter of the [EVOLVE\\_SQL\\_PLAN\\_BASELINE](#) Function.

**Table 135-1 DBMS\_SPM Constants**

Constant	Type	Value	Description
AUTO_LIMIT	INTEGER	2147483647	Oracle determines the appropriate time spent by the <a href="#">EVOLVE_SQL_PLAN_BASELINE</a> Function.
NO_LIMIT	INTEGER	2147483647 -1	There is no limit to the time spent by the <a href="#">EVOLVE_SQL_PLAN_BASELINE</a> Function.

## Examples

Detailed examples are located under the following topics:

- [Migrating Stored Outlines to SQL Plan Baselines](#)
- [Migrating Outlines to Utilize SQL Plan Management Features](#)
- [Migrating Outlines to Preserve Stored Outline Behavior](#)
- [Performing Follow-Up Tasks After Stored Outline Migration](#)



## Data Structures

---

The DBMS\_SPM package defines a TABLE type.

### Table Types

- [NAMELIST Table Type](#)

## NAMELIST Table Type

This type allows for a list of names as an input parameter.

### Syntax

```
TYPE name_list IS TABLE OF VARCHAR2(30);
```

---

## Summary of DBMS\_SPM Subprograms

This table lists the package subprograms in alphabetical order.

**Table 135–2 DBMS\_SPM Package Subprograms**

Subprogram	Description
<a href="#">ALTER_SQL_PLAN_BASELINE Function</a> on page 135-10	Changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format
<a href="#">CONFIGURE Procedure</a> on page 135-12	Sets configuration options for SQL management base, in parameter/value format
<a href="#">CREATE_STGTAB_BASELINE Procedure</a> on page 135-13	Creates a staging table that used for transporting SQL plan baselines from one system to another
<a href="#">DROP_SQL_PLAN_BASELINE Function</a> on page 135-14	drops a single plan, or all plans associated with a SQL statement
<a href="#">EVOLVE_SQL_PLAN_BASELINE Function</a> on page 135-15	Evolves SQL plan baselines associated with one or more SQL statements
<a href="#">LOAD_PLANS_FROM_CURSOR_CACHE Functions</a> on page 135-17	Loads one or more plans present in the cursor cache for a SQL statement
<a href="#">LOAD_PLANS_FROM_SQLSET Function</a> on page 135-19	Loads plans stored in a SQL tuning set (STS) into SQL plan baselines
<a href="#">MIGRATE_STORED_OUTLINE Functions</a> on page 135-20	Migrates existing stored outlines to SQL plan baselines
<a href="#">PACK_STGTAB_BASELINE Function</a> on page 135-22	Packs (exports) SQL plan baselines from SQL management base into a staging table
<a href="#">UNPACK_STGTAB_BASELINE Function</a> on page 135-23	Unpacks (imports) SQL plan baselines from a staging table into SQL management base

## ALTER\_SQL\_PLAN\_BASELINE Function

This function changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format.

### Syntax

```
DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
  sql_handle      IN VARCHAR2 := NULL,
  plan_name       IN VARCHAR2 := NULL,
  attribute_name  IN VARCHAR2,
  attribute_value IN VARCHAR2)
RETURN PLS_INTEGER;
```

### Parameters

**Table 135–3 ALTER\_SQL\_PLAN\_BASELINE Function Parameters**

Parameter	Description
sql_handle	SQL statement handle. It identifies plans associated with a SQL statement for an attribute change. If NULL then plan_name must be specified.
plan_name	Plan name. It identifies a specific plan. Default NULL means set the attribute for all plans associated with a SQL statement identified by sql_handle. If NULL then sql_handle must be specified.
attribute_name	Name of plan attribute to set (see table below).
attribute_value	Value of plan attribute to use (see table below)

**Table 135–4 Names & Values for ALTER\_SQL\_PLAN\_BASELINE Function Parameters**

Name	Description	Possible Values
enabled	'YES' means the plan is available for use by the optimizer. It may or may not be used depending on accepted status.	'YES' or 'NO'
fixed	'YES' means the SQL plan baseline is not evolved over time. A fixed plan takes precedence over a non-fixed plan.	'YES' or 'NO'
autopurge	'YES' means the plan is purged if it is not used for a time period. 'NO' means it is never purged.	'YES' or 'NO'
plan_name	Name of the plan	String of up to 30-characters
description	Plan description.	String of up to 500-characters

### Return Values

The number of plans altered.

**Usage Notes**

When a single plan is specified, one of various statuses, or plan name, or description can be altered. When all plans for a SQL statement are specified, one of various statuses, or description can be altered. This function can be called numerous times, each time setting a different plan attribute of same plan(s) or different plan(s).

## CONFIGURE Procedure

This procedure sets configuration options for SQL management base, in parameter/value format. This function can be called numerous times, each time setting a different configuration option.

### Syntax

```
DBMS_SPM.CONFIGURE (
  parameter_name  IN VARCHAR2,
  parameter_value IN NUMBER);
```

### Parameters

**Table 135–5 CONFIGURE Procedure Parameters**

Parameter	Description
parameter_name	Name of parameter to set (see table below)
parameter_value	Value of parameter to use (see table below)

**Table 135–6 Names & Values for CONFIGURE Procedure Parameters**

Name	Description	Possible Values	Default Value
space_budget_percent	Maximum percent of SYSAUX space that can be used for SQL management base	1, 2, ..., 50	10
plan_retention_weeks	Number of weeks to retain unused plans before they are purged	5, 6, ..., 523	53

### Usage Notes

- The default space budget for SQL management base is no more than ten percent of the size of SYSAUX tablespace. The space budget can be set to a maximum of 50%. The default unused plan retention period is one year and one week, which means a plan will be automatically purged if it has not been used for more than a year. The retention period can be set to a maximum of 523 weeks (i.e. a little over 10 years).
- When the space occupied by SQL management base exceeds the defined space budget limit, a weekly database alert is generated.

## CREATE\_STGTAB\_BASELINE Procedure

This procedure creates a staging table used for transporting SQL plan baselines from one system to another.

### Syntax

```
DBMS_SPM.CREATE_STGTAB_BASELINE (
  table_name      IN VARCHAR2,
  table_owner     IN VARCHAR2 := NULL,
  tablespace_name IN VARCHAR2 := NULL);
```

### Parameters

**Table 135–7 CREATE\_STGTAB\_BASELINE Procedure Parameters**

Parameter	Description
table_name	Name of staging table to create for the purpose of packing and unpacking SQL plan baselines
table_owner	Name of owner of the staging table. Default NULL means current schema is the table owner.
tablespace_name	Name of tablespace. Default NULL means create staging table in the default tablespace.

### Usage Notes

The creation of staging table is the first step. To migrate SQL plan baselines from one system to another, the user/DBA has to perform a series of steps as follows:

1. Create a staging table in the source system
2. Select SQL plan baselines in the source system and pack them into the staging table
3. Export staging table into a flat file using Oracle EXP utility or Data Pump
4. Transfer flat file to the target system
5. Import staging table from the flat file using Oracle IMP utility or Data Pump
6. Select SQL plan baselines from the staging table and unpack them into the target system

## DROP\_SQL\_PLAN\_BASELINE Function

This function drops a single plan, or all plans associated with a SQL statement.

### Syntax

```
DBMS_SPM.DROP_SQL_PLAN_BASELINE (  
    sql_handle      IN VARCHAR2 := NULL,  
    plan_name      IN VARCHAR2 := NULL)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 135–8** DROP\_SQL\_PLAN\_BASELINE Function Parameters

Parameter	Description
sql_handle	SQL statement handle. It identifies plans associated with a SQL statement that are to be dropped. If NULL then plan_name must be specified.
plan_name	Plan name. It identifies a specific plan. Default NULL means to drop all plans associated with the SQL statement identified by sql_handle.

### Return Values

The number of plans dropped



## EVOLVE\_SQL\_PLAN\_BASELINE Function

This function evolves SQL plan baselines associated with one or more SQL statements. A SQL plan baseline is evolved when one or more of its non-accepted plans is changed to an accepted plan or plans. If interrogated by the user (parameter `verify = 'YES'`), the execution performance of each non-accepted plan is compared against the performance of a plan chosen from the associated SQL plan baseline. If the non-accepted plan performance is found to be better than SQL plan baseline performance, the non-accepted plan is changed to an accepted plan provided such action is permitted by the user (parameter `commit = 'YES'`).

The second form of the function employs a plan list format.

### Syntax

```
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (
  sql_handle   IN VARCHAR2 := NULL,
  plan_name    IN VARCHAR2 := NULL,
  time_limit   IN INTEGER   := DBMS_SPM.AUTO_LIMIT,
  verify       IN VARCHAR2 := 'YES',
  commit       IN VARCHAR2 := 'YES')
RETURN CLOB;
```

```
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (
  plan_list    IN DBMS_SPM.NAME_LIST,
  time_limit   IN INTEGER   := DBMS_SPM.AUTO_LIMIT,
  verify       IN VARCHAR2 := 'YES',
  commit       IN VARCHAR2 := 'YES')
RETURN CLOB;
```

### Parameters

**Table 135–9** *EVOLVE\_SQL\_PLAN\_BASELINE Function Parameters*

Parameter	Description
<code>sql_handle</code>	SQL statement identifier. Unless <code>plan_name</code> is specified, <code>NULL</code> means to consider all statements with non-accepted plans in their SQL plan baselines.
<code>plan_name</code>	Plan identifier. Default <code>NULL</code> means to consider all non-accepted plans in the SQL plan baseline of either the identified SQL statement or all SQL statements if <code>sql_handle</code> is <code>NULL</code> .
<code>plan_list</code>	A list of plan names. Each plan in the list can belong to same or different SQL statement.
<code>time_limit</code>	Time limit in number of minutes. This applies only if <code>verify = 'YES'</code> . The time limit is global and it is used as follows: The time limit for first non-accepted plan verification is set equal to the input value; the time limit for second non-accepted plan verification is set equal to (input value - time spent in first plan verification); and so on. <ul style="list-style-type: none"> <li>■ <code>DBMS_SPM.AUTO_LIMIT</code> (Default) lets the system choose an appropriate time limit based on the number of plan verifications required to be done.</li> <li>■ <code>DBMS_SPM.NO_LIMIT</code> means there is no time limit.</li> <li>■ A positive integer value represents a user specified time limit.</li> </ul>

**Table 135–9 (Cont.) EVOLVE\_SQL\_PLAN\_BASELINE Function Parameters**

Parameter	Description
<code>verify</code>	<p>Specifies whether to execute the plans and compare the performance before changing non-accepted plans into accepted plans. A performance verification involves executing a non-accepted plan and a plan chosen from corresponding SQL plan baseline and comparing their performance statistics. If non-accepted plan shows performance improvement, it is changed to an accepted plan.</p> <ul style="list-style-type: none"> <li>▪ 'YES' (Default) - verifies that a non-accepted plan gives better performance before changing it to an accepted plan</li> <li>▪ 'NO' - directs not to execute plans but only to change non-accepted plans into accepted plans</li> </ul>
<code>commit</code>	<p>Specifies whether to update the ACCEPTED status of non-accepted plans from 'NO' to 'YES'.</p> <ul style="list-style-type: none"> <li>▪ 'YES' (Default) - perform updates of qualifying non-accepted plans and generate a report that shows the updates and the result of performance verification when <code>verify = 'YES'</code>.</li> <li>▪ 'NO' - generate a report without any updates. Note that <code>commit = 'NO'</code> together with <code>verify = 'NO'</code> represents a no-op.</li> </ul>

## Return Values

A CLOB containing a formatted text report showing non-accepted plans in sequence, each with a possible change of its ACCEPTED status, and if `verify = 'YES'` the result of their performance verification.

## Usage Notes

Invoking this subprogram requires the ADMINISTER SQL MANAGEMENT OBJECT privilege.

## LOAD\_PLANS\_FROM\_CURSOR\_CACHE Functions

This function loads one or more plans present in the cursor cache for a SQL statement, or a set of SQL statements. It has four overloads: using SQL statement text, using SQL handle, using SQL ID, or using attribute\_name and attribute\_value pair.

### Syntax

```
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
  sql_id          IN VARCHAR2,
  plan_hash_value IN NUMBER   := NULL,
  sql_text       IN CLOB,
  fixed         IN VARCHAR2  := 'NO',
  enabled       IN VARCHAR2  := 'YES')
RETURN PLS_INTEGER;
```

```
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
  sql_id          IN VARCHAR2,
  plan_hash_value IN NUMBER   := NULL,
  sql_handle     IN VARCHAR2,
  fixed         IN VARCHAR2  := 'NO',
  enabled       IN VARCHAR2  := 'YES')
RETURN PLS_INTEGER;
```

```
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
  sql_id          IN VARCHAR2,
  plan_hash_value IN NUMBER   := NULL,
  fixed         IN VARCHAR2  := 'NO',
  enabled       IN VARCHAR2  := 'YES')
RETURN PLS_INTEGER;
```

```
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
  attribute_name IN VARCHAR2,
  attribute_value IN VARCHAR2,
  fixed         IN VARCHAR2  := 'NO',
  enabled       IN VARCHAR2  := 'YES')
RETURN PLS_INTEGER;
```

### Parameters

**Table 135–10** LOAD\_PLANS\_FROM\_CURSOR\_CACHE Function Parameters

Parameter	Description
sql_id	SQL statement identifier. Identifies a SQL statement in the cursor cache. Note: In the third overload the text of identified SQL statement is extracted from cursor cache and is used to identify the SQL plan baseline into which the plan(s) are loaded. If the SQL plan baseline doesn't exist it is created.
plan_hash_value	Plan identifier. Default NULL means capture all plans present in the cursor cache for the SQL statement identified by SQL_ID.
sql_text	SQL text to use in identifying the SQL plan baseline into which the plans are loaded. If the SQL plan baseline does not exist, it is created. The use of text is crucial when the user tunes a SQL statement by adding hints to its text and then wants to load the resulting plan(s) into the SQL plan baseline of the original SQL statement.

**Table 135–10 (Cont.) LOAD\_PLANS\_FROM\_CURSOR\_CACHE Function Parameters**

Parameter	Description
<code>sql_handle</code>	SQL handle to use in identifying the SQL plan baseline into which the plans are loaded. The <code>sql_handle</code> must denote an existing SQL plan baseline. The use of handle is crucial when the user tunes a SQL statement by adding hints to its text and then wants to load the resulting plan(s) into the SQL plan baseline of the original SQL statement.
<code>fixed</code>	Default 'NO' means the loaded plans are used as non-fixed plans. Value 'YES' means the loaded plans are used as fixed plans and the SQL plan baseline will not be evolved over time.
<code>attribute_name</code>	One of possible attribute names: <ul style="list-style-type: none"> <li>▪ <code>SQL_TEXT</code></li> <li>▪ <code>PARSING_SCHEMA_NAME</code></li> <li>▪ <code>MODULE</code></li> <li>▪ <code>ACTION</code></li> </ul>
<code>attribute_value</code>	Attribute value is used as a search pattern of LIKE predicate if attribute name is <code>SQL_TEXT</code> . Otherwise, it is used as an equality search value. (for example, for specifying <code>attribute_name =&gt; 'SQL_TEXT'</code> , and <code>attribute_value =&gt; '% HR-123 %'</code> means applying <code>SQL_TEXT LIKE '% HR-123 %'</code> as a selection filter. Similarly, specifying <code>attribute_name =&gt; 'MODULE'</code> , and <code>attribute_value =&gt; 'HR'</code> means applying <code>'MODULE = 'HR'</code> as a plan selection filter). The attribute value is upper-cased except when it is enclosed in double quotes or attribute name is <code>SQL_TEXT</code> .
<code>enabled</code>	Default 'YES' means the loaded plans are enabled for use by the optimizer

**Return Values**

Number of plans loaded

**Usage Notes**

Invoking this subprogram requires the ADMINISTER SQL MANAGEMENT OBJECT privilege.

## LOAD\_PLANS\_FROM\_SQLSET Function

This function loads plans stored in a SQL tuning set (STS) into SQL plan baselines. The plans loaded from STS are not verified for performance but added as accepted plans to existing or new SQL plan baselines. This function can be used to seed SQL management base with new SQL plan baselines.

### Syntax

```
DBMS_SPM.LOAD_PLANS_FROM_SQLSET (
  sqlset_name      IN  VARCHAR2,
  sqlset_owner     IN  VARCHAR2 := NULL,
  basic_filter     IN  VARCHAR2 := NULL,
  fixed            IN  VARCHAR2 := 'NO',
  enabled          IN  VARCHAR2 := 'YES'
  commit_rows     IN  NUMBER   := 1000)
RETURN PLS_INTEGER;
```

### Parameters

**Table 135–11** *LOAD\_PLANS\_FROM\_SQLSET Function Parameters*

Parameter	Description
sqlset_name	Name of the STS from where the plans are loaded into SQL plan baselines
sqlset_owner	Owner of STS. NULL means current schema is the owner.
basic_filter	A filter applied to the STS to select only qualifying plans to be loaded. The filter can take the form of any WHERE clause predicate that can specified against the view DBA_SQLSET_STATEMENTS. For example <code>basic_filter =&gt; 'sql_text like 'select /*LOAD_STS*/%'</code> or <code>basic_filter =&gt; 'sql_id="b62q7nc33gzwx"'</code> .
fixed	Default 'NO' means the loaded plans are used as non-fixed plans. Value 'YES' means the loaded plans are used as fixed plans and the SQL plan baseline will not be evolved over time.
enabled	Default 'YES' means the loaded plans are enabled for use by the optimizer
commit_rows	Number of SQL plans to load before doing a periodic commit. This helps to shorten the undo log.

### Return Values

The number of plans loaded

### Usage Notes

- To load plans from a remote system, first load the plans into an STS on the remote system, export/import the STS from remote to local system, and then use this function.
- To load plans from Automatic Workload Repository (AWR), first load the plans stored in AWR snapshots into an STS, and then use this procedure.
- The user can also capture plans resident in the cursor cache for one or more SQL statements into an STS, and then use this procedure.

## MIGRATE\_STORED\_OUTLINE Functions

This function migrates stored outlines for one or more SQL statements to plan baselines in the SQL management base (SMB). Users can specify which stored outline(s) to be migrated based on outline name, SQL text, or outline category, or migrate all stored outlines in the system to SQL plan baselines.

This second overload of the function migrates stored outlines for one or more SQL statements to plan baselines in the SQL management base (SMB) given one or more outline names.

### Syntax

```
DBMS_SPM.MIGRATE_STORED_OUTLINE (
    attribute_name    IN  VARCHAR2,
    attribute_value   IN  CLOB,
    fixed            IN  VARCHAR2 := 'NO')
RETURN CLOB;

DBMS_SPM.MIGRATE_STORED_OUTLINE (
    outln_list       IN  DBMS_SPM.NAME_LIST,
    fixed            IN  VARCHAR2 := 'NO')
RETURN CLOB;
```

### Parameters

**Table 135–12** *MIGRATE\_STORED\_OUTLINE Function Parameters*

Parameter	Description
<code>attribute_name</code>	Specifies the type of parameter used in <code>attribute_value</code> to identify the migrated stored outlines. It is case insensitive. Possible values: <ul style="list-style-type: none"> <li>▪ <code>outline_name</code></li> <li>▪ <code>sql_text</code></li> <li>▪ <code>category</code></li> <li>▪ <code>all</code></li> </ul>
<code>attribute_value</code>	Based on <code>attribute_name</code> , this can be: <ul style="list-style-type: none"> <li>▪ Name of stored outline to be migrated</li> <li>▪ SQL text of stored outlines to be migrated</li> <li>▪ Category of stored outlines to be migrated</li> <li>▪ NULL if <code>attribute_name</code> is <code>all</code></li> </ul>
<code>fixed</code>	NO (default) or YES. Specifies the "fixed" status of the plans generated during migration. By default, plans are generated as "non-fixed" plans.
<code>outln_list</code>	List of outline names to be migrated

### Return Values

A CLOB containing a formatted report to describe the statistics during the migration, including:

- Number of stored outlines successfully migrated

- Number of stored outlines (and also the corresponding outline names) failed to be migrated and the reasons for the failure

### Usage Note

- When the user specifies an outline name, the function migrates stored outlines to plan baseline based on given outline name, which uniquely identifies a single stored outline to be migrated.
- When the user specifies SQL text, the function migrates all stored outlines created for a given SQL statement. A single SQL statement can have multiple stored outlines created for it under different category names. One plan baseline plan is created for each stored outline. The new plan baselines have category names set to `DEFAULT`. The module name of a plan baseline is set to be the same as the stored outline.
- When the user specifies a category name, the function migrates all stored outlines with the given category name. Only one stored outline exists per category per SQL statement. One plan baseline is created for each stored outline.
- When user specifies to migrate `all`, the function migrates all stored outlines in the system to plan baselines. One plan baseline is created for each stored outline.

## PACK\_STGTAB\_BASELINE Function

This function packs (exports) SQL plan baselines from SQL management base into a staging table.

### Syntax

```
DBMS_SPM.PACK_STGTAB_BASELINE (
    table_name      IN VARCHAR2,
    table_owner     IN VARCHAR2 := NULL,
    sql_handle      IN VARCHAR2 := NULL,
    plan_name       IN VARCHAR2 := NULL,
    sql_text        IN CLOB      := NULL,
    creator         IN VARCHAR2 := NULL,
    origin          IN VARCHAR2 := NULL,
    enabled         IN VARCHAR2 := NULL,
    accepted        IN VARCHAR2 := NULL,
    fixed           IN VARCHAR2 := NULL,
    module          IN VARCHAR2 := NULL,
    action          IN VARCHAR2 := NULL)
RETURN NUMBER;
```

### Parameters

**Table 135–13** *PACK\_STGTAB\_BASELINE Function Parameters*

Parameter	Description
table_name	Name of staging table into which SQL plan baselines are packed (case insensitive unless double quoted)
table_owner	Name of staging table owner. Default NULL means current schema is the table owner
sql_handle	SQL handle (case sensitive)
plan_name	Plan name (case sensitive, % wildcards accepted)
sql_text	SQL text string (case sensitive, % wildcards accepted)
creator	Creator of SQL plan baseline (case insensitive unless double quoted)
origin	Origin of SQL plan baseline, should be 'MANUAL-LOAD', 'AUTO-CAPTURE', 'MANUAL_SQLTUNE' or 'AUTO-QLTUNE' (case insensitive)
enabled	Must be 'YES' or 'NO' (case insensitive)
accepted	Must be 'YES' or 'NO' (case insensitive)
fixed	Must be 'YES' or 'NO' (case insensitive)
module	Module (case sensitive)
action	Action (case sensitive)

### Return Values

Number of SQL plan baselines packed



## UNPACK\_STGTAB\_BASELINE Function

This function unpacks (imports) SQL plan baselines from a staging table into SQL management base.

### Syntax

```
DBMS_SPM.UNPACK_STGTAB_BASELINE (
    table_name      IN VARCHAR2,
    table_owner     IN VARCHAR2 := NULL,
    sql_handle      IN VARCHAR2 := NULL,
    plan_name       IN VARCHAR2 := NULL,
    sql_text        IN CLOB      := NULL,
    creator         IN VARCHAR2 := NULL,
    origin          IN VARCHAR2 := NULL,
    enabled         IN VARCHAR2 := NULL,
    accepted        IN VARCHAR2 := NULL,
    fixed           IN VARCHAR2 := NULL,
    module          IN VARCHAR2 := NULL,
    action          IN VARCHAR2 := NULL)
RETURN NUMBER;
```

### Parameters

**Table 135–14 UNPACK\_STGTAB\_BASELINE Function Parameters**

Parameter	Description
table_name	Name of staging table from which SQL plan baselines are unpacked (case insensitive unless double quoted)
table_owner	Name of staging table owner. Default NULL means current schema is the table owner
sql_handle	SQL handle (case sensitive)
plan_name	Plan name (case sensitive, % wildcards accepted)
sql_text	SQL text string (case sensitive, % wildcards accepted)
creator	Creator of SQL plan baseline (case insensitive unless double quoted)
origin	Origin of SQL plan baseline, should be 'MANUAL-LOAD', 'AUTO-CAPTURE', 'MANUAL_SQLTUNE' or 'AUTO-SQLTUNE' (case insensitive)
enabled	Must be 'YES' or 'NO' (case insensitive)
accepted	Must be 'YES' or 'NO' (case insensitive)
fixed	Must be 'YES' or 'NO' (case insensitive)
module	Module (case sensitive)
action	Action (case sensitive)

### Return Values

Number of plans unpacked



The `DBMS_SQL` package provides an interface to use dynamic SQL to parse any data manipulation language (DML) or data definition language (DDL) statement using PL/SQL. For example, you can enter a `DROP TABLE` statement from within a stored procedure by using the `PARSE` procedure supplied with the `DBMS_SQL` package.

**See Also:** For more information on native dynamic SQL, see *Oracle Database PL/SQL Language Reference*.

This chapter contains the following topics:

- [Using DBMS\\_SQL](#)
  - Overview
  - Security Model
  - Constants
  - Exceptions
  - Operational Notes
  - Examples
- [Data Structures](#)
  - RECORD TYPES
  - TABLE TYPES
- [Summary of DBMS\\_SQL Subprograms](#)

## Using DBMS\_SQL

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Exceptions](#)
- [Operational Notes](#)
- [Examples](#)

## Overview

Oracle lets you write stored procedures and anonymous PL/SQL blocks that use dynamic SQL. Dynamic SQL statements are not embedded in your source program; rather, they are stored in character strings that are input to, or built by, the program at runtime. This enables you to create more general-purpose procedures. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.

Native Dynamic SQL is an alternative to DBMS\_SQL that lets you place dynamic SQL statements directly into PL/SQL blocks. In most situations, Native Dynamic SQL is easier to use and performs better than DBMS\_SQL. However, Native Dynamic SQL itself has certain limitations:

- There is no support for so-called Method 4 (for dynamic SQL statements with an unknown number of inputs or outputs)

Also, there are some tasks that can only be performed using DBMS\_SQL.

The ability to use dynamic SQL from within stored procedures generally follows the model of the Oracle Call Interface (OCI).

**See Also:** *Oracle Call Interface Programmer's Guide*

PL/SQL differs somewhat from other common programming languages, such as C. For example, addresses (also called pointers) are not user-visible in PL/SQL. As a result, there are some differences between the Oracle Call Interface and the DBMS\_SQL package. These differences include the following:

- The OCI uses bind by address, while the DBMS\_SQL package uses bind by value.
- With DBMS\_SQL you must call VARIABLE\_VALUE to retrieve the value of an OUT parameter for an anonymous block, and you must call COLUMN\_VALUE after fetching rows to actually retrieve the values of the columns in the rows into your program.
- The current release of the DBMS\_SQL package does not provide CANCEL cursor procedures.
- Indicator variables are not required, because NULLs are fully supported as values of a PL/SQL variable.

A sample usage of the DBMS\_SQL package follows. For users of the Oracle Call Interface, this code should seem fairly straightforward.

## Security Model

DBMS\_SQL is a SYS-owned package compiled with AUTHID CURRENT\_USER. Any DBMS\_SQL subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

**See Also:** *Oracle Database PL/SQL Language Reference* for more information about using Invoker Rights or Definer Rights

In Oracle Database 11g, Release 1 (11.1), Oracle introduces a number of enhancements to DBMS\_SQL to improve the security of the package.

### Preventing Malicious or Accidental Access of Open Cursor Numbers

An error, ORA-29471, is raised when any DBMS\_SQL subprogram is called with a cursor number that does not denote an open cursor. When the error is raised, an alert is issued to the alert log and DBMS\_SQL becomes inoperable for the life of the session.

If the actual value for the cursor number in a call to the [IS\\_OPEN Function](#) denotes a cursor currently open in the session, the return value is TRUE. If the actual value is NULL, then the return value is FALSE. Otherwise, this raises an ORA-29471 error.

Note that the [OPEN\\_CURSOR Function](#) is the only DBMS\_SQL subprogram that has no formal parameter for the cursor number; rather, it returns a cursor number. Therefore it is not within the scope of these rules.

### Preventing Inappropriate Use of a Cursor

Cursors are protected from security breaches that subvert known existing cursors.

Checks are made when binding and executing. Optionally, checks may be performed for every single DBMS\_SQL subprogram call. The check is:

- The `current_user` is the same on calling the subprogram as it was on calling the most recent parse.
- The enabled roles on calling the subprogram must be a superset of the enabled roles on calling the most recent parse.

Consistent with the use of definer's rights subprograms, roles do not apply.

If either check fails, and ORA-29470 error is raised.

The mechanism for defining when checks are performed is a new overload for the `OPEN_CURSOR` subprogram which takes a formal parameter, `security_level`, with allowed values NULL, 1 and 2.

- When `security_level = 1` (or is NULL), the checks are made only when binding and executing.
- When `security_level = 2`, the checks are always made.

### Upgrade Considerations

This security regime is stricter than those in Oracle Database 10g, Release 2 (10.2) and previous releases. As a consequence, users of DBMS\_SQL may encounter runtime errors on upgrade. While these security enhancements make for more secure applications, users may wish to relax the security checks temporarily as they migrate to Oracle Database 11g, Release 1 (11.1). If so, please consult with Oracle Support on steps to relax the above security restrictions.

## Constants

The constants described in [Table 136-1](#) are used with the `language_flag` parameter of the [PARSE Procedures](#).

**Table 136-1** *DBMS\_SQL Constants*

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
V6	INTEGER	0	Specifies Oracle database version 6 behavior
NATIVE	INTEGER	1	Specifies normal behavior for the database to which the program is connected
V7	INTEGER	2	Specifies Oracle database version 7 behavior

## Exceptions

```
inconsistent_type EXCEPTION;  
  pragma exception_init(inconsistent_type, -6562);
```

This exception is raised by the [COLUMN\\_VALUE Procedure](#) or the [VARIABLE\\_VALUE Procedures](#) when the type of the given OUT parameter (for where to put the requested value) is different from the type of the value.



## Operational Notes

- [Execution Flow](#)
- [Processing Queries](#)
- [Processing Updates, Inserts, and Deletes](#)
- [Locating Errors](#)

### Execution Flow

1. [OPEN\\_CURSOR](#)
2. [PARSE](#)
3. [BIND\\_VARIABLE](#) or [BIND\\_ARRAY](#)
4. [DEFINE\\_COLUMN](#), [DEFINE\\_COLUMN\\_LONG](#), or [DEFINE\\_ARRAY](#)
5. [EXECUTE](#)
6. [FETCH\\_ROWS](#) or [EXECUTE\\_AND\\_FETCH](#)
7. [VARIABLE\\_VALUE](#), [COLUMN\\_VALUE](#), or [COLUMN\\_VALUE\\_LONG](#)
8. [CLOSE\\_CURSOR](#)

#### **OPEN\_CURSOR**

To process a SQL statement, you must have an open cursor. When you call the [OPEN\\_CURSOR Function](#), you receive a cursor ID number for the data structure representing a valid cursor maintained by Oracle. These cursors are distinct from cursors defined at the precompiler, OCI, or PL/SQL level, and are used only by the DBMS\_SQL package.

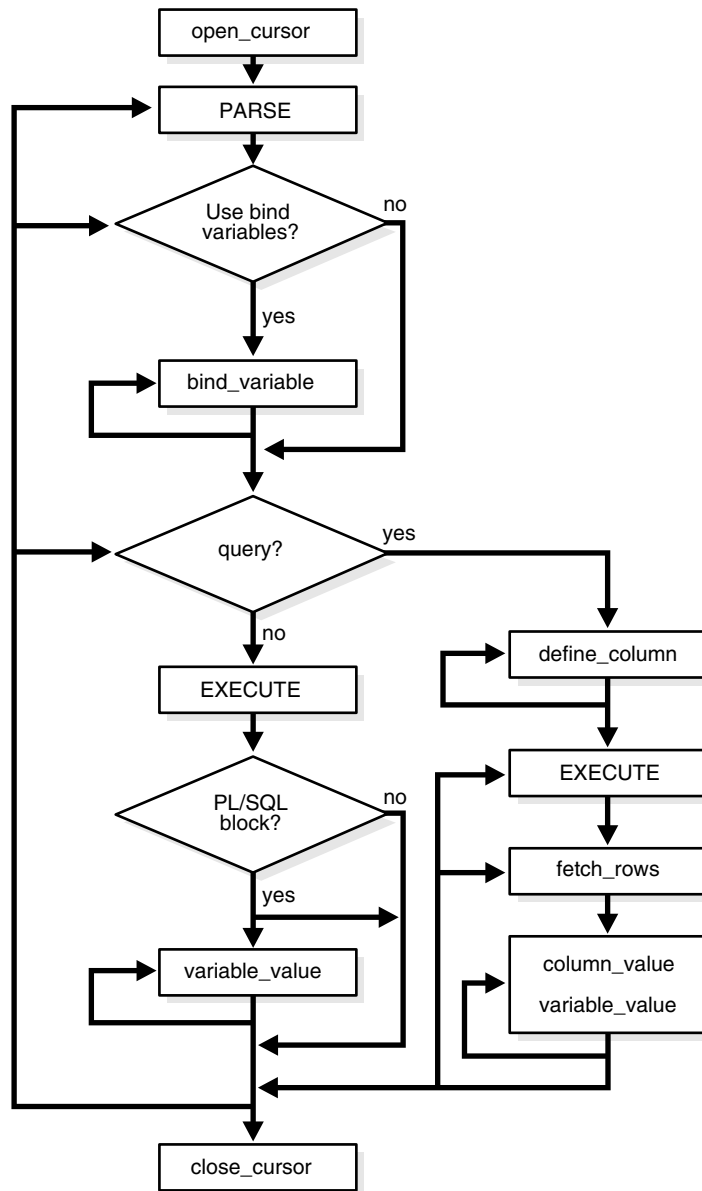
#### **PARSE**

Every SQL statement must be parsed by calling the [PARSE Procedures](#). Parsing the statement checks the statement's syntax and associates it with the cursor in your program.

You can parse any DML or DDL statement. DDL statements are run on the parse, which performs the implied commit.

The execution flow of DBMS\_SQL is shown in [Figure 136-1](#).

Figure 136-1 DBMS\_SQL Execution Flow



This illustration describes execution flow in DBMS\_SQL that begins with "open\_cursor" and ends with "close\_cursor". Key to the symbols used:

- ... -> denotes "continue to next step", for example "open cursor" -> "PARSE"
- ... material enclosed in double quotation marks denotes an action, for example "PARSE"
- ... material enclosed in single quotation marks denotes a choice point, for example 'Use bind variables': Yes/No

"open\_cursor" -> "PARSE" -> Choice point: 'Use bind variables': Yes/No

'Use bind variables': Yes/No ...Yes -> "bind variable" (option to repeat) -> Choice point: 'query' ->

'Use bind variables': Yes/No ...No -> Choice point: 'query' ->

Choice point: 'query': Yes/No

- Choice point: 'query': Yes/No ...Yes -> "DEFINE\_COLUMN" (option to repeat) -> "EXECUTE" -> "FETCH\_ROWS" -> "column\_value, variable\_value" (option to repeat) -> option to loop back to "EXECUTE", option to loop back to "fetch\_rows", option to loop back to "PARSE", option to loop back to 'Use bind variables', option to loop back to 'query' -> "close\_cursor"
- Choice point: query Yes/No ...No -> "EXECUTE" -> Choice point: pl/sql block(s): Yes/No
  - Choice point: pl/sql block(s): Yes/No ...Yes -> Choice point: 'Use variable values': Yes/No
    - \* Choice point: 'Use variable values': Yes/No ... Yes -> "variable\_value" (option to repeat) -> option to loop back to "PARSE", option to loop back to 'Use bind variables', option to loop back to 'query' -> close\_cursor
    - \* Choice point: 'Use variable values': Yes/No ... No -> option to loop back to "PARSE", option to loop back to 'Use bind variables', option to loop back to 'query' -> close\_cursor
  - Choice point: pl/sql block(s): Yes/No ...No -> option to loop back to "PARSE", option to loop back to 'Use bind variables', option to loop back to 'query' -> close\_cursor

\*\*\*\*\*

#### **BIND\_VARIABLE or BIND\_ARRAY**

Many DML statements require that data in your program be input to Oracle. When you define a SQL statement that contains input data to be supplied at runtime, you must use placeholders in the SQL statement to mark where data must be supplied.

For each placeholder in the SQL statement, you must call one of the bind procedures, the [BIND\\_ARRAY Procedures](#) on page 136-52 or the [BIND\\_VARIABLE Procedures](#) on page 136-55, to supply the value of a variable in your program (or the values of an array) to the placeholder. When the SQL statement is subsequently run, Oracle uses the data that your program has placed in the output and input, or bind, variables.

DBMS\_SQL can run a DML statement multiple times — each time with a different bind variable. The [BIND\\_ARRAY](#) procedure lets you bind a collection of scalars, each value of which is used as an input variable once for each EXECUTE. This is similar to the array interface supported by the OCI.

#### **DEFINE\_COLUMN, DEFINE\_COLUMN\_LONG, or DEFINE\_ARRAY**

The columns of the row being selected in a SELECT statement are identified by their relative positions as they appear in the select list, from left to right. For a query, you must call one of the define procedures ([DEFINE\\_COLUMN](#), [DEFINE\\_COLUMN\\_LONG](#), or [DEFINE\\_ARRAY](#)) to specify the variables that are to receive the SELECT values, much the way an INTO clause does for a static query.

Use the [DEFINE\\_COLUMN\\_LONG](#) procedure to define LONG columns, in the same way that [DEFINE\\_COLUMN](#) is used to define non-LONG columns. You must call [DEFINE\\_COLUMN\\_LONG](#) before using the [COLUMN\\_VALUE\\_LONG](#) procedure to fetch from the LONG column.

Use the [DEFINE\\_ARRAY](#) procedure to define a PL/SQL collection into which you want to fetch rows in a single SELECT statement. [DEFINE\\_ARRAY](#) provides an interface to fetch multiple rows at one fetch. You must call [DEFINE\\_ARRAY](#) before using the [COLUMN\\_VALUE](#) procedure to fetch the rows.

**EXECUTE**

Call the `EXECUTE` function to run your SQL statement.

**FETCH\_ROWS or EXECUTE\_AND\_FETCH**

The `FETCH_ROWS` function retrieves the rows that satisfy the query. Each successive fetch retrieves another set of rows, until the fetch is unable to retrieve anymore rows. Instead of calling `EXECUTE` and then `FETCH_ROWS`, you may find it more efficient to call `EXECUTE_AND_FETCH` if you are calling `EXECUTE` for a single execution.

**VARIABLE\_VALUE, COLUMN\_VALUE, or COLUMN\_VALUE\_LONG**

For queries, call `COLUMN_VALUE` to determine the value of a column retrieved by the `FETCH_ROWS` call. For anonymous blocks containing calls to PL/SQL procedures or DML statements with `returning` clause, call `VARIABLE_VALUE` to retrieve the values assigned to the output variables when statements were run.

To fetch just part of a `LONG` database column (which can be up to two gigabytes in size), use the `COLUMN_VALUE_LONG` procedure. You can specify the offset (in bytes) into the column value, and the number of bytes to fetch.

**CLOSE\_CURSOR**

When you no longer need a cursor for a session, close the cursor by calling `CLOSE_CURSOR`. If you are using an Oracle Open Gateway, then you may need to close cursors at other times as well. Consult your *Oracle Open Gateway* documentation for additional information.

If you neglect to close a cursor, then the memory used by that cursor remains allocated even though it is no longer needed.

**Processing Queries**

If you are using dynamic SQL to process a query, then you must perform the following steps:

1. Specify the variables that are to receive the values returned by the `SELECT` statement by calling the [DEFINE\\_COLUMN Procedures](#), the [DEFINE\\_COLUMN\\_LONG Procedure](#), or the [DEFINE\\_ARRAY Procedure](#).
2. Run your `SELECT` statement by calling the [EXECUTE Function](#).
3. Call the [FETCH\\_ROWS Function](#) (or `EXECUTE_AND_FETCH`) to retrieve the rows that satisfied your query.
4. Call [COLUMN\\_VALUE Procedure](#) or [COLUMN\\_VALUE\\_LONG Procedure](#) to determine the value of a column retrieved by the [FETCH\\_ROWS Function](#) for your query. If you used anonymous blocks containing calls to PL/SQL procedures, then you must call the [VARIABLE\\_VALUE Procedures](#) to retrieve the values assigned to the output variables of these procedures.

**Processing Updates, Inserts, and Deletes**

If you are using dynamic SQL to process an `INSERT`, `UPDATE`, or `DELETE`, then you must perform the following steps:

1. You must first run your `INSERT`, `UPDATE`, or `DELETE` statement by calling the [EXECUTE Function](#).
2. If statements have the `returning` clause, then you must call the [VARIABLE\\_VALUE Procedures](#) to retrieve the values assigned to the output variables.

## Locating Errors

There are additional functions in the DBMS\_SQL package for obtaining information about the last referenced cursor in the session. The values returned by these functions are only meaningful immediately after a SQL statement is run. In addition, some error-locating functions are only meaningful after certain DBMS\_SQL calls. For example, you call the [LAST\\_ERROR\\_POSITION Function](#) immediately after a PARSE.

## Examples

This section provides example procedures that make use of the DBMS\_SQL package.

### Example 1

This example does not require the use of dynamic SQL because the text of the statement is known at compile time, but it illustrates the basic concept underlying the package.

The DEMO procedure deletes all of the employees from the EMP table whose salaries are greater than the salary that you specify when you run DEMO.

```
CREATE OR REPLACE PROCEDURE demo(salary IN NUMBER) AS
    cursor_name INTEGER;
    rows_processed INTEGER;
BEGIN
    cursor_name := dbms_sql.open_cursor;
    DBMS_SQL.PARSE(cursor_name, 'DELETE FROM emp WHERE sal > :x',
        DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(cursor_name, ':x', salary);
    rows_processed := DBMS_SQL.EXECUTE(cursor_name);
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
EXCEPTION
WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

### Example 2

The following sample procedure is passed a SQL statement, which it then parses and runs:

```
CREATE OR REPLACE PROCEDURE exec(string IN varchar2) AS
    cursor_name INTEGER;
    ret INTEGER;
BEGIN
    cursor_name := DBMS_SQL.OPEN_CURSOR;
```

DDL statements are run by the parse call, which performs the implied commit.

```
    DBMS_SQL.PARSE(cursor_name, string, DBMS_SQL.NATIVE);
    ret := DBMS_SQL.EXECUTE(cursor_name);
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Creating such a procedure enables you to perform the following operations:

- The SQL statement can be dynamically generated at runtime by the calling program.
- The SQL statement can be a DDL statement or a DML without binds.

For example, after creating this procedure, you could make the following call:

```
exec('create table acct(c1 integer)');
```

You could even call this procedure remotely, as shown in the following example. This lets you perform remote DDL.

```
exec@hq.com('CREATE TABLE acct(c1 INTEGER)');
```

**Example 3**

The following sample procedure is passed the names of a source and a destination table, and copies the rows from the source table to the destination table. This sample procedure assumes that both the source and destination tables have the following columns:

```
id          of type NUMBER
name       of type VARCHAR2(30)
birthdate of type DATE
```

This procedure does not specifically require the use of dynamic SQL; however, it illustrates the concepts of this package.

```
CREATE OR REPLACE PROCEDURE copy (
    source      IN VARCHAR2,
    destination IN VARCHAR2) IS
    id_var      NUMBER;
    name_var    VARCHAR2(30);
    birthdate_var DATE;
    source_cursor INTEGER;
    destination_cursor INTEGER;
    ignore      INTEGER;
BEGIN

    -- Prepare a cursor to select from the source table:
    source_cursor := dbms_sql.open_cursor;
    DBMS_SQL.PARSE(source_cursor,
        'SELECT id, name, birthdate FROM ' || source,
        DBMS_SQL.NATIVE);
    DBMS_SQL.DEFINE_COLUMN(source_cursor, 1, id_var);
    DBMS_SQL.DEFINE_COLUMN(source_cursor, 2, name_var, 30);
    DBMS_SQL.DEFINE_COLUMN(source_cursor, 3, birthdate_var);
    ignore := DBMS_SQL.EXECUTE(source_cursor);

    -- Prepare a cursor to insert into the destination table:
    destination_cursor := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(destination_cursor,
        'INSERT INTO ' || destination ||
        ' VALUES (:id_bind, :name_bind, :birthdate_bind)',
        DBMS_SQL.NATIVE);

    -- Fetch a row from the source table and insert it into the destination table:
    LOOP
        IF DBMS_SQL.FETCH_ROWS(source_cursor)>0 THEN
            -- get column values of the row
            DBMS_SQL.COLUMN_VALUE(source_cursor, 1, id_var);
            DBMS_SQL.COLUMN_VALUE(source_cursor, 2, name_var);
            DBMS_SQL.COLUMN_VALUE(source_cursor, 3, birthdate_var);

            -- Bind the row into the cursor that inserts into the destination table. You
            -- could alter this example to require the use of dynamic SQL by inserting an
            -- if condition before the bind.
            DBMS_SQL.BIND_VARIABLE(destination_cursor, ':id_bind', id_var);
            DBMS_SQL.BIND_VARIABLE(destination_cursor, ':name_bind', name_var);
            DBMS_SQL.BIND_VARIABLE(destination_cursor, ':birthdate_bind',
            birthdate_var);
            ignore := DBMS_SQL.EXECUTE(destination_cursor);
        ELSE

            -- No more rows to copy:

```

```

        EXIT;
    END IF;
END LOOP;

-- Commit and close all cursors:
COMMIT;
DBMS_SQL.CLOSE_CURSOR(source_cursor);
DBMS_SQL.CLOSE_CURSOR(destination_cursor);
EXCEPTION
    WHEN OTHERS THEN
        IF DBMS_SQL.IS_OPEN(source_cursor) THEN
            DBMS_SQL.CLOSE_CURSOR(source_cursor);
        END IF;
        IF DBMS_SQL.IS_OPEN(destination_cursor) THEN
            DBMS_SQL.CLOSE_CURSOR(destination_cursor);
        END IF;
        RAISE;
END;
/

```

### Examples 3, 4, and 5: Bulk DML

This series of examples shows how to use bulk array binds (table items) in the SQL DML statements INSERT, UPDATE and DELETE.

Here is an example of a bulk INSERT statement that demonstrates adding seven new employees to the emp table:

```

DECLARE
    stmt VARCHAR2(200);
    empno_array      DBMS_SQL.NUMBER_TABLE;
    empname_array    DBMS_SQL.VARCHAR2_TABLE;
    jobs_array       DBMS_SQL.VARCHAR2_TABLE;
    mgr_array        DBMS_SQL.NUMBER_TABLE;
    hiredate_array   DBMS_SQL.VARCHAR2_TABLE;
    sal_array        DBMS_SQL.NUMBER_TABLE;
    comm_array       DBMS_SQL.NUMBER_TABLE;
    deptno_array     DBMS_SQL.NUMBER_TABLE;
    c                NUMBER;
    dummy           NUMBER;
BEGIN
    empno_array(1) := 9001;
    empno_array(2) := 9002;
    empno_array(3) := 9003;
    empno_array(4) := 9004;
    empno_array(5) := 9005;
    empno_array(6) := 9006;
    empno_array(7) := 9007;

    empname_array(1) := 'Dopey';
    empname_array(2) := 'Grumpy';
    empname_array(3) := 'Doc';
    empname_array(4) := 'Happy';
    empname_array(5) := 'Bashful';
    empname_array(6) := 'Sneezy';
    empname_array(7) := 'Sleepy';

    jobs_array(1) := 'Miner';
    jobs_array(2) := 'Miner';
    jobs_array(3) := 'Miner';
    jobs_array(4) := 'Miner';

```



```
jobs_array(5) := 'Miner';
jobs_array(6) := 'Miner';
jobs_array(7) := 'Miner';

mgr_array(1) := 9003;
mgr_array(2) := 9003;
mgr_array(3) := 9003;
mgr_array(4) := 9003;
mgr_array(5) := 9003;
mgr_array(6) := 9003;
mgr_array(7) := 9003;

hiredate_array(1) := '06-DEC-2006';
hiredate_array(2) := '06-DEC-2006';
hiredate_array(3) := '06-DEC-2006';
hiredate_array(4) := '06-DEC-2006';
hiredate_array(5) := '06-DEC-2006';
hiredate_array(6) := '06-DEC-2006';
hiredate_array(7) := '06-DEC-2006';

sal_array(1) := 1000;
sal_array(2) := 1000;
sal_array(3) := 1000;
sal_array(4) := 1000;
sal_array(5) := 1000;
sal_array(6) := 1000;
sal_array(7) := 1000;

comm_array(1) := 0;
comm_array(2) := 0;
comm_array(3) := 0;
comm_array(4) := 0;
comm_array(5) := 0;
comm_array(6) := 0;
comm_array(7) := 0;

deptno_array(1) := 11;
deptno_array(2) := 11;
deptno_array(3) := 11;
deptno_array(4) := 11;
deptno_array(5) := 11;
deptno_array(6) := 11;
deptno_array(7) := 11;

stmt := 'INSERT INTO emp VALUES(
      :num_array, :name_array, :jobs_array, :mgr_array, :hiredate_array,
      :sal_array, :comm_array, :deptno_array)';
c := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
DBMS_SQL.BIND_ARRAY(c, ':num_array', empno_array);
DBMS_SQL.BIND_ARRAY(c, ':name_array', empname_array);
DBMS_SQL.BIND_ARRAY(c, ':jobs_array', jobs_array);
DBMS_SQL.BIND_ARRAY(c, ':mgr_array', mgr_array);
DBMS_SQL.BIND_ARRAY(c, ':hiredate_array', hiredate_array);
DBMS_SQL.BIND_ARRAY(c, ':sal_array', sal_array);
DBMS_SQL.BIND_ARRAY(c, ':comm_array', comm_array);
DBMS_SQL.BIND_ARRAY(c, ':deptno_array', deptno_array);

dummy := DBMS_SQL.EXECUTE(c);
DBMS_SQL.CLOSE_CURSOR(c);
```

```

EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
        DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/
SHOW ERRORS;

```

Here is an example of a bulk UPDATE statement that demonstrates updating salaries for four existing employees in the emp table:

```

DECLARE
    stmt VARCHAR2(200);
    empno_array    DBMS_SQL.NUMBER_TABLE;
    salary_array   DBMS_SQL.NUMBER_TABLE;
    c              NUMBER;
    dummy         NUMBER;
BEGIN

    empno_array(1) := 7369;
    empno_array(2) := 7876;
    empno_array(3) := 7900;
    empno_array(4) := 7934;

    salary_array(1) := 10000;
    salary_array(2) := 10000;
    salary_array(3) := 10000;
    salary_array(4) := 10000;

    stmt := 'update emp set sal = :salary_array
            WHERE empno = :num_array';
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_ARRAY(c, ':num_array', empno_array);
    DBMS_SQL.BIND_ARRAY(c, ':salary_array', salary_array);
    dummy := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.CLOSE_CURSOR(c);

    EXCEPTION WHEN OTHERS THEN
        IF DBMS_SQL.IS_OPEN(c) THEN
            DBMS_SQL.CLOSE_CURSOR(c);
        END IF;
        RAISE;
END;
/

```

In a DELETE statement, for example, you could bind in an array in the WHERE clause and have the statement be run for each element in the array:

```

DECLARE
    stmt VARCHAR2(200);
    dept_no_array DBMS_SQL.NUMBER_TABLE;
    c NUMBER;
    dummy NUMBER;
begin
    dept_no_array(1) := 10; dept_no_array(2) := 20;
    dept_no_array(3) := 30; dept_no_array(4) := 40;
    dept_no_array(5) := 30; dept_no_array(6) := 40;
    stmt := 'delete from emp where deptno = :dept_array';
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);

```

```

DBMS_SQL.BIND_ARRAY(c, ':dept_array', dept_no_array, 1, 4);
dummy := DBMS_SQL.EXECUTE(c);
DBMS_SQL.CLOSE_CURSOR(c);

EXCEPTION WHEN OTHERS THEN
  IF DBMS_SQL.IS_OPEN(c) THEN
    DBMS_SQL.CLOSE_CURSOR(c);
  END IF;
  RAISE;
END;
/

```

In the preceding example, only elements 1 through 4 are used as specified by the `BIND_ARRAY` call. Each element of the array potentially deletes a large number of employees from the database.

### Examples 6 and 7: Defining an Array

The following examples show how to use the `DEFINE_ARRAY` procedure:

```

declare
  c      NUMBER;
  d      NUMBER;
  n_tab  DBMS_SQL.NUMBER_TABLE;
  indx   NUMBER := -10;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  dbms_sql.parse(c, 'select n from t order by 1', DBMS_SQL.NATIVE);

  DBMS_SQL.DEFINE_ARRAY(c, 1, n_tab, 10, indx);

  d := DBMS_SQL.EXECUTE(c);
  loop
    d := DBMS_SQL.FETCH_ROWS(c);

    DBMS_SQL.COLUMN_VALUE(c, 1, n_tab);

    EXIT WHEN d != 10;
  END LOOP;

  DBMS_SQL.CLOSE_CURSOR(c);

  EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
END;
/

```

Each time the preceding example does a [FETCH\\_ROWS Function](#) call, it fetches 10 rows that are kept in `DBMS_SQL` buffers. When the [COLUMN\\_VALUE Procedure](#) call is run, those rows move into the PL/SQL table specified (in this case `n_tab`), at positions -10 to -1, as specified in the `DEFINE` statements. When the second batch is fetched in the loop, the rows go to positions 0 to 9; and so on.

A current index into each array is maintained automatically. This index is initialized to "indx" at `EXECUTE` and keeps getting updated every time a `COLUMN_VALUE` call is made. If you re-execute at any point, then the current index for each `DEFINE` is re-initialized to "indx".

In this way the entire result of the query is fetched into the table. When `FETCH_ROWS` cannot fetch 10 rows, it returns the number of rows actually fetched (if no rows could be fetched, then it returns zero) and exits the loop.

Here is another example of using the `DEFINE_ARRAY` procedure:

Consider a table `MULTI_TAB` defined as:

```
CREATE TABLE multi_tab (num NUMBER,
                        dat1 DATE,
                        var VARCHAR2(24),
                        dat2 DATE)
```

To select everything from this table and move it into four PL/SQL tables, you could use the following simple program:

```
declare
  c          NUMBER;
  d          NUMBER;
  n_tab     DBMS_SQL.NUMBER_TABLE;
  d_tab1    DBMS_SQL.DATE_TABLE;
  v_tab     DBMS_SQL.VARCHAR2_TABLE;
  d_tab2    DBMS_SQL.DATE_TABLE;
  indx     NUMBER := 10;
BEGIN

  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'select * from multi_tab order by 1', DBMS_SQL.NATIVE);

  DBMS_SQL.DEFINE_ARRAY(c, 1, n_tab, 5, indx);
  DBMS_SQL.DEFINE_ARRAY(c, 2, d_tab1, 5, indx);
  DBMS_SQL.DEFINE_ARRAY(c, 3, v_tab, 5, indx);
  DBMS_SQL.DEFINE_ARRAY(c, 4, d_tab2, 5, indx);

  d := DBMS_SQL.EXECUTE(c);

  loop
    d := DBMS_SQL.FETCH_ROWS(c);

    DBMS_SQL.COLUMN_VALUE(c, 1, n_tab);
    DBMS_SQL.COLUMN_VALUE(c, 2, d_tab1);
    DBMS_SQL.COLUMN_VALUE(c, 3, v_tab);
    DBMS_SQL.COLUMN_VALUE(c, 4, d_tab2);

    EXIT WHEN d != 5;
  END LOOP;

  DBMS_SQL.CLOSE_CURSOR(c);

  /*

  The four tables can be used for anything. One usage might be to use BIND_ARRAY to
  move the rows to another table by using a statement such as 'INSERT into SOME_T
  values (:a, :b, :c, :d);

  */

  EXCEPTION WHEN OTHERS THEN
    IF DBMS_SQL.IS_OPEN(c) THEN
      DBMS_SQL.CLOSE_CURSOR(c);
    END IF;
    RAISE;
```

```
END;
/
```

### Example 8: Describe Columns

This can be used as a substitute to the SQL\*Plus DESCRIBE call by using a SELECT \* query on the table that you want to describe.

```
DECLARE
  c          NUMBER;
  d          NUMBER;
  col_cnt    INTEGER;
  f          BOOLEAN;
  rec_tab    DBMS_SQL.DESC_TAB;
  col_num    NUMBER;
PROCEDURE print_rec(rec in DBMS_SQL.DESC_REC) IS
BEGIN
  DBMS_OUTPUT.NEW_LINE;
  DBMS_OUTPUT.PUT_LINE('col_type          = '
    || rec.col_type);
  DBMS_OUTPUT.PUT_LINE('col_maxlen       = '
    || rec.col_max_len);
  DBMS_OUTPUT.PUT_LINE('col_name        = '
    || rec.col_name);
  DBMS_OUTPUT.PUT_LINE('col_name_len    = '
    || rec.col_name_len);
  DBMS_OUTPUT.PUT_LINE('col_schema_name = '
    || rec.col_schema_name);
  DBMS_OUTPUT.PUT_LINE('col_schema_name_len = '
    || rec.col_schema_name_len);
  DBMS_OUTPUT.PUT_LINE('col_precision   = '
    || rec.col_precision);
  DBMS_OUTPUT.PUT_LINE('col_scale       = '
    || rec.col_scale);
  DBMS_OUTPUT.PUT('col_null_ok          = ');
  IF (rec.col_null_ok) THEN
    DBMS_OUTPUT.PUT_LINE('true');
  ELSE
    DBMS_OUTPUT.PUT_LINE('false');
  END IF;
END;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;

  DBMS_SQL.PARSE(c, 'SELECT * FROM scott.bonus', DBMS_SQL.NATIVE);

  d := DBMS_SQL.EXECUTE(c);

  DBMS_SQL.DESCRIBE_COLUMNS(c, col_cnt, rec_tab);

/*
 * Following loop could simply be for j in 1..col_cnt loop.
 * Here we are simply illustrating some of the PL/SQL table
 * features.
 */
  col_num := rec_tab.first;
  IF (col_num IS NOT NULL) THEN
    LOOP
      print_rec(rec_tab(col_num));
      col_num := rec_tab.next(col_num);
      EXIT WHEN (col_num IS NULL);
    END LOOP;
  END IF;
END;
```

```

        END LOOP;
    END IF;

    DBMS_SQL.CLOSE_CURSOR(c);
END;
/

```

### Example 9: RETURNING clause

The RETURNING clause was added to DML statements in an earlier Oracle database release. With this clause, INSERT, UPDATE, and DELETE statements can return values of expressions. These values are returned in bind variables.

DBMS\_SQL.BIND\_VARIABLE is used to bind these outbinds if a single row is inserted, updated, or deleted. If multiple rows are inserted, updated, or deleted, then DBMS\_SQL.BIND\_ARRAY is used. DBMS\_SQL.VARIABLE\_VALUE must be called to get the values in these bind variables.

---



---

**Note:** This is similar to DBMS\_SQL.VARIABLE\_VALUE, which must be called after running a PL/SQL block with an out-bind inside DBMS\_SQL.

---



---

#### i) Single row insert

```

CREATE OR REPLACE PROCEDURE single_Row_insert
    (c1 NUMBER, c2 NUMBER, r OUT NUMBER) IS
    c NUMBER;
    n NUMBER;
begin
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'INSERT INTO tab VALUES (:bnd1, :bnd2) ' ||
        'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
    n := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r); -- get value of outbind variable
    DBMS_SQL.CLOSE_CURSOR(c);
END;
/

```

#### ii) Single row update

```

CREATE OR REPLACE PROCEDURE single_Row_update
    (c1 NUMBER, c2 NUMBER, r out NUMBER) IS
    c NUMBER;
    n NUMBER;
BEGIN
    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, 'UPDATE tab SET c1 = :bnd1, c2 = :bnd2 ' ||
        'WHERE rownum < 2 ' ||
        'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
    DBMS_SQL.BIND_VARIABLE(c, 'bnd3', r);
    n := DBMS_SQL.EXECUTE(c);
    DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r); -- get value of outbind variable
    DBMS_SQL.CLOSE_CURSOR(c);
END;

```

/

## iii) Single row delete

```

CREATE OR REPLACE PROCEDURE single_Row_Delete
  (c1 NUMBER, r OUT NUMBER) IS
c NUMBER;
n number;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'DELETE FROM tab WHERE ROWNUM = :bnd1 ' ||
    'RETURNING c1*c2 INTO :bnd2', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd2', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/

```

## iv) Multiple row insert

```

CREATE OR REPLACE PROCEDURE multi_Row_insert
  (c1 DBMS_SQL.NUMBER_TABLE, c2 DBMS_SQL.NUMBER_TABLE,
  r OUT DBMS_SQL.NUMBER_TABLE) IS
c NUMBER;
n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'insert into tab VALUES (:bnd1, :bnd2) ' ||
    'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, 'bnd1', c1);
  DBMS_SQL.BIND_ARRAY(c, 'bnd2', c2);
  DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/

```

## v) Multiple row Update.

```

CREATE OR REPLACE PROCEDURE multi_Row_update
  (c1 NUMBER, c2 NUMBER, r OUT DBMS_SQL.NUMBER_TABLE) IS
c NUMBER;
n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'UPDATE tab SET c1 = :bnd1 WHERE c2 = :bnd2 ' ||
    'RETURNING c1*c2 INTO :bnd3', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd1', c1);
  DBMS_SQL.BIND_VARIABLE(c, 'bnd2', c2);
  DBMS_SQL.BIND_ARRAY(c, 'bnd3', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd3', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/

```

---



---

**Note:** bnd1 and bnd2 can be arrays as well. The value of the expression for all the rows updated will be in bnd3. There is no way of differentiating which rows were updated of each value of bnd1 and bnd2.

---



---

#### vi) Multiple row delete

```
CREATE OR REPLACE PROCEDURE multi_row_delete
  (c1 DBMS_SQL.NUMBER_TABLE,
   r OUT DBMS_SQL.NUMBER_TABLE) IS
  c NUMBER;
  n NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'DELETE FROM tab WHERE c1 = :bnd1' ||
    'RETURNING c1*c2 INTO :bnd2', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, 'bnd1', c1);
  DBMS_SQL.BIND_ARRAY(c, 'bnd2', r);
  n := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', r);-- get value of outbind variable
  DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

#### vii) Out-bind in bulk PL/SQL

```
CREATE OR REPLACE PROCEDURE foo (n NUMBER, square OUT NUMBER) IS
BEGIN square := n * n; END;/

CREATE OR REPLACE PROCEDURE bulk_plsql
  (n DBMS_SQL.NUMBER_TABLE, square OUT DBMS_SQL.NUMBER_TABLE) IS
  c NUMBER;
  r NUMBER;
BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, 'BEGIN foo(:bnd1, :bnd2); END;', DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_ARRAY(c, 'bnd1', n);
  DBMS_SQL.BIND_ARRAY(c, 'bnd2', square);
  r := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.VARIABLE_VALUE(c, 'bnd2', square);
END;
/
```

---



---

**Note:** DBMS\_SQL.BIND\_ARRAY of number\_Table internally binds a number. The number of times statement is run depends on the number of elements in an inbind array.

---



---

#### Example 10: Binds and Defines of User-defined Types in DBMS\_SQL

```
CREATE TYPE dnames_var IS VARRAY(7) OF VARCHAR2(30)
/

CREATE TABLE depts (region VARCHAR2(25), dept_names dnames_var)
/

INSERT INTO depts VALUES('Europe', dnames_var('Shipping','Sales','Finance'))
/
INSERT INTO depts VALUES('Americas', dnames_var('Sales','Finance','Shipping'))
```



```

/
INSERT INTO depts
    VALUES ('Asia', dnames_var('Finance', 'Payroll', 'Shipping', 'Sales'))
/

CREATE OR REPLACE PROCEDURE update_depts(new_dnames dnames_var, region VARCHAR2)
IS
    some_dnames dnames_var;
    c          NUMBER;
    r          NUMBER;
    sql_stmt   VARCHAR2(32767) :=
        'UPDATE depts SET dept_names = :b1 WHERE region = :b2 RETURNING dept_names
        INTO :b3';

BEGIN

    c := DBMS_SQL.OPEN_CURSOR;

    DBMS_SQL.PARSE(c, sql_stmt, dbms_sql.native);

    DBMS_SQL.BIND_VARIABLE(c, 'b1', new_dnames);
    DBMS_SQL.BIND_VARIABLE(c, 'b2', region);
    DBMS_SQL.BIND_VARIABLE(c, 'b3', some_dnames);

    r := DBMS_SQL.EXECUTE(c);

    -- Get value of outbind variable
    DBMS_SQL.VARIABLE_VALUE(c, 'b3', some_dnames);

    DBMS_SQL.CLOSE_CURSOR(c);

    -- select dept_names
    sql_stmt := 'SELECT dept_names FROM depts WHERE region = :b1';

    c := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(c, sql_stmt, dbms_sql.native);

    DBMS_SQL.DEFINE_COLUMN(c, 1, some_dnames);
    DBMS_SQL.BIND_VARIABLE(c, 'b1', region);

    r := DBMS_SQL.EXECUTE_AND_FETCH(c);

    DBMS_SQL.COLUMN_VALUE(c, 1, some_dnames);

    DBMS_SQL.CLOSE_CURSOR(c);

    -- loop through some_dnames collections
    FOR i IN some_dnames.FIRST .. some_dnames.LAST LOOP
        DBMS_OUTPUT.PUT_LINE('Dept. Name = ' || some_dnames(i) || ' Updated!');
    END LOOP;

END;
/

DECLARE
    new_dnames dnames_var;
BEGIN
    new_dnames := dnames_var('Benefits', 'Advertising', 'Contracting',
                             'Executive', 'Marketing');
    update_depts(new_dnames, 'Asia');
END;

```

/

---

## Data Structures

The DBMS\_SQL package defines the following RECORD types and TABLE types.

### RECORD Types

- DESC\_REC Record Type
- DESC\_REC2 Record Type
- DESC\_REC3 Record Type

### TABLE Types

- BFILE\_TABLE Table Type
- BINARY\_DOUBLE\_TABLE Table Type
- BINARY\_FLOAT\_TABLE Table Type
- BLOB\_TABLE Table Type
- CLOB\_TABLE Table Type
- DATE\_TABLE Table Type
- DESC\_TAB Table Type
- DESC\_TAB2 Table Type
- DESC\_TAB3 Table Type
- INTERVAL\_DAY\_TO\_SECOND\_TABLE Table Type
- INTERVAL\_YEAR\_TO\_MONTH\_TABLE Table Type
- NUMBER\_TABLE Table Type
- TIME\_TABLE Table Type
- TIME\_WITH\_TIME\_ZONE\_TABLE Table Type
- TIMESTAMP\_TABLE Table Type
- TIMESTAMP\_WITH\_LTZ\_TABLE Table Type
- TIMESTAMP\_WITH\_TIME\_ZONE\_TABLE Table Type
- UROWID\_TABLE Table Type
- VARCHAR2\_TABLE Table Type
- VARCHAR2A Table Type
- VARCHAR2S Table Type

## DESC\_REC Record Type

---

**Note:** This type has been deprecated in favor of the [DESC\\_REC2 Record Type](#).

---

This record type holds the describe information for a single column in a dynamic query. It is the element type of the DESC\_TAB table type and the [DESCRIBE\\_COLUMNS Procedure](#).

### Syntax

```
TYPE desc_rec IS RECORD (
    col_type          BINARY_INTEGER := 0,
    col_max_len       BINARY_INTEGER := 0,
    col_name          VARCHAR2(32)   := '',
    col_name_len      BINARY_INTEGER := 0,
    col_schema_name   VARCHAR2(32)   := '',
    col_schema_name_len BINARY_INTEGER := 0,
    col_precision     BINARY_INTEGER := 0,
    col_scale         BINARY_INTEGER := 0,
    col_charsetid     BINARY_INTEGER := 0,
    col_charsetform   BINARY_INTEGER := 0,
    col_null_ok       BOOLEAN        := TRUE);
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

### Fields

**Table 136–2** DESC\_REC Fields

Field	Description
col_type	Type of column
col_max_len	Maximum column length
col_name	Name of column
col_name_len	Length of column name
col_schema_name	Column schema name
col_schema_name_len	Length of column schema name
col_precision	Precision of column
col_scale	Scale of column
col_charsetid	Column character set id
col_charsetform	Column character set form
col_null_ok	NULL column flag; TRUE, if NULL possible

## DESC\_REC2 Record Type

DESC\_REC2 is the element type of the DESC\_TAB2 table type and the [DESCRIBE\\_COLUMNS2 Procedure](#).

This record type is identical to DESC\_REC except for the col\_name field, which has been expanded to the maximum possible size for VARCHAR2. It is therefore preferred to DESC\_REC because column name values can be greater than 32 characters. DESC\_REC is deprecated as a result.

### Syntax

```
TYPE desc_rec2 IS RECORD (
  col_type          binary_integer := 0,
  col_max_len       binary_integer := 0,
  col_name          varchar2(32767) := '',
  col_name_len      binary_integer := 0,
  col_schema_name   varchar2(32)   := '',
  col_schema_name_len binary_integer := 0,
  col_precision     binary_integer := 0,
  col_scale         binary_integer := 0,
  col_charsetid     binary_integer := 0,
  col_charsetform   binary_integer := 0,
  col_null_ok       boolean        := TRUE);
```

### Fields

**Table 136–3** DESC\_REC2 Fields

Field	Description
col_type	Type of column
col_max_len	Maximum column length
col_name	Name of column
col_name_len	Length of column name
col_schema_name	Column schema name
col_schema_name_len	Length of column schema name
col_precision	Precision of column
col_scale	Scale of column
col_charsetid	Column character set id
col_charsetform	Column character set form
col_null_ok	NULL column flag; TRUE, if NULL possible

## DESC\_REC3 Record Type

DESC\_REC3 is the element type of the DESC\_TAB3 table type and the [DESCRIBE\\_COLUMNS3 Procedure](#).

DESC\_REC3 is identical to DESC\_REC2 except for two additional fields to hold the type name (`type_name`) and type name len (`type_name_len`) of a column in a dynamic query. These two fields hold the type name and type name length when the column is a user-defined type (a collection or object type). The `col_type_name` and `col_type_name_len` fields are only populated when the `col_type` field's value is 109, the Oracle type number for user-defined types.

### Syntax

```
TYPE desc_rec3 IS RECORD (
  col_type          binary_integer := 0,
  col_max_len       binary_integer := 0,
  col_name          varchar2(32767) := '',
  col_name_len      binary_integer := 0,
  col_schema_name   varchar2(32) := '',
  col_schema_name_len binary_integer := 0,
  col_precision     binary_integer := 0,
  col_scale         binary_integer := 0,
  col_charsetid     binary_integer := 0,
  col_charsetform   binary_integer := 0,
  col_null_ok       boolean := TRUE,
  col_type_name     varchar2(32767) := '',
  col_type_name_len binary_integer := 0);
```

### Fields

**Table 136–4** DESC\_REC3 Fields

Field	Description
<code>col_type</code>	Type of column
<code>col_max_len</code>	Maximum column length
<code>col_name</code>	Name of column
<code>col_name_len</code>	Length of column name
<code>col_schema_name</code>	Column schema name
<code>col_schema_name_len</code>	Length of column schema name
<code>col_precision</code>	Precision of column
<code>col_scale</code>	Scale of column
<code>col_charsetid</code>	Column character set ID
<code>col_charsetform</code>	Column character set form
<code>col_null_ok</code>	NULL column flag; TRUE, if NULL possible
<code>col_type_name</code>	User-define type column type name, this field is valid when <code>col_type</code> is 109
<code>col_type_name_len</code>	Length of user-define type column type name, this field is valid when <code>col_type</code> is 109

## **BFILE\_TABLE Table Type**

This is a table of BFILE.

### **Syntax**

```
TYPE bfile_table IS TABLE OF BFILE INDEX BY BINARY_INTEGER;
```

## **BINARY\_DOUBLE\_TABLE Table Type**

This is a table of BINARY\_DOUBLE.

### **Syntax**

```
TYPE binary_double_table IS TABLE OF BINARY_DOUBLE INDEX BY BINARY_INTEGER;
```



## **BINARY\_FLOAT\_TABLE Table Type**

This is a table of `BINARY_FLOAT`.

### **Syntax**

```
TYPE binary_float_table IS TABLE OF BINARY_FLOAT INDEX BY BINARY_INTEGER;
```

## **BLOB\_TABLE Table Type**

This is a table of BLOB.

### **Syntax**

```
TYPE blob_table IS TABLE OF BLOB INDEX BY BINARY_INTEGER;
```

## **CLOB\_TABLE Table Type**

This is a table of CLOB.

### **Syntax**

```
TYPE clob_table IS TABLE OF CLOB INDEX BY BINARY_INTEGER;
```

## DATE\_TABLE Table Type

This is a table of DATE.

### Syntax

```
type date_table IS TABLE OF DATE INDEX BY BINARY_INTEGER;
```

## DESC\_TAB Table Type

This is a table of [DESC\\_REC Record Type](#).

### Syntax

```
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

## DESC\_TAB2 Table Type

This is a table of [DESC\\_REC2 Record Type](#).

### Syntax

```
TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER;
```

## DESC\_TAB3 Table Type

This is a table of [DESC\\_REC3 Record Type](#).

### Syntax

```
TYPE desc_tab3 IS TABLE OF desc_rec3 INDEX BY BINARY_INTEGER;
```

## INTERVAL\_DAY\_TO\_SECOND\_TABLE Table Type

This is a table of DSINTERVAL\_UNCONSTRAINED.

### Syntax

```
TYPE interval_day_to_second_Table IS TABLE OF  
    DSINTERVAL_UNCONSTRAINED INDEX BY binary_integer;
```



## INTERVAL\_YEAR\_TO\_MONTH\_TABLE Table Type

This is a table of YMINTERVAL\_UNCONSTRAINED.

### Syntax

```
TYPE interval_year_to_month_table IS TABLE OF YMINTERVAL_UNCONSTRAINED  
INDEX BY BINARY_INTEGER;
```

## NUMBER\_TABLE Table Type

This is a table of NUMBER.

### Syntax

```
TYPE number_table IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

## TIME\_TABLE Table Type

This is a table of `TIME_UNCONSTRAINED`.

### Syntax

```
TYPE time_table IS TABLE OF TIME_UNCONSTRAINED INDEX BY BINARY_INTEGER;
```

## TIME\_WITH\_TIME\_ZONE\_TABLE Table Type

This is a table of `TIME_TZ_UNCONSTRAINED`.

### Syntax

```
TYPE time_with_time_zone_table IS TABLE OF TIME_TZ_UNCONSTRAINED  
INDEX BY BINARY_INTEGER;;
```

## **TIMESTAMP\_TABLE Table Type**

This is a table of `TIMESTAMP_UNCONSTRAINED`.

### **Syntax**

```
TYPE timestamp_table IS TABLE OF TIMESTAMP_UNCONSTRAINED INDEX BY BINARY_INTEGER;
```

## TIMESTAMP\_WITH\_LTZ\_TABLE Table Type

This is a table of `TIMESTAMP_LTZ_UNCONSTRAINED`

### Syntax

```
TYPE timestamp_with_ltz_table IS TABLE OF  
    TIMESTAMP_LTZ_UNCONSTRAINED INDEX BY binary_integer;
```

## TIMESTAMP\_WITH\_TIME\_ZONE\_TABLE Table Type

This is a table of `TIMESTAMP_TZ_UNCONSTRAINED`.

### Syntax

```
TYPE timestamp_with_time_zone_Table IS TABLE OF  
    TIMESTAMP_TZ_UNCONSTRAINED INDEX BY binary_integer;
```

## UROWID\_TABLE Table Type

This is a table of UROWID.

### Syntax

```
TYPE urowid_table IS TABLE OF UROWID INDEX BY BINARY_INTEGER;
```



## **VARCHAR2\_TABLE Table Type**

This is table of VARCHAR2 (2000).

### **Syntax**

```
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
```

## VARCHAR2A Table Type

This is table of VARCHAR2 (32767).

### Syntax

```
TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

## VARCHAR2S Table Type

This is table of VARCHAR2 (256) .

---

---

**Note:** This type has been superseded by the [VARCHAR2A Table Type](#). Although it is currently retained for backward compatibility of legacy code, it is in the process of deprecation and will be de-supported in a future release.

---

---

### Syntax

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
```

---

## Summary of DBMS\_SQL Subprograms

**Table 136–5 DBMS\_SQL Package Subprograms**

Subprogram	Description
<a href="#">BIND_ARRAY Procedures</a> on page 136-52	Binds a given value to a given collection
<a href="#">BIND_VARIABLE Procedures</a> on page 136-55	Binds a given value to a given variable
<a href="#">CLOSE_CURSOR Procedure</a> on page 136-57	Closes given cursor and frees memory
<a href="#">COLUMN_VALUE Procedure</a> on page 136-58	Returns value of the cursor element for a given position in a cursor
<a href="#">COLUMN_VALUE_LONG Procedure</a> on page 136-61	Returns a selected part of a LONG column, that has been defined using <code>DEFINE_COLUMN_LONG</code>
<a href="#">DEFINE_ARRAY Procedure</a> on page 136-62	Defines a collection to be selected from the given cursor, used only with <code>SELECT</code> statements
<a href="#">DEFINE_COLUMN Procedures</a> on page 136-65	Defines a column to be selected from the given cursor, used only with <code>SELECT</code> statements
<a href="#">DEFINE_COLUMN_CHAR Procedure</a> on page 136-67	Defines a column of type <code>CHAR</code> to be selected from the given cursor, used only with <code>SELECT</code> statements
<a href="#">DEFINE_COLUMN_LONG Procedure</a> on page 136-68	Defines a LONG column to be selected from the given cursor, used only with <code>SELECT</code> statements
<a href="#">DEFINE_COLUMN_RAW Procedure</a> on page 136-69	Defines a column of type <code>RAW</code> to be selected from the given cursor, used only with <code>SELECT</code> statements
<a href="#">DEFINE_COLUMN_ROWID Procedure</a> on page 136-70	Defines a column of type <code>ROWID</code> to be selected from the given cursor, used only with <code>SELECT</code> statements
<a href="#">DESCRIBE_COLUMNS Procedure</a> on page 136-71	Describes the columns for a cursor opened and parsed through <code>DBMS_SQL</code>
<a href="#">DESCRIBE_COLUMNS2 Procedure</a> on page 136-72	Describes describes the specified column, an alternative to <a href="#">DESCRIBE_COLUMNS Procedure</a>
<a href="#">DESCRIBE_COLUMNS3 Procedure</a> on page 136-73	Describes describes the specified column, an alternative to <a href="#">DESCRIBE_COLUMNS Procedure</a>
<a href="#">EXECUTE Function</a> on page 136-75	Executes a given cursor
<a href="#">EXECUTE_AND_FETCH Function</a> on page 136-76	Executes a given cursor and fetch rows
<a href="#">FETCH_ROWS Function</a> on page 136-77	Fetches a row from a given cursor
<a href="#">IS_OPEN Function</a> on page 136-78	Returns <code>TRUE</code> if given cursor is open
<a href="#">LAST_ERROR_POSITION Function</a> on page 136-79	Returns byte offset in the SQL statement text where the error occurred
<a href="#">LAST_ROW_COUNT Function</a> on page 136-80	Returns cumulative count of the number of rows fetched
<a href="#">LAST_ROW_ID Function</a> on page 136-81	Returns <code>ROWID</code> of last row processed
<a href="#">LAST_SQL_FUNCTION_CODE Function</a> on page 136-82	Returns SQL function code for statement

**Table 136-5 (Cont.) DBMS\_SQL Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">OPEN_CURSOR Function</a> on page 136-83	Returns cursor ID number of new cursor
<a href="#">PARSE Procedures</a> on page 136-84	Parses given statement
<a href="#">TO_CURSOR_NUMBER Function</a> on page 136-84	Takes an OPENED strongly or weakly-typed ref cursor and transforms it into a DBMS_SQL cursor number
<a href="#">TO_REFCURSOR Function</a> on page 136-89	Takes an OPENED, PARSED, and EXECUTED cursor and transforms/migrates it into a PL/SQL manageable REF CURSOR (a weakly-typed cursor) that can be consumed by PL/SQL native dynamic SQL switched to use native dynamic SQL
<a href="#">VARIABLE_VALUE Procedures</a> on page 136-91	Returns value of named variable for given cursor

## BIND\_ARRAY Procedures

This procedure binds a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement.

### Syntax

```
DBMS_SQL.BIND_ARRAY (
  c           IN INTEGER,
  name        IN VARCHAR2,
  <table_variable> IN <datatype>
  [, index1   IN INTEGER,
  index2      IN INTEGER] );
```

Where the <table\_variable> and its corresponding <datatype> can be any one of the following matching pairs:

```
<clob_tab>      Clob_Table
<bflt_tab>      Binary_Float_Table
<bdbl_tab>      Binary_Double_Table
<blob_tab>     Blob_Table
<bfile_tab>     Bfile_Table
<date_tab>     Date_Table
<num_tab>      Number_Table
<urowid_tab>   Urowid_Table
<vchr2_tab>    Varchar2_Table
<tm_tab>       Time_Table
<ttz_tab>      Time_With_Time_Zone_Table
<tms_tab>      Timestamp_Table
<tstz_tab>     Timestamp_With_ltz_Table;
<tstz_tab>     Timestamp_With_Time_Zone_Table
<ids_tab>     Interval_Day_To_Second_Table
<iym_tab>     Interval_Year_To_Month_Table
```

Notice that the BIND\_ARRAY procedure is overloaded to accept different datatypes.

### Parameters

**Table 136–6 BIND\_ARRAY Procedure Parameters**

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the collection in the statement.
table_variable	Local variable that has been declared as <datatype>.
index1	Index for the table element that marks the lower bound of the range.
index2	Index for the table element that marks the upper bound of the range.

### Usage Notes

The length of the bind variable name should be <=30 bytes.

For binding a range, the table must contain the elements that specify the range — tab(index1) and tab(index2) — but the range does not have to be dense. Index1 must be less than or equal to index2. All elements between tab(index1) and tab(index2) are used in the bind.

If you do not specify indexes in the bind call, and two different binds in a statement specify tables that contain a different number of elements, then the number of elements actually used is the minimum number between all tables. This is also the case if you specify indexes — the minimum range is selected between the two indexes for all tables.

Not all bind variables in a query have to be array binds. Some can be regular binds and the same value are used for each element of the collections in expression evaluations (and so forth).

**See Also:** "Examples 3, 4, and 5: Bulk DML" on page 136-14 for examples of how to bind collections.

### Bulk Array Binds

Bulk selects, inserts, updates, and deletes can enhance the performance of applications by bundling many calls into one. The DBMS\_SQL package lets you work on collections of data using the PL/SQL table type.

*Table items* are unbounded homogeneous collections. In persistent storage, they are like other relational tables and have no intrinsic ordering. But when a table item is brought into the workspace (either by querying or by navigational access of persistent data), or when it is created as the value of a PL/SQL variable or parameter, its elements are given subscripts that can be used with array-style syntax to get and set the values of elements.

The subscripts of these elements need not be dense, and can be any number including negative numbers. For example, a table item can contain elements at locations -10, 2, and 7 only.

When a table item is moved from transient workspace to persistent storage, the subscripts are not stored; the table item is unordered in persistent storage.

At bind time the table is copied out from the PL/SQL buffers into local DBMS\_SQL buffers (the same as for all scalar types) and then the table is manipulated from the local DBMS\_SQL buffers. Therefore, if you change the table after the bind call, then that change does not affect the way the execute acts.

### Types for Scalar and LOB Collections

You can declare a local variable as one of the following table-item types, which are defined as public types in DBMS\_SQL.

```

TYPE binary_double_table
           IS TABLE OF BINARY_DOUBLE INDEX BY BINARY_INTEGER;
TYPE binary_float_table
           IS TABLE OF BINARY_FLOAT INDEX BY BINARY_INTEGER;
TYPE bfile_table
           IS TABLE OF BFILE INDEX BY BINARY_INTEGER;
TYPE blob_table
           IS TABLE OF BLOB INDEX BY BINARY_INTEGER;
TYPE clob_table
           IS TABLE OF CLOB INDEX BY BINARY_INTEGER;
TYPE date_table
           IS TABLE OF DATE INDEX BY BINARY_INTEGER;
TYPE interval_day_to_second_Table
           IS TABLE OF dsinterval_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE interval_year_to_MONTH_Table
           IS TABLE OF yminterval_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE number_table
           IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
TYPE time_table
           IS TABLE OF time_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE time_with_time_zone_table

```

```

                IS TABLE OF time_tz_unconstrained
                                INDEX BY BINARY_INTEGER;
TYPE timestamp_table
                IS TABLE OF timestamp_unconstrained
                                INDEX BY BINARY_INTEGER;
TYPE timestamp_with_ltz_Table
                IS TABLE OF timestamp_ltz_unconstrained
                                INDEX BY BINARY_INTEGER;
TYPE timestamp_with_time_zone_Table
                IS TABLE OF timestamp_tz_unconstrained
                                INDEX BY BINARY_INTEGER;
TYPE urowid_table IS TABLE OF UROWID          INDEX BY BINARY_INTEGER;
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

```



## BIND\_VARIABLE Procedures

This procedure binds a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement.

### Syntax

```
DBMS_SQL.BIND_VARIABLE (
    c           IN INTEGER,
    name        IN VARCHAR2,
    value       IN <datatype>);
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_UNCONSTRAINED
TIME_TZ_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
user-defined object types
collections (VARRAYs and nested tables)
REFs
Opaque types
```

Notice that BIND\_VARIABLE is overloaded to accept different datatypes.

The following syntax is also supported for BIND\_VARIABLE. The square brackets [] indicate an optional parameter for the BIND\_VARIABLE function.

```
DBMS_SQL.BIND_VARIABLE (
    c           IN INTEGER,
    name        IN VARCHAR2,
    value       IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN INTEGER]);
```

To bind CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.BIND_VARIABLE_CHAR (
    c           IN INTEGER,
    name        IN VARCHAR2,
    value       IN CHAR CHARACTER SET ANY_CS [,out_value_size IN INTEGER]);
```

```
DBMS_SQL.BIND_VARIABLE_RAW (
    c           IN INTEGER,
    name        IN VARCHAR2,
    value       IN RAW [,out_value_size IN INTEGER]);
```

```
DBMS_SQL.BIND_VARIABLE_ROWID (
    c           IN INTEGER,
```

```
name          IN VARCHAR2,
value         IN ROWID);
```

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

## Pragmas

```
pragma restrict_references(bind_variable,WNDS);
```

## Parameters

**Table 136–7 BIND\_VARIABLE Procedure Parameters**

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the variable in the statement.
value	Value that you want to bind to the variable in the cursor. For IN and IN/OUT variables, the value has the same type as the type of the value being passed in for this parameter.
out_value_size	Maximum expected OUT value size, in bytes, for the VARCHAR2, RAW, CHAR OUT or IN/OUT variable.  If no size is given, then the length of the current value is used. This parameter must be specified if the value parameter is not initialized.

## Usage Notes

If the variable is an IN or IN/OUT variable or an IN collection, then the given bind value must be valid for the variable or array type. Bind values for OUT variables are ignored.

The bind variables or collections of a SQL statement are identified by their names. When binding a value to a bind variable or bind array, the string identifying it in the statement must contain a leading colon, as shown in the following example:

```
SELECT emp_name FROM emp WHERE SAL > :X;
```

For this example, the corresponding bind call would look similar to

```
BIND_VARIABLE(cursor_name, ':X', 3500);
```

or

```
BIND_VARIABLE (cursor_name, 'X', 3500);
```

The length of the bind variable name should be <=30 bytes.

**See Also:** ["Examples 3, 4, and 5: Bulk DML"](#) on page 136-14 for examples of how to bind collections.

## CLOSE\_CURSOR Procedure

This procedure closes a given cursor.

### Syntax

```
DBMS_SQL.CLOSE_CURSOR (  
    c      IN OUT INTEGER);
```

### Pragmas

```
pragma restrict_references(close_cursor,RNDS,WNDS);
```

### Parameters

**Table 136–8** *CLOSE\_CURSOR Procedure Parameters*

Parameter	Mode	Description
c	IN	ID number of the cursor that you want to close.
c	OUT	Cursor is set to null. After you call CLOSE_CURSOR, the memory allocated to the cursor is released and you can no longer fetch from that cursor.

## COLUMN\_VALUE Procedure

This procedure returns the value of the cursor element for a given position in a given cursor. This procedure is used to access the data fetched by calling `FETCH_ROWS`.

### Syntax

```
DBMS_SQL.COLUMN_VALUE (
    c                IN  INTEGER,
    position         IN  INTEGER,
    value            OUT <datatype>
    [,column_error   OUT NUMBER]
    [,actual_length  OUT INTEGER]);
```

Where square brackets [ ] indicate optional parameters and <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_TZ_UNCONSTRAINED
TIME_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
user-defined object types
collections (VARRAYs and nested tables)
REFs
Opaque types
```

For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.COLUMN_VALUE_CHAR (
    c                IN  INTEGER,
    position         IN  INTEGER,
    value            OUT CHAR CHARACTER SET ANY_CS
    [,column_error   OUT NUMBER]
    [,actual_length  OUT INTEGER]);
```

```
DBMS_SQL.COLUMN_VALUE_RAW (
    c                IN  INTEGER,
    position         IN  INTEGER,
    value            OUT RAW
    [,column_error   OUT NUMBER]
    [,actual_length  OUT INTEGER]);
```

```
DBMS_SQL.COLUMN_VALUE_ROWID (
    c                IN  INTEGER,
    position         IN  INTEGER,
    value            OUT ROWID
    [,column_error   OUT NUMBER]
```

```
[,actual_length OUT INTEGER]);
```

The following syntax enables the `COLUMN_VALUE` procedure to accommodate bulk operations:

```
DBMS_SQL.COLUMN_VALUE(
  c                IN                INTEGER,
  position         IN                INTEGER,
  <param_name>    IN OUT NOCOPY    <table_type>);
```

Where the `<param_name>` and its corresponding `<table_type>` can be any one of these matching pairs:

```
bdbl_tab    Binary_Double_Table
bflt_tab    Binary_Float_Table
bf_tab      Bfile_Table
bl_tab      Blob_Table
cl_tab      Clob_Table
d_tab       Date_Table
ids_tab     Interval_Day_To_Second_Table
iym_tab     Interval_Year_To_Month_Table
n_tab       Number_Table
tm_tab      Time_Table
ttz_tab     Time_With_Time_Zone_Table
tms_tab     Timestamp_Table
tstz_tab    Timestamp_With_ltz_Table;
tstz_tab    Timestamp_With_Time_Zone_Table
ur_tab      Urowid_Table
c_tab       Varchar2_Table
```

## Pragmas

```
pragma restrict_references(column_value,RNDS,WNDS);
```

## Parameters

**Table 136–9** *COLUMN\_VALUE Procedure Parameters (Single Row)*

Parameter	Description
<code>c</code>	ID number of the cursor from which you are fetching the values.
<code>position</code>	Relative position of the column in the cursor. The first column in a statement has position 1.
<code>value</code>	Returns the value at the specified column. Oracle raises exception <code>ORA-06562, inconsistent_type</code> , if the type of this output parameter differs from the actual type of the value, as defined by the call to <code>DEFINE_COLUMN</code> .
<code>column_error</code>	Returns any error code for the specified column value.
<code>actual_length</code>	The actual length, before any truncation, of the value in the specified column.

**Table 136–10** *COLUMN\_VALUE Procedure Parameters (Bulk)*

Parameter	Description
<code>c</code>	ID number of the cursor from which you are fetching the values.
<code>position</code>	Relative position of the column in the cursor. The first column in a statement has position 1.
<code>&lt;param_name&gt;</code>	Local variable that has been declared <code>&lt;table_type&gt;</code> . <code>&lt;param_name&gt;</code> is an IN OUT NOCOPY parameter for bulk operations.  For bulk operations, the subprogram appends the new elements at the appropriate (implicitly maintained) index. For instance if on utilizing the <a href="#">DEFINE_ARRAY Procedure</a> a batch size (the <code>cnt</code> parameter) of 10 rows was specified and a start index ( <code>lower_bound</code> ) of 1 was specified, then the first call to this subprogram after calling the <a href="#">FETCH_ROWS Function</a> will populate elements at index 1..10, and the next call will populate elements 11..20, and so on.

## Exceptions

INCONSISTENT\_TYPE (ORA-06562) is raised if the type of the given OUT parameter `value` is different from the actual type of the value. This type was the given type when the column was defined by calling procedure `DEFINE_COLUMN`.

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

## COLUMN\_VALUE\_LONG Procedure

This procedure gets part of the value of a long column.

### Syntax

```
DBMS_SQL.COLUMN_VALUE_LONG (
  c           IN  INTEGER,
  position    IN  INTEGER,
  length      IN  INTEGER,
  offset      IN  INTEGER,
  value       OUT VARCHAR2,
  value_length OUT INTEGER);
```

### Pragmas

```
pragma restrict_references (column_value_long, RNDS, WNDS);
```

### Parameters

**Table 136–11** COLUMN\_VALUE\_LONG Procedure Parameters

Parameter	Description
c	Cursor ID number of the cursor from which to get the value.
position	Position of the column of which to get the value.
length	Number of bytes of the long value to fetch.
offset	Offset into the long field for start of fetch.
value	Value of the column as a VARCHAR2.
value_length	Number of bytes actually returned in value.

## DEFINE\_ARRAY Procedure

This procedure defines the collection for column into which you want to fetch rows (with a `FETCH_ROWS` call). This procedure lets you do batch fetching of rows from a single `SELECT` statement. A single fetch call brings over a number of rows into the PL/SQL aggregate object.

When you fetch the rows, they are copied into `DBMS_SQL` buffers until you run a `COLUMN_VALUE` call, at which time the rows are copied into the table that was passed as an argument to the `COLUMN_VALUE` call.

### Scalar and LOB Types for Collections

You can declare a local variable as one of the following table-item types, and then fetch any number of rows into it using `DBMS_SQL`. (These are the same types as you can specify for the `BIND_ARRAY` procedure.)

```

TYPE binary_double_table
           IS TABLE OF BINARY_DOUBLE INDEX BY BINARY_INTEGER;
TYPE binary_float_table
           IS TABLE OF BINARY_FLOAT INDEX BY BINARY_INTEGER;
TYPE bfile_table IS TABLE OF BFILE INDEX BY BINARY_INTEGER;
TYPE blob_table IS TABLE OF BLOB INDEX BY BINARY_INTEGER;
TYPE clob_table IS TABLE OF CLOB INDEX BY BINARY_INTEGER;
TYPE date_table IS TABLE OF DATE INDEX BY BINARY_INTEGER;
TYPE interval_day_to_second_Table
           IS TABLE OF dsinterval_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE interval_year_to_MONTH_Table
           IS TABLE OF yminterval_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE number_table IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
TYPE time_table IS TABLE OF time_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE time_with_time_zone_table
           IS TABLE OF time_tz_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE timestamp_table
           IS TABLE OF timestamp_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE timestamp_with_ltz_Table
           IS TABLE OF timestamp_ltz_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE timestamp_with_time_zone_Table
           IS TABLE OF timestamp_tz_unconstrained
                                   INDEX BY BINARY_INTEGER;
TYPE urowid_table IS TABLE OF UROWID INDEX BY BINARY_INTEGER;
TYPE varchar2_table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

```

### Syntax

```

DBMS_SQL.DEFINE_ARRAY (
  c           IN INTEGER,
  position    IN INTEGER,
  <table_variable> IN <datatype>
  cnt        IN INTEGER,
  lower_bnd  IN INTEGER);

```



Where `<table_variable>` and its corresponding `<datatype>` can be any one of the following matching pairs, `DEFINE_ARRAY` being overloaded to accept different datatypes:

<code>&lt;clob_tab&gt;</code>	<code>Clob_Table</code>
<code>&lt;bflt_tab&gt;</code>	<code>Binary_Float_Table</code>
<code>&lt;bdbl_tab&gt;</code>	<code>Binary_Double_Table</code>
<code>&lt;blob_tab&gt;</code>	<code>Blob_Table</code>
<code>&lt;bfile_tab&gt;</code>	<code>Bfile_Table</code>
<code>&lt;date_tab&gt;</code>	<code>Date_Table</code>
<code>&lt;num_tab&gt;</code>	<code>Number_Table</code>
<code>&lt;urowid_tab&gt;</code>	<code>Urowid_Table</code>
<code>&lt;vchr2_tab&gt;</code>	<code>Varchar2_Table</code>
<code>&lt;tm_tab&gt;</code>	<code>Time_Table</code>
<code>&lt;ttz_tab&gt;</code>	<code>Time_With_Time_Zone_Table</code>
<code>&lt;tms_tab&gt;</code>	<code>Timestamp_Table</code>
<code>&lt;tstz_tab&gt;</code>	<code>Timestamp_With_ltz_Table;</code>
<code>&lt;tstz_tab&gt;</code>	<code>Timestamp_With_Time_Zone_Table</code>
<code>&lt;ids_tab&gt;</code>	<code>Interval_Day_To_Second_Table</code>
<code>&lt;iym_tab&gt;</code>	<code>Interval_Year_To_Month_Table</code>

## Pragmas

```
pragma restrict_references(define_array,RNDS,WNDS);
```

The subsequent `FETCH_ROWS` call fetch "count" rows. When the `COLUMN_VALUE` call is made, these rows are placed in positions `lower_bnd`, `lower_bnd+1`, `lower_bnd+2`, and so on. While there are still rows coming, the user keeps issuing `FETCH_ROWS/COLUMN_VALUE` calls. The rows keep accumulating in the table specified as an argument in the `COLUMN_VALUE` call.

## Parameters

**Table 136–12** *DEFINE\_ARRAY Procedure Parameters*

Parameter	Description
<code>c</code>	ID number of the cursor to which you want to bind an array.
<code>position</code>	Relative position of the column in the array being defined. The first column in a statement has position 1.
<code>table_variable</code>	Local variable that has been declared as <code>&lt;datatype&gt;</code> .
<code>cnt</code>	Number of rows that must be fetched.
<code>lower_bnd</code>	Results are copied into the collection, starting at this lower bound index.

## Usage Notes

The count (`cnt`) must be an integer greater than zero; otherwise an exception is raised. The `lower_bnd` can be positive, negative, or zero. A query on which a `DEFINE_ARRAY` call was issued cannot contain array binds.

## Examples

```
PROCEDURE BULK_PLSQL(deptid NUMBER)
  TYPE namelist IS TABLE OF employees.last_name%TYPE;
  TYPE sallist IS TABLE OF employees.salary%TYPE;
  names      namelist;
  sals       sallist;
```

```

c          NUMBER;
r          NUMBER;
sql_stmt VARCHAR2(32767) :=
    'SELECT last_name, salary FROM employees WHERE department_id = :b1';

BEGIN
  c := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(c, sql_stmt, dbms_sql.native);

  DBMS_SQL.BIND_VARIABLE(c, 'b1', deptid);

  DBMS_SQL.DEFINE_ARRAY(c, 1, names, 5);
  DBMS_SQL.DEFINE_ARRAY(c, 2, sals, 5);

  r := DBMS_SQL.EXECUTE(c);

  LOOP
    r := DBMS_SQL.FETCH_ROWS(c);
    DBMS_SQL.COLUMN_VALUE(c, 1, names);
    DBMS_SQL.COLUMN_VALUE(c, 2, sals);
    EXIT WHEN r != 5;
  END LOOP;

  DBMS_SQL.CLOSE_CURSOR(c);

  -- loop through the names and sals collections
  FOR i IN names.FIRST .. names.LAST LOOP
    DBMS_OUTPUT.PUT_LINE('Name = ' || names(i) || ', salary = ' || sals(i));
  END LOOP;
END;
/

```

**See Also:** ["Examples 6 and 7: Defining an Array"](#) on page 136-17 for examples of how to define collections.

## DEFINE\_COLUMN Procedures

This procedure defines a column to be selected from the given cursor. This procedure is only used with `SELECT` cursors.

The column being defined is identified by its relative position in the `SELECT` list of the statement in the given cursor. The type of the `COLUMN` value determines the type of the column being defined.

See also the [DEFINE\\_COLUMN\\_CHAR Procedure](#), [DEFINE\\_COLUMN\\_LONG Procedure](#), [DEFINE\\_COLUMN\\_RAW Procedure](#) and [DEFINE\\_COLUMN\\_ROWID Procedure](#).

### Syntax

```
DBMS_SQL.DEFINE_COLUMN (
  c           IN INTEGER,
  position   IN INTEGER,
  column     IN <datatype>);
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_UNCONSTRAINED
TIME_TZ_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
YMINTERVAL_UNCONSTRAINED
user-defined object types
collections (VARRAYs and nested tables)
REFs
Opaque types
```

Note that `DEFINE_COLUMN` is overloaded to accept different datatypes.

The following syntax is also supported for the `DEFINE_COLUMN` procedure:

```
DBMS_SQL.DEFINE_COLUMN (
  c           IN INTEGER,
  position   IN INTEGER,
  column     IN VARCHAR2 CHARACTER SET ANY_CS,
  column_size IN INTEGER);
```

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

### Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

## Parameters

**Table 136–13** *DEFINE\_COLUMN Procedure Parameters*

Parameter	Description
<code>c</code>	ID number of the cursor for the row being defined to be selected
<code>position</code>	Relative position of the column in the row being defined. The first column in a statement has position 1.
<code>column</code>	Value of the column being defined. The type of this value determines the type for the column being defined.
<code>column_size</code>	Maximum expected size of the column value in bytes for columns of type <code>VARCHAR2</code> , and <code>RAW</code> .

## Usage Notes

When using character length semantics the maximum number of bytes that can be returned for a column value of type `VARCHAR2` is calculated as: `column_size * maximum character byte size for the current character set`. For example, specifying the `column_size` as 10 means that a maximum of 30 (10\*3) bytes can be returned when using character length semantics with a UTF8 character set regardless of the number of characters this represents.

## DEFINE\_COLUMN\_CHAR Procedure

This procedure defines a column with CHAR data to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the [DEFINE\\_COLUMN Procedures](#), [DEFINE\\_COLUMN\\_LONG Procedure](#), [DEFINE\\_COLUMN\\_RAW Procedure](#) and [DEFINE\\_COLUMN\\_ROWID Procedure](#).

### Syntax

```
DBMS_SQL.DEFINE_COLUMN_CHAR (
  c           IN INTEGER,
  position   IN INTEGER,
  column     IN CHAR CHARACTER SET ANY_CS,
  column_size IN INTEGER);
```

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

### Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

### Parameters

**Table 136–14** DEFINE\_COLUMN\_CHAR Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected
position	Relative position of the column in the row being defined. The first column in a statement has position 1.
column	Value of the column being defined. The type of this value determines the type for the column being defined.
column_size	Maximum expected size of the column value in characters for columns of type CHAR.

## DEFINE\_COLUMN\_LONG Procedure

This procedure defines a LONG column for a SELECT cursor. The column being defined is identified by its relative position in the SELECT list of the statement for the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the [DEFINE\\_COLUMN Procedures](#), [DEFINE\\_COLUMN\\_CHAR Procedure](#), [DEFINE\\_COLUMN\\_RAW Procedure](#) and [DEFINE\\_COLUMN\\_ROWID Procedure](#).

### Syntax

```
DBMS_SQL.DEFINE_COLUMN_LONG (  
    c           IN INTEGER,  
    position    IN INTEGER);
```

### Parameters

**Table 136–15** DEFINE\_COLUMN\_LONG Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected.
position	Relative position of the column in the row being defined. The first column in a statement has position 1.

## DEFINE\_COLUMN\_RAW Procedure

This procedure defines a column of type RAW to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the [DEFINE\\_COLUMN Procedures](#), [DEFINE\\_COLUMN\\_CHAR Procedure](#), [DEFINE\\_COLUMN\\_LONG Procedure](#) and [DEFINE\\_COLUMN\\_ROWID Procedure](#).

### Syntax

```
DBMS_SQL.DEFINE_COLUMN_RAW (
  c           IN INTEGER,
  position   IN INTEGER,
  column     IN RAW,
  column_size IN INTEGER);
```

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

### Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

### Parameters

**Table 136–16** *DEFINE\_COLUMN Procedure Parameters*

Parameter	Description
c	ID number of the cursor for the row being defined to be selected
position	Relative position of the column in the row being defined. The first column in a statement has position 1.
column	Value of the column being defined. The type of this value determines the type for the column being defined.
column_size	Maximum expected size of the column value in bytes for columns of and RAW

## DEFINE\_COLUMN\_ROWID Procedure

This procedure defines a column of type ROWID to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

See also the [DEFINE\\_COLUMN Procedures](#), [DEFINE\\_COLUMN\\_CHAR Procedure](#), [DEFINE\\_COLUMN\\_LONG Procedure](#) and [DEFINE\\_COLUMN\\_RAW Procedure](#).

### Syntax

```
DBMS_SQL.DEFINE_COLUMN_ROWID (
  c           IN INTEGER,
  position   IN INTEGER,
  column     IN ROWID);
```

**See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide*

### Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

### Parameters

**Table 136–17** DEFINE\_COLUMN Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected
position	Relative position of the column in the row being defined. The first column in a statement has position 1.
column	Value of the column being defined. The type of this value determines the type for the column being defined.



## DESCRIBE\_COLUMNS Procedure

This procedure describes the columns for a cursor opened and parsed through DBMS\_SQL.

### Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS (  
    c           IN  INTEGER,  
    col_cnt    OUT INTEGER,  
    desc_t     OUT DESC_TAB);
```

### Parameters

**Table 136-18** *DESCRIBE\_COLUMNS Procedure Parameters*

Parameter	Description
c	ID number of the cursor for the columns being described
col_cnt	Number of columns in the select list of the query
desc_t	Describe table to fill in with the description of each of the columns of the query

**See Also:** ["Example 8: Describe Columns"](#) on page 136-19 illustrates how to use DESCRIBE\_COLUMNS.

## DESCRIBE\_COLUMNS2 Procedure

This procedure describes the specified column. This is an alternative to [DESCRIBE\\_COLUMNS Procedure](#).

### Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS2 (  
  c           IN  INTEGER,  
  col_cnt    OUT INTEGER,  
  desc_t     OUT DESC_TAB2);
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(describe_columns2,WNDS);
```

### Parameters

**Table 136–19** DESCRIBE\_COLUMNS2 Procedure Parameters

Parameter	Description
c	ID number of the cursor for the columns being described.
col_cnt	Number of columns in the select list of the query.
desc_t	Describe table to fill in with the description of each of the columns of the query. This table is indexed from one to the number of elements in the select list of the query.

## DESCRIBE\_COLUMNS3 Procedure

This procedure describes the specified column. This is an alternative to [DESCRIBE\\_COLUMNS Procedure](#).

### Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS3 (
  c          IN  INTEGER,
  col_cnt   OUT INTEGER,
  desc_t    OUT DESC_TAB3);
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES(describe_columns3,WNDS);
```

### Parameters

**Table 136–20 DESCRIBE\_COLUMNS3 Procedure Parameters**

Parameter	Description
c	ID number of the cursor for the columns being described.
col_cnt	Number of columns in the select list of the query.
desc_t	Describe table to fill in with the description of each of the columns of the query. This table is indexed from one to the number of elements in the select list of the query.

### Usage Notes

The cursor passed in by the cursor ID has to be OPENED and PARSED, otherwise an "invalid cursor id" error is raised.

### Examples

```
CREATE TYPE PROJECT_T AS OBJECT
  ( projname      VARCHAR2(20),
    mgr           VARCHAR2(20))
/

CREATE TABLE projecttab(deptno NUMBER, project HR.PROJECT_T)
/

DECLARE
  curid      NUMBER;
  desctab   DBMS_SQL.DESC_TAB3;
  colcnt    NUMBER;
  sql_stmt  VARCHAR2(200) := 'select * from projecttab';
BEGIN

  curid := DBMS_SQL.OPEN_CURSOR;

  DBMS_SQL.PARSE(curid, sql_stmt, DBMS_SQL.NATIVE);

  DBMS_SQL.DESCRIBE_COLUMNS3(curid, colcnt, desctab);

  FOR i IN 1 .. colcnt LOOP
    IF desctab(i).col_type = 109 THEN
```

```
        DBMS_OUTPUT.PUT(desctab(i).col_name || ' is user-defined type: ');
        DBMS_OUTPUT.PUT_LINE(desctab(i).col_schema_name || '.' ||
                               desctab(i).col_type_name);
    END IF;
END LOOP;

DBMS_SQL.CLOSE_CURSOR(curid);
END;
/
```

Output:

```
PROJECT is user-defined type: HR.PROJECT_T
```

## EXECUTE Function

This function executes a given cursor. This function accepts the ID number of the cursor and returns the number of rows processed. The return value is only valid for INSERT, UPDATE, and DELETE statements; for other types of statements, including DDL, the return value is undefined and should be ignored.

### Syntax

```
DBMS_SQL.EXECUTE (  
    c    IN INTEGER)  
RETURN INTEGER;
```

### Parameters

**Table 136–21 EXECUTE Function Parameters**

Parameter	Description
c	Cursor ID number of the cursor to execute.

### Return Values

Returns number of rows processed

### Usage Notes

The DBMS\_SQL cursor that is returned by the [TO\\_CURSOR\\_NUMBER Function](#) performs in the same way as a DBMS\_SQL cursor that has already been executed. Consequently, calling EXECUTE for this cursor will cause an error.

## EXECUTE\_AND\_FETCH Function

This function executes the given cursor and fetches rows. This function provides the same functionality as calling `EXECUTE` and then calling `FETCH_ROWS`. Calling `EXECUTE_AND_FETCH` instead, however, may reduce the number of network round-trips when used against a remote database.

The `EXECUTE_AND_FETCH` function returns the number of rows actually fetched.

### Syntax

```
DBMS_SQL.EXECUTE_AND_FETCH (  
    c          IN INTEGER,  
    exact      IN BOOLEAN DEFAULT FALSE)  
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (execute_and_fetch, WNDS);
```

### Parameters

**Table 136–22 EXECUTE\_AND\_FETCH Function Parameters**

Parameter	Description
<code>c</code>	ID number of the cursor to execute and fetch.
<code>exact</code>	Set to <code>TRUE</code> to raise an exception if the number of rows actually matching the query differs from one.  Note: Oracle does not support the exact fetch <code>TRUE</code> option with <code>LONG</code> columns.  Even if an exception is raised, the rows are still fetched and available.

### Return Values

Returns designated rows

## FETCH\_ROWS Function

This function fetches a row from a given cursor. You can call `FETCH_ROWS` repeatedly as long as there are rows remaining to be fetched. These rows are retrieved into a buffer, and must be read by calling `COLUMN_VALUE`, for each column, after each call to `FETCH_ROWS`.

The `FETCH_ROWS` function accepts the ID number of the cursor to fetch, and returns the number of rows actually fetched.

### Syntax

```
DBMS_SQL.FETCH_ROWS (  
    c          IN INTEGER)  
    RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references(fetch_rows,WNDS);
```

### Parameters

**Table 136–23** *FETCH\_ROWS Function Parameters*

Parameter	Description
c	ID number.

### Return Values

Returns a row from a given cursor

## IS\_OPEN Function

This function checks to see if the given cursor is currently open.

### Syntax

```
DBMS_SQL.IS_OPEN (  
    c          IN INTEGER)  
    RETURN BOOLEAN;
```

### Pragmas

```
pragma restrict_references(is_open,RNDS,WNDS);
```

### Parameters

**Table 136–24 IS\_OPEN Function Parameters**

Parameter	Description
c	Cursor ID number of the cursor to check.

### Return Values

Returns `TRUE` for any cursor number that has been opened but not closed, and `FALSE` for a `NULL` cursor number. Note that the [CLOSE\\_CURSOR Procedure](#) Procedure `NULLs` out the cursor variable passed to it.

### Exceptions

`ORA-29471 DBMS_SQL access denied`: This is raised if an invalid cursor ID number is detected. Once a session has encountered and reported this error, every subsequent `DBMS_SQL` call in the same session will raise this error, meaning that `DBMS_SQL` is non-operational for this session.



## LAST\_ERROR\_POSITION Function

This function returns the byte offset in the SQL statement text where the error occurred. The first character in the SQL statement is at position 0.

### Syntax

```
DBMS_SQL.LAST_ERROR_POSITION  
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (last_error_position, RNDS, WNDS);
```

### Return Values

Returns the byte offset in the SQL statement text where the error occurred

### Usage Notes

Call this function after a `PARSE` call, before any other `DBMS_SQL` procedures or functions are called.

## LAST\_ROW\_COUNT Function

This function returns the cumulative count of the number of rows fetched.

### Syntax

```
DBMS_SQL.LAST_ROW_COUNT  
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (last_row_count, RNDS, WNDS);
```

### Return Values

Returns the cumulative count of the number of rows fetched

### Usage Notes

Call this function after a `FETCH_ROWS` or an `EXECUTE_AND_FETCH` call. If called after an `EXECUTE` call, then the value returned is zero.

## LAST\_ROW\_ID Function

This function returns the ROWID of the last row processed.

### Syntax

```
DBMS_SQL.LAST_ROW_ID  
RETURN ROWID;
```

### Pragmas

```
pragma restrict_references (last_row_id, RNDS, WNDS);
```

### Return Values

Returns the ROWID of the last row processed

### Usage Notes

Call this function after a `FETCH_ROWS` or an `EXECUTE_AND_FETCH` call.

## LAST\_SQL\_FUNCTION\_CODE Function

This function returns the SQL function code for the statement. These codes are listed in the *Oracle Call Interface Programmer's Guide*.

### Syntax

```
DBMS_SQL.LAST_SQL_FUNCTION_CODE  
RETURN INTEGER;
```

### Pragmas

```
pragma restrict_references (last_sql_function_code, RNDS, WNDS);
```

### Return Values

Returns the SQL function code for the statement

### Usage Notes

You should call this function immediately after the SQL statement is run; otherwise, the return value is undefined.

## OPEN\_CURSOR Function

This procedure opens a new cursor. The second overload takes a `security_level` parameter to apply fine-grained control to the security of the opened cursor.

When you no longer need this cursor, you must close it explicitly by calling the [CLOSE\\_CURSOR Procedure](#).

### Syntax

```
DBMS_SQL.OPEN_CURSOR
RETURN INTEGER;

DBMS_SQL.OPEN_CURSOR (
    security_level IN INTEGER)
RETURN INTEGER;
```

### Parameters

**Table 136–25 OPEN\_CURSOR Function Parameters**

Parameter	Description
<code>security_level</code>	<p>Specifies the level of security protection to enforce on the opened cursor. Valid security level values are 0, 1, and 2. When a NULL argument value is provided to this overload, as well as for cursors opened using the overload of <code>open_cursor</code> without the <code>security_level</code> parameter, the default security level value 1 will be enforced on the opened cursor.</p> <ul style="list-style-type: none"> <li>▪ Level 0 - allows all DBMS_SQL operations on the cursor without any security checks. The cursor may be fetched from, and even re-bound and re-executed, by code running with a different effective userid or roles than those in effect at the time the cursor was parsed. This level of security is off by default.</li> <li>▪ Level 1 - requires that the effective userid and roles of the caller to DBMS_SQL for bind and execute operations on this cursor must be the same as those of the caller of the most recent parse operation on this cursor.</li> <li>▪ Level 2 - requires that the effective userid and roles of the caller to DBMS_SQL for all bind, execute, define, describe, and fetch operations on this cursor must be the same as those of the caller of the most recent parse operation on this cursor.</li> </ul>

### Pragmas

```
pragma restrict_references (open_cursor, RNDS, WNDS);
```

### Return Values

Returns the cursor ID number of the new cursor

### Usage Notes

You can use cursors to run the same SQL statement repeatedly or to run a new SQL statement. When a cursor is reused, the contents of the corresponding cursor data area are reset when the new SQL statement is parsed. It is never necessary to close and reopen a cursor before reusing it.

## PARSE Procedures

This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.

There are three versions of the PARSE procedure:

- Taking a VARCHAR2 statement as an argument
- Taking VARCHAR2a, table of VARCHAR2(32767), as an argument. The VARCHAR2A overload version of the procedure concatenates elements of a PL/SQL table statement and parses the resulting string. You can use this procedure to parse a statement that is longer than the limit for a single VARCHAR2 variable by splitting up the statement.
- Taking a CLOB statement as an argument. You can use the CLOB overload version of the parse procedure to parse a SQL statement larger than 32K bytes.

### Syntax

```
DBMS_SQL.PARSE (
    c                IN    INTEGER,
    statement        IN    VARCHAR2,
    language_flag    IN    INTEGER,
    edition          IN    VARCHAR2 DEFAULT NULL,
    apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL,
    fire_apply_trigger IN    BOOLEAN DEFAULT TRUE);
```

```
DBMS_SQL.PARSE (
    c                IN    INTEGER,
    statement        IN    CLOB,
    language_flag    IN    INTEGER,
    edition          IN    VARCHAR2 DEFAULT NULL,
    apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL,
    fire_apply_trigger IN    BOOLEAN DEFAULT TRUE);
```

```
DBMS_SQL.PARSE (
    c                IN    INTEGER,
    statement        IN    VARCHAR2A,
    lb              IN    INTEGER,
    ub              IN    INTEGER,
    lfflg          IN    BOOLEAN,
    language_flag    IN    INTEGER,
    edition          IN    VARCHAR2 DEFAULT NULL,
    apply_crossedition_trigger IN VARCHAR2 DEFAULT NULL,
    fire_apply_trigger IN    BOOLEAN DEFAULT TRUE);
```

## Parameters

**Table 136–26 PARSE Procedure Parameters**

Parameter	Description
c	ID number of the cursor in which to parse the statement.
statement	<p>SQL statement to be parsed. SQL statements larger than 32K that may be stored in CLOBs.</p> <p>Unlike PL/SQL statements, your SQL statement should not include a final semicolon. For example:</p> <pre>DBMS_SQL.PARSE(cursor1, 'BEGIN proc; END;', 2); DBMS_SQL.PARSE(cursor1, 'INSERT INTO tab VALUES(1)', 2);</pre>
lb	Lower bound for elements in the statement
ub	Upper bound for elements in the statement
lfflg	If TRUE, then insert a linefeed after each element on concatenation.
language_flag	<p>Determines how Oracle handles the SQL statement. The following options are recognized:</p> <ul style="list-style-type: none"> <li>▪ V6 (or 0) specifies version 6 behavior.</li> <li>▪ NATIVE (or 1) specifies normal behavior for the database to which the program is connected.</li> <li>▪ V7 (or 2) specifies Oracle database version 7 behavior.</li> </ul>
edition	<p>Specifies the edition to run the statement in. Passing NULL indicates the statement should to run in the caller's current edition. The contents of the string are processed as a SQL identifier; double-quotes must surround the remainder of the string if special characters or lower case characters are present in the edition's actual name, and if double-quotes are not used the contents will be uppercased. If the edition is specified with a non-NULL value, the user with which the statement is to be executed must have USE privilege on the named edition.</p>
apply_crossedition_trigger	<p>Specifies the unqualified name of a forward crossedition trigger that is to be applied to the specified SQL. The name is resolved using the edition and current_schema setting in which the statement is to be executed. The trigger must be owned by the user that will execute the statement.</p> <p>If a non-NULL value is specified, the specified crossedition trigger will be executed assuming fire_apply_trigger is TRUE, the trigger is enabled, the trigger is defined on the table which is the target of the statement, the type of the statement matches the trigger's dml_event_clause, any effective WHEN and UPDATE OF restrictions are satisfied, and so on. Other forward crossedition triggers may also be executed, selected using the "crossedition trigger DML rules" applied as if the specified trigger was doing a further DML to the table that is the target of the statement. Non-crossedition triggers and reverse crossedition triggers will not be executed.</p> <p>The contents of the string are processed as a SQL identifier; double-quotes must surround the remainder of the string if special characters or lower case characters are present in the trigger's actual name, and if double-quotes are not used, the contents will be uppercased.</p>

**Table 136–26 (Cont.) PARSE Procedure Parameters**

Parameter	Description
fire_apply_trigger	Indicates whether the specified <code>apply_crossedition_trigger</code> is itself to be executed, or should only be a guide used in selecting other triggers. This is typically set <code>FALSE</code> when the statement is a replacement for the actions the <code>apply_crossedition_trigger</code> would itself perform. If <code>FALSE</code> , the specified trigger is not executed, but other triggers are still selected for firing as if the specified trigger was doing a DML to the table that is the target of the statement. The <code>apply_crossedition_trigger</code> and <code>fire_apply_trigger</code> parameters are ignored if the statement is not a DML.

## Usage Notes

- Using `DBMS_SQL` to dynamically run DDL statements can result in the program hanging. For example, a call to a procedure in a package results in the package being locked until the execution returns to the user side. Any operation that results in a conflicting lock, such as dynamically trying to drop the package before the first lock is released, results in a hang.
- Because client-side code cannot reference remote package variables or constants, you must explicitly use the values of the constants.

For example, the following code does *not* compile on the client:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, DBMS_SQL.NATIVE); -- uses constant
DBMS_SQL.NATIVE
```

The following code works on the client, because the argument is explicitly provided:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, 1); -- compiles on the client
```

- The `VARCHAR2S` type is currently supported for backward compatibility of legacy code. However, you are advised to use `VARCHAR2A` both for its superior capability and because `VARCHAR2S` will be deprecated in a future release.
- To parse SQL statements larger than 32 KB, the new `CLOB` overload version of the `PARSE` procedure can be used instead of the `VARCHAR2A` overload

## Exceptions

If you create a type/procedure/function/package using `DBMS_SQL` that has compilation warnings, an `ORA-24344` exception is raised, and the procedure is still created.



## TO\_CURSOR\_NUMBER Function

This function takes an OPENed strongly or weakly-typed ref cursor and transforms it into a DBMS\_SQL cursor number.

### Syntax

```
DBMS_SQL.TO_CURSOR_NUMBER(
  rc IN OUT SYS_REFCURSOR)
RETURN INTEGER;
```

### Parameters

**Table 136–27 TO\_CURSOR\_NUMBER Function Parameters**

Parameter	Description
rc	REF CURSOR to be transformed into a cursor number

### Return Values

Returns a DBMS\_SQL manageable cursor number transformed from a REF CURSOR

### Usage Notes

- The REF CURSOR passed in has to be OPENed, otherwise an error is raised.
- Once the REF CURSOR is transformed into a DBMS\_SQL cursor number, the REF CURSOR is no longer accessible by any native dynamic SQL operations.
- The DBMS\_SQL cursor that is returned by this subprogram performs in the same way as a DBMS\_SQL cursor that has already been executed.

### Examples

```
CREATE OR REPLACE PROCEDURE DO_QUERY(sql_stmt VARCHAR2) IS
  TYPE CurType IS REF CURSOR;
  src_cur      CurType;
  curid        NUMBER;
  desctab      DBMS_SQL.DESCTAB;
  colcnt       NUMBER;
  namevar      VARCHAR2(50);
  numvar       NUMBER;
  datevar      DATE;
  empno        NUMBER := 100;
BEGIN

  -- sql_stmt := 'select ..... from employees where employee_id = :b1';
  OPEN src_cur FOR sql_stmt USING empno;

  -- Switch from native dynamic SQL to DBMS_SQL
  curid := DBMS_SQL.TO_CURSOR_NUMBER (src_cur);

  DBMS_SQL.DESCRIBE_COLUMNS(curid, colcnt, desctab);

  -- Define columns
  FOR i IN 1 .. colcnt LOOP
    IF desctab(i).col_type = 2 THEN
      DBMS_SQL.DEFINE_COLUMN(curid, i, numvar);
```

```
        ELSIF desctab(i).col_type = 12 THEN
            DBMS_SQL.DEFINE_COLUMN(curid, i, datevar);
        .....
        ELSE
            DBMS_SQL.DEFINE_COLUMN(curid, i, namevar, 25);
        END IF;
    END LOOP;

-- Fetch Rows
    WHILE DBMS_SQL.FETCH_ROWS(curid) > 0 LOOP
        FOR i IN 1 .. colcnt LOOP
            IF (desctab(i).col_type = 1) THEN
                DBMS_SQL.COLUMN_VALUE(curid, i, namevar);
            ELSIF (desctab(i).col_type = 2) THEN
                DBMS_SQL.COLUMN_VALUE(curid, i, numvar);
            ELSIF (desctab(i).col_type = 12) THEN
                DBMS_SQL.COLUMN_VALUE(curid, i, datevar);
        .....
            END IF;
        END LOOP;
    END LOOP;

    DBMS_SQL.CLOSE_CURSOR(curid);
END;
/
```

## TO\_REFCURSOR Function

This function takes an OPENed, PARSEd, and EXECUTEd cursor and transforms/migrates it into a PL/SQL manageable REF CURSOR (a weakly-typed cursor) that can be consumed by PL/SQL native dynamic SQL switched to use native dynamic SQL. This subprogram is only used with SELECT cursors.

### Syntax

```
DBMS_SQL.TO_REFCURSOR (
    cursor_number IN OUT INTEGER)
RETURN SYS_REFCURSOR;
```

### Parameters

**Table 136–28 TO\_REFCURSOR Function Parameters**

Parameter	Description
cursor_number	Cursor number of the cursor to be transformed into REF CURSOR

### Return Values

Returns a PL/SQL REF CURSOR transformed from a DBMS\_SQL cursor number

### Usage Notes

- The cursor passed in by the cursor\_number has to be OPENed, PARSEd, and EXECUTEd; otherwise an error is raised.
- Once the cursor\_number is transformed into a REF CURSOR, the cursor\_number is no longer accessible by any DBMS\_SQL operations.
- After a cursor\_number is transformed into a REF CURSOR, using DBMS\_SQL.IS\_OPEN to check to see if the cursor\_number is still open results in an error.

### Examples

```
CREATE OR REPLACE PROCEDURE DO_QUERY(mgr_id NUMBER) IS
    TYPE CurType IS REF CURSOR;
    src_cur          CurType;
    curid            NUMBER;
    sql_stmt         VARCHAR2(200);
    ret              INTEGER;
    empnos           DBMS_SQL.Number_Table;
    depts            DBMS_SQL.Number_Table;
BEGIN

    -- DBMS_SQL.OPEN_CURSOR
    curid := DBMS_SQL.OPEN_CURSOR;

    sql_stmt :=
        'SELECT EMPLOYEE_ID, DEPARTMENT_ID from employees where MANAGER_ID = :b1';

    DBMS_SQL.PARSE(curid, sql_stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(curid, 'b1', mgr_id);
    ret := DBMS_SQL.EXECUTE(curid);

    -- Switch from DBMS_SQL to native dynamic SQL
```

```
src_cur := DBMS_SQL.TO_REFCURSOR(curid);

-- Fetch with native dynamic SQL
FETCH src_cur BULK COLLECT INTO empnos, depts;

IF empnos.COUNT > 0 THEN
  DBMS_OUTPUT.PUT_LINE('EMPNO DEPTNO');
  DBMS_OUTPUT.PUT_LINE('-----');
  -- Loop through the empnos and depts collections
  FOR i IN 1 .. empnos.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(empnos(i) || ' ' || depts(i));
  END LOOP;
END IF;

-- Close cursor
CLOSE src_cur;
END;
/
```

## VARIABLE\_VALUE Procedures

This procedure returns the value of the named variable for a given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with returning clause.

### Syntax

```
DBMS_SQL.VARIABLE_VALUE (
  c           IN  INTEGER,
  name        IN  VARCHAR2,
  value       OUT NOCOPY <datatype>);
```

Where <datatype> can be any one of the following types:

```
BINARY_DOUBLE
BINARY_FLOAT
BFILE
BLOB
CLOB CHARACTER SET ANY_CS
DATE
DSINTERVAL_UNCONSTRAINED
NUMBER
TIME_TZ_UNCONSTRAINED
TIME_UNCONSTRAINED
TIMESTAMP_LTZ_UNCONSTRAINED
TIMESTAMP_TZ_UNCONSTRAINED
TIMESTAMP_UNCONSTRAINED
UROWID
VARCHAR2 CHARACTER SET ANY_CS
YMINTERVAL_UNCONSTRAINED
user-defined object types
collections (VARRAYs and nested tables)
REFs
Opaque types
```

For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.VARIABLE_VALUE_CHAR (
  c           IN  INTEGER,
  name        IN  VARCHAR2,
  value       OUT CHAR CHARACTER SET ANY_CS);
```

```
DBMS_SQL.VARIABLE_VALUE_RAW (
  c           IN  INTEGER,
  name        IN  VARCHAR2,
  value       OUT RAW);
```

```
DBMS_SQL.VARIABLE_VALUE_ROWID (
  c           IN  INTEGER,
  name        IN  VARCHAR2,
  value       OUT ROWID);
```

The following syntax enables the VARIABLE\_VALUE procedure to accommodate bulk operations:

```
DBMS_SQL.VARIABLE_VALUE (
  c           IN  INTEGER,
  name        IN  VARCHAR2,
```

```
value          OUT NOCOPY <table_type>;
```

For bulk operations, <table\_type> can be:

```
Binary_Double_Table
Binary_Float_Table
Bfile_Table
Blob_Table
Clob_Table
Date_Table
Interval_Day_To_Second_Table
Interval_Year_To_Month_Table
Number_Table
Time_Table
Time_With_Time_Zone_Table
Timestamp_Table
Timestamp_With_ltz_Table;
Timestamp_With_Time_Zone_Table
Urowid_Table
Varchar2_Table
```

## Pragmas

```
pragma restrict_references(variable_value,RNDS,WNDS);
```

## Parameters

**Table 136–29 VARIABLE\_VALUE Procedure Parameters**

Parameter	Description
c	ID number of the cursor from which to get the values.
name	Name of the variable for which you are retrieving the value.
value	<ul style="list-style-type: none"> <li>▪ Single row option: Returns the value of the variable for the specified position. Oracle raises the exception <code>ORA-06562, inconsistent_type</code>, if the type of this output parameter differs from the actual type of the value, as defined by the call to <code>BIND_VARIABLE</code>.</li> <li>▪ Array option: Local variable that has been declared &lt;table_type&gt;. For bulk operations, value is an <code>OUT NOCOPY</code> parameter.</li> </ul>

The DBMS\_SQLDIAG package provides an interface to the SQL Diagnosability functionality.

**See Also:** *Oracle Database Administrator's Guide* for more information about "Managing Diagnostic Data"

This chapter contains the following topics:

- [Using DBMS\\_SQLDIAG](#)
  - Overview
  - Constants
  - Examples
- [Summary of DBMS\\_SQLDIAG Subprograms](#)

## Using DBMS\_SQLDIAG

- [Overview](#)
- [Constants](#)
- [Examples](#)



## Overview

In the rare case that a SQL statement fails with a critical error, you can run the SQL Repair Advisor to try to repair the failed statement.

This section covers the following topics:

- [About the SQL Repair Advisor](#)
- [Running the SQL Repair Advisor](#)
- [Removing a SQL Patch](#)

### About the SQL Repair Advisor

You run the SQL Repair Advisor after a SQL statement fails with a critical error. The advisor analyzes the statement and in many cases recommends a patch to repair the statement. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternate execution plan for future executions.

### Running the SQL Repair Advisor

You run the SQL Repair Advisor by creating and executing a diagnostic task using the `CREATE_DIAGNOSIS_TASK` and `EXECUTE_DIAGNOSIS_TASK` respectively. The SQL Repair Advisor first reproduces the critical error and then tries to produce a workaround in the form of SQL patch.

#### 1. Identify the problem SQL statement

Consider the SQL statement that gives a critical error:

```
DELETE FROM t t1 WHERE t1.a = 'a' AND ROWID <> (SELECT MAX(ROWID) FROM t t2
WHERE t1.a= t2.a AND t1.b = t2.b AND t1.d=t2.d)
```

You use the SQL Repair advisor to repair this critical error.

#### 2. Create a diagnosis task

Invoke `DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK`. You can specify an optional task name, an optional time limit for the advisor task, and problem type. In the example below, we specify the SQL text, the task name as 'error\_task' and a problem type as 'DBMS\_SQLDIAG.PROBLEM\_TYPE\_COMPILATION\_ERROR'.

```
DECLARE
    rep_out          CLOB;
    t_id             VARCHAR2(50);
BEGIN
    t_id := DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK(
        sql_text => 'DELETE FROM t t1 WHERE t1.a = 'a' AND ROWID <> (SELECT
MAX(ROWID) FROM t t2 WHERE t1.a= t2.a AND t1.b = t2.b AND t1.d=t2.d)',
        task_name => 'error_task',
        problem_type =>DBMS_SQLDIAG.PROBLEM_TYPE_COMPILATION_ERROR);
```

#### 3. Execute the diagnosis task

To execute the workaround generation and analysis phase of the SQL Repair Advisor, you call `DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK` with the task ID returned by the `CREATE_DIAGNOSIS_TASK`. After a short delay, the SQL Repair Advisor returns. As part of its execution, the SQL Repair Advisor keeps a record of its findings which can be accessed through the reporting facilities of SQL Repair Advisor.

```
DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK (t_id);
```

#### 4. Report the diagnosis task

The analysis of the diagnosis task is accessed through `dbms_sqldiag.report_diagnosis_task`. If the SQL Repair Advisor was able to find a workaround, it recommends a SQL Patch. A SQL Patch is similar to a SQL profile but unlike the SQL Profile, it is used to workaround compilation or execution errors.

```
rep_out := DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK (t_id, DBMS_SQLDIAG.TYPE_TEXT);

DBMS_OUTPUT.PUT_LINE ('Report : ' || rep_out);

END;
/
```

#### 5. Applying the patch

If a patch recommendation is present in the report, you can run the `ACCEPT_SQL_PATCH` command to accept the patch by invoking `DBMS_SQLDIAG.ACCEPT_SQL_PATCH`. This procedure takes the `task_name` as an argument.

```
EXECUTE DBMS_SQLDIAG.ACCEPT_SQL_PATCH(task_name => 'error_task', task_owner =>
'SYS', replace => TRUE);
```

#### 6. Test the patch

Now that you have accepted the patch, you can rerun the SQL statement. This time, it will not give you the critical error. If you run 'explain plan' for this statement, you will see that a SQL patch was used to generate the plan.

```
DELETE FROM t t1 WHERE t1.a = 'a' AND ROWID <> (select max(rowid) FROM t t2
WHERE t1.a= t2.a AND t1.b = t2.b AND t1.d=t2.d);
```

#### Removing a SQL Patch

In a situation where you obtained an official patch from Oracle to fix an error, or upgraded to the next patchset or release of Oracle which included the fix for the error, you call `DBMS_SQLDIAG.DROP_SQL_PATCH` with the patch name to drop the SQL patch. The patch name can be obtained from the explain plan section or by querying the view `DBA_SQL_PATCHES`.

## Constants

The DBMS\_SQLDIAG package uses the constants shown in the following tables:

- [Table 137-1, "DBMS\\_SQLDIAG Constants - SQLDIAG Advisor Name"](#) describes the name of SQL repair advisor as seen by the advisor framework
- [Table 137-2, "DBMS\\_SQLDIAG Constants - SQLDIAG Advisor Task Scope Parameter Values"](#) describes SQLDIAG advisor task scope parameter values
- [Table 137-3, "DBMS\\_SQLDIAG Constants - SQLDIAG Advisor time\\_limit Constants"](#) describes SQLDIAG advisor time\_limit constants
- [Table 137-4, "DBMS\\_SQLDIAG Constants - Report Type \(possible values\) Constants"](#) describes possible formats for a report
- [Table 137-5, "DBMS\\_SQLDIAG Constants - Report Level \(possible values\) Constants"](#) describes possible levels of detail in the report
- [Table 137-6, "DBMS\\_SQLDIAG Constants - Report Section \(possible values\) Constants"](#) describes possible report sections (comma delimited)
- [Table 137-7, "DBMS\\_SQLDIAG Constants - Problem Type Constants"](#) describes possible values for the problem\_type parameter of the [CREATE\\_DIAGNOSIS\\_TASK Functions](#)
- [Table 137-8, "DBMS\\_SQLDIAG Constants - Findings Filter Constants"](#) describes possible values for the \_sql\_findings\_mode parameter

**Table 137-1 DBMS\_SQLDIAG Constants - SQLDIAG Advisor Name**

Constant	Type	Value	Description
ADV_SQL_DIAG_NAME	VARCHAR2(18)	'SQL Repair Advisor'	Name of SQL repair advisor as seen by the advisor framework

**Table 137-2 DBMS\_SQLDIAG Constants - SQLDIAG Advisor Task Scope Parameter Values**

Constant	Type	Value	Description
SCOPE_COMPREHENSIVE	VARCHAR2(13)	'COMPREHENSIVE'	Detailed analysis of the problem which may take more time to execute
SCOPE_LIMITED	VARCHAR2(7)	'LIMITED'	Brief analysis of the problem

**Table 137-3 DBMS\_SQLDIAG Constants - SQLDIAG Advisor time\_limit Constants**

Constant	Type	Value	Description
TIME_LIMIT_DEFAULT	NUMBER	1800	Default time limit for analysis of the problem

**Table 137–4 DBMS\_SQLDIAG Constants - Report Type (possible values) Constants**

Constant	Type	Value	Description
TYPE_HTML	VARCHAR2(4)	'HTML'	Report from the <a href="#">REPORT_DIAGNOSIS_TASK</a> Function in HTML form
TYPE_TEXT	VARCHAR2(4)	'TEXT'	Report from the <a href="#">REPORT_DIAGNOSIS_TASK</a> Function in text form
TYPE_XML	VARCHAR2(3)	'XML'	Report from the <a href="#">REPORT_DIAGNOSIS_TASK</a> Function in XML form

**Table 137–5 DBMS\_SQLDIAG Constants - Report Level (possible values) Constants**

Constant	Type	Value	Description
LEVEL_ALL	VARCHAR2(3)	'ALL'	Complete report including annotations about statements skipped over
LEVEL_BASIC	VARCHAR2(5)	'BASIC'	Shows information about every statement analyzed, including recommendations not implemented
LEVEL_TYPICAL	VARCHAR2(7)	'TYPICAL'	Simple report shows only information about the actions taken by the advisor.

**Table 137–6 DBMS\_SQLDIAG Constants - Report Section (possible values) Constants**

Constant	Type	Value	Description
SECTION_ALL	VARCHAR2(3)	'ALL'	All statements
SECTION_ERRORS	VARCHAR2(6)	'ERRORS'	Statements with errors
SECTION_FINDINGS	VARCHAR2(8)	'FINDINGS'	Tuning findings
SECTION_INFORMATION	VARCHAR2(11)	'INFORMATION'	General information
SECTION_PLANS	VARCHAR2(5)	'PLANS'	Explain plans
SECTION_SUMMARY	VARCHAR2(7)	'SUMMARY'	Summary information

**Table 137–7 DBMS\_SQLDIAG Constants - Problem Type Constants**

Constant	Type	Value	Description
PROBLEM_TYPE_PERFORMANCE	NUMBER	1	User suspects this is a performance problem

**Table 137-7 (Cont.) DBMS\_SQLDIAG Constants - Problem Type Constants**

Constant	Type	Value	Description
PROBLEM_TYPE_WRONG_RESULTS	NUMBER	2	User suspects the query is giving inconsistent results
PROBLEM_TYPE_COMPILATION_ERROR	NUMBER	3	User sees a crash in compilation
PROBLEM_TYPE_EXECUTION_ERROR	NUMBER	4	User sees a crash in execution
PROBLEM_TYPE_ALT_PLAN_GEN	NUMBER	5	User to explore all alternative plans

**Table 137-8 DBMS\_SQLDIAG Constants - Findings Filter Constants**

Constant	Type	Value	Description
SQLDIAG_FINDINGS_ALL	NUMBER	1	Show all possible findings
SQLDIAG_FINDINGS_VALIDATION	NUMBER	2	Show status of validation rules over structures
SQLDIAG_FINDINGS_FEATURES	NUMBER	3	Show only features used by the query
SQLDIAG_FINDINGS_FILTER_PLANS	NUMBER	4	Show the alternative plans generated by the advisor
SQLDIAG_FINDINGS_CR_DIFF	NUMBER	5	Show difference between two plans
SQLDIAG_FINDINGS_MASK_VARIANT	NUMBER	6	Mask info for testing
SQLDIAG_FINDINGS_OBJ_FEATURES	NUMBER	7	Show features usage history
SQLDIAG_FINDINGS_BASIC_INFO	NUMBER	8	Show the alternative plans generated by the advisor

## Examples

### Patch Pack / Unpack

Patches can be exported out of one system and imported into another by means of a staging table, provided by subprograms in this package. Like with SQL diagnosis sets, the operation of inserting into the staging table is called a "pack", and the operation of creating patches from staging table data is termed the "unpack".

DBAs should perform a pack/unpack as follows:

1. Create a staging table owned by user 'SH' through a call to `CREATE_STGTAB_SQLPATCH`:

```
EXEC DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH(
    table_name      => 'STAGING_TABLE',
    schema_name     => 'SH');
```

2. Call `PACK_STGTAB_SQLPATCH` one or more times to write SQL patch data into the staging table. In this case, copy data for all SQL patches in the `DEFAULT` category into a staging table owned by the current schema owner:

```
EXEC DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH(
    staging_table_name => 'STAGING_TABLE');
```

3. In this case, only a single SQL patch `SP_FIND_EMPLOYEE` is copied into a staging table owned by the current schema owner:

```
EXEC DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH(
    patch_name       => 'SP_FIND_EMPLOYEE',
    staging_table_name => 'STAGING_TABLE');
```

The staging table can then be moved to another system using either `datapump`, `import/export` commands or through a `databaselink`.

4. Call `UNPACK_STGTAB_SQLPATCH` to create SQL patches on the new system from the patch data in the staging table. In this case, change the name in the data for the `SP_FIND_EMPLOYEE` patch stored in the staging table to `'SP_FIND_EMP_PROD'`:

```
exec dbms_sqldiag.remap_stgtab_sqlpatch(
    old_patch_name   => 'SP_FIND_EMPLOYEE',
    new_patch_name   => 'SP_FIND_EMP_PROD');
```

## Summary of DBMS\_SQLDIAG Subprograms

**Table 137-9 DBMS\_SQLDIAG Package Subprograms**

Subprogram	Description
<a href="#">ACCEPT_SQL_PATCH Function &amp; Procedure</a> on page 137-11	Accepts a recommended SQL patch as recommended by the specified SQL diagnosis task
<a href="#">ALTER_SQL_PATCH Procedure</a> on page 137-13	Alters specific attributes of an existing SQL patch object
<a href="#">CANCEL_DIAGNOSIS_TASK Procedure</a> on page 137-14	Cancel a diagnostic task
<a href="#">CREATE_DIAGNOSIS_TASK Functions</a> on page 137-15	Creates a diagnostic task in order to diagnose a single SQL statement
<a href="#">CREATE_STGTAB_SQLPATCH Procedure</a> on page 137-17	Creates the staging table used for transporting SQL patches from one system to another
<a href="#">DROP_DIAGNOSIS_TASK Procedure</a> on page 137-18	Drops a diagnostic task
<a href="#">DROP_SQL_PATCH Procedure</a> on page 137-19	Drops the named SQL patch from the database
<a href="#">EXECUTE_DIAGNOSIS_TASK Procedure</a> on page 137-20	Executes a diagnostic task
<a href="#">EXPLAIN_SQL_TESTCASE Function</a> on page 137-21	Explains a SQL test case
<a href="#">EXPORT_SQL_TESTCASE Procedures</a> on page 137-22	Exports a SQL test case to a directory
<a href="#">EXPORT_SQL_TESTCASE_DIR_BY_INC Function</a> on page 137-25	Generates a SQL Test Case corresponding to the incident ID passed as an argument.
<a href="#">EXPORT_SQL_TESTCASE_DIR_BY_TXT Function</a> on page 137-26	Generates a SQL Test Case corresponding to the SQL passed as an argument
<a href="#">GET_FIX_CONTROL Function</a> on page 137-27	Returns the value of fix control for a given bug number
<a href="#">GET_SQL Function</a> on page 137-28	Imports a SQL test case
<a href="#">IMPORT_SQL_TESTCASE Procedures</a> on page 137-29	Imports a SQL test case into a schema
<a href="#">INCIDENTID_2_SQL Procedure</a> on page 137-31	Initializes a <code>sql_setrow</code> from an incident ID
<a href="#">INTERRUPT_DIAGNOSIS_TASK Procedure</a> on page 137-32	Interrupts a diagnostic task
<a href="#">LOAD_SQLSET_FROM_TCB Function</a> on page 137-33	Loads a <code>SQLSET</code> from Test Case Builder (TCB) file
<a href="#">PACK_STGTAB_SQLPATCH Procedure</a> on page 137-34	SQL patches into the staging table created by the <a href="#">CREATE_STGTAB_SQLPATCH Procedure</a>

**Table 137-9 (Cont.) DBMS\_SQLDIAG Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">REPORT_DIAGNOSIS_TASK</a> Function on page 137-35	Reports on a diagnostic task
<a href="#">RESET_DIAGNOSIS_TASK</a> Procedure on page 137-36	Resets a diagnostic task
<a href="#">RESUME_DIAGNOSIS_TASK</a> Procedure on page 137-37	Resumes a diagnostic task
<a href="#">SET_DIAGNOSIS_TASK_PARAMETER</a> Procedure on page 137-38	Sets a diagnosis task parameter
<a href="#">UNPACK_STGTAB_SQLPATCH</a> Procedure on page 137-39	Unpacks from the staging table populated by a call to the <a href="#">PACK_STGTAB_SQLPATCH</a> Procedure, using the patch data stored in the staging table to create patches on this system

---



## ACCEPT\_SQL\_PATCH Function & Procedure

This procedure accepts a recommended SQL patch as recommended by the specified SQL diagnosis task.

### Syntax

```
DBMS_SQLDIAG.ACCEPT_SQL_PATCH (
  task_name      IN  VARCHAR2,
  object_id      IN  NUMBER := NULL,
  name           IN  VARCHAR2 := NULL,
  description    IN  VARCHAR2 := NULL,
  category       IN  VARCHAR2 := NULL,
  task_owner    IN  VARCHAR2 := NULL,
  replace       IN  BOOLEAN := FALSE,
  force_match    IN  BOOLEAN := FALSE)
RETURN VARCHAR2;

DBMS_SQLDIAG.ACCEPT_SQL_PATCH (
  task_name      IN  VARCHAR2,
  object_id      IN  NUMBER := NULL,
  name           IN  VARCHAR2 := NULL,
  description    IN  VARCHAR2 := NULL,
  category       IN  VARCHAR2 := NULL,
  task_owner    IN  VARCHAR2 := NULL,
  replace       IN  BOOLEAN := FALSE,
  force_match    IN  BOOLEAN := FALSE);
```

### Parameters

**Table 137–10** ACCEPT\_SQL\_PATCH Function & Procedure Parameters

Parameter	Description
taskname	Name of the SQL diagnosis task
object_id	Identifier of the advisor framework object representing the SQL statement associated to the diagnosis task
name	Name of the patch. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system will generate a unique name for the SQL patch.
description	User specified string describing the purpose of this SQL patch. Maximum size of description is 500.
category	Category name which must match the value of the SQLDIAGNOSE_CATEGORY parameter in a session for the session to use this patch. It defaults to the value DEFAULT. This is also the default of the SQLDIAGNOSE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name create a unique key for a patch. An accept will fail if this combination is duplicated.
task_owner	Owner of the diagnosis task. This is an optional parameter that has to be specified to accept a SQL Patch associated to a diagnosis task owned by another user. The current user is the default value.

**Table 137–10 (Cont.) ACCEPT\_SQL\_PATCH Function & Procedure Parameters**

Parameter	Description
<code>replace</code>	If the patch already exists, it will be replaced if this argument is <code>TRUE</code> . It is an error to pass a name that is already being used for another signature/category pair, even with <code>replace</code> set to <code>TRUE</code> .
<code>force_match</code>	If <code>TRUE</code> this causes SQL Patches to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the <code>FORCE</code> option of the <code>CURSOR_SHARING</code> parameter. If <code>FALSE</code> , literals are not transformed. This is analogous to the matching algorithm used by the <code>EXACT</code> option of the <code>CURSOR_SHARING</code> parameter.

**Return Values**

Name of the SQL patch

**Usage Notes**

Requires `CREATE ANY SQL PROFILE` privilege

## ALTER\_SQL\_PATCH Procedure

This procedure alters specific attributes of an existing SQL patch object.

### Syntax

```
DBMS_SQLDIAG.ALTER_SQL_PATCH (
  name           IN VARCHAR2,
  attribute_name IN VARCHAR2,
  value          IN VARCHAR2);
```

### Parameters

**Table 137–11 ALTER\_SQL\_PATCH Procedure Parameters**

Parameter	Description
name	Name of SQL patch to alter.
attribute_name	Name of SQL patch to alter. Possible values: <ul style="list-style-type: none"> <li>■ STATUS -&gt; can be set to ENABLED or DISABLED</li> <li>■ NAME -&gt; can be reset to a valid name (must be a valid Oracle identifier and must be unique).</li> <li>■ DESCRIPTION -&gt; can be set to any string of size no more than 500</li> <li>■ CATEGORY -&gt; can be reset to a valid category name (must be valid Oracle identifier and must be unique when combined with normalized SQL text)</li> </ul> This parameter is mandatory and is case sensitive.
value	New value of the attribute. See attribute_name for valid attribute values. This parameter is mandatory.

### Usage Notes

Requires ALTER ANY SQL PATCH privilege

## CANCEL\_DIAGNOSIS\_TASK Procedure

This procedure cancels a diagnostic task.

### Syntax

```
DBMS_SQLDIAG.CANCEL_DIAGNOSIS_TASK (  
    taskname          IN  VARCHAR2);
```

### Parameters

**Table 137–12** CANCEL\_DIAGNOSIS\_TASK Procedure Parameters

Parameter	Description
taskname	Name of task

## CREATE\_DIAGNOSIS\_TASK Functions

This function creates a diagnostic task in order to diagnose a single SQL statement. It returns a SQL diagnosis task unique name

### Syntax

Prepares the diagnosis of a single statement given its text:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
    sql_text          IN   CLOB,
    bind_list         IN   sql_binds := NULL,
    user_name         IN   VARCHAR2 := NULL,
    scope             IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit        IN   NUMBER    := TIME_LIMIT_DEFAULT,
    task_name         IN   VARCHAR2 := NULL,
    description       IN   VARCHAR2 := NULL,
    problem_type      IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE)
RETURN VARCHAR2;
```

Prepares the diagnosis of a single statement from the Cursor Cache given its identifier:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
    sql_id            IN   VARCHAR2,
    plan_hash_value   IN   NUMBER    := NULL,
    scope             IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit        IN   NUMBER    := TIME_LIMIT_DEFAULT,
    task_name         IN   VARCHAR2 := NULL,
    description       IN   VARCHAR2 := NULL,
    problem_type      IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE)
RETURN VARCHAR2;
```

Prepares the diagnosis of a Sqlset:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
    sqlset_name       IN   VARCHAR2,
    basic_filter      IN   VARCHAR2 := NULL,
    object_filter     IN   VARCHAR2 := NULL,
    rank1             IN   VARCHAR2 := NULL,
    rank2             IN   VARCHAR2 := NULL,
    rank3             IN   VARCHAR2 := NULL,
    result_percentage IN   NUMBER    := NULL,
    result_limit      IN   NUMBER    := NULL,
    scope             IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit        IN   NUMBER    := TIME_LIMIT_DEFAULT,
    task_name         IN   VARCHAR2 := NULL,
    description       IN   VARCHAR2 := NULL,
    plan_filter       IN   VARCHAR2 := 'MAX_ELAPSED_TIME',
    sqlset_owner      IN   VARCHAR2 := NULL,
    problem_type      IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE)
RETURN VARCHAR2;
```

### Parameters

**Table 137–13 CREATE\_DIAGNOSIS\_TASK Function Parameters**

Parameter	Description
sql_text	Text of a SQL statement
bind_list	Set of bind values

**Table 137-13 (Cont.) CREATE\_DIAGNOSIS\_TASK Function Parameters**

<b>Parameter</b>	<b>Description</b>
user_name	Username for who the statement/sqlset will be diagnosed
scope	Diagnosis scope (limited/comprehensive)
time_limit	Maximum duration in seconds for the diagnosis session
task_name	Optional diagnosis task name
description	Maximum of 256 SQL diagnosis session description
problem_type	Determines the goal of the task. Possible values are: <ul style="list-style-type: none"> <li>■ PROBLEM_TYPE_WRONG_RESULTS</li> <li>■ PROBLEM_TYPE_COMPILATION_ERROR</li> <li>■ PROBLEM_TYPE_EXECUTION_ERROR</li> </ul>
sql_id	Identifier of the statement
plan_hash_value	Hash value of the SQL execution plan
sqlset_name	Sqlset name
basic_filter	SQL predicate to filter the SQL from the SQL tuning set (STS)
object_filter	Object filter
rank(i)	Order-by clause on the selected SQL
result_percentage	Percentage on the sum of a ranking measure
result_limit	Top L(imit) SQL from (filtered/ranked) SQL
plan_filter	Plan filter. It is applicable in case there are multiple plans (plan_hash_value). This filter allows selecting one plan (plan_hash_value) only. Possible values are: <ul style="list-style-type: none"> <li>■ LAST_GENERATED: plan with most recent timestamp</li> <li>■ FIRST_GENERATED: opposite to LAST_GENERATED</li> <li>■ LAST_LOADED: plan with most recent first_load_time stat info</li> <li>■ FIRST_LOADED: opposite to LAST_LOADED</li> <li>■ MAX_ELAPSED_TIME: plan with maximum elapsed time</li> <li>■ MAX_BUFFER_GETS: plan with maximum buffer gets</li> <li>■ MAX_DISK_READS: plan with maximum disk reads</li> <li>■ MAX_DIRECT_WRITES: plan with maximum direct writes</li> <li>■ MAX_OPTIMIZER_COST: plan with maximum optimum cost</li> </ul>
sqlset_owner	Owner of the sqlset, or null for current schema owner

## CREATE\_STGTAB\_SQLPATCH Procedure

This procedure creates the staging table used for transporting SQL patches from one system to another.

### Syntax

```
DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH (  
    table_name      IN  VARCHAR2 ,  
    schema_name     IN  VARCHAR2 := NULL ,  
    tablespace_name IN  VARCHAR2 := NULL ) ;
```

### Parameters

**Table 137-14** CREATE\_STGTAB\_SQLPATCH Procedure Parameters

Parameter	Description
table_name	(Mandatory) Name of the table to create (case-sensitive)
schema_name	Schema to create the table in, or NULL for current schema (case-sensitive)
tablespace_name	Tablespace to store the staging table within, or NULL for current user's default tablespace (case-sensitive)

## DROP\_DIAGNOSIS\_TASK Procedure

This procedure drops a diagnostic task.

### Syntax

```
DBMS_SQLDIAG.DROP_DIAGNOSIS_TASK (  
    taskname          IN  VARCHAR2);
```

### Parameters

**Table 137–15** *DROP\_DIAGNOSIS\_TASK Procedure Parameters*

Parameter	Description
taskname	Name of task



## DROP\_SQL\_PATCH Procedure

This procedure drops the named SQL patch from the database.

### Syntax

```
DBMS_SQLDIAG.DROP_SQL_PATCH (  
    name      IN  VARCHAR2,  
    ignore    IN  BOOLEAN := FALSE);
```

### Parameters

**Table 137–16** *DROP\_SQL\_PATCH Function & Procedure Parameters*

Parameter	Description
name	Name of patch to be dropped. The name is case sensitive.
ignore	Ignore errors due to object not existing.

### Usage Notes

Requires DROP ANY SQL PATCH privilege

## EXECUTE\_DIAGNOSIS\_TASK Procedure

This procedure executes a diagnostic task.

### Syntax

```
DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK (  
    taskname          IN  VARCHAR2);
```

### Parameters

**Table 137–17 EXECUTE\_DIAGNOSIS\_TASK Procedure Parameters**

Parameter	Description
taskname	Name of task

## EXPLAIN\_SQL\_TESTCASE Function

This procedure explains a SQL test case.

### Syntax

```
DBMS_SQLDIAG.EXPLAIN_SQL_TESTCASE (  
    sqlTestCase      IN  CLOB)  
RETURN CLOB;
```

### Parameters

**Table 137-18** *EXPLAIN\_SQL\_TESTCASE Function Parameters*

Parameter	Description
sqlTestCase	XML document describing the SQL test case

## EXPORT\_SQL\_TESTCASE Procedures

This procedure exports a SQL test case to a directory.

### Syntax

This variant has to be provided with the SQL information.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory          IN          VARCHAR2,
    sql_text           IN          CLOB,
    user_name          IN          VARCHAR2 := NULL,
    bind_list          IN          sql_binds := NULL,
    exportEnvironment IN          BOOLEAN := TRUE,
    exportMetadata     IN          BOOLEAN := TRUE,
    exportData         IN          BOOLEAN := FALSE,
    exportPkgbody      IN          BOOLEAN := FALSE,
    samplingPercent    IN          NUMBER := 100,
    ctrlOptions        IN          VARCHAR2 := NULL,
    timeLimit          IN          NUMBER := 0,
    testcase_name      IN          VARCHAR2 := NULL,
    testcase           IN OUT NOCOPY CLOB,
    preserveSchemaMapping IN      BOOLEAN := FALSE);
```

This variant extracts the SQL information from an incident file.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory          IN          VARCHAR2,
    incident_id        IN          VARCHAR2,
    exportEnvironment IN          BOOLEAN := TRUE,
    exportMetadata     IN          BOOLEAN := TRUE,
    exportData         IN          BOOLEAN := FALSE,
    exportPkgbody      IN          BOOLEAN := FALSE,
    samplingPercent    IN          NUMBER := 100,
    ctrlOptions        IN          VARCHAR2 := NULL,
    timeLimit          IN          NUMBER :=
        DBMS_SQLDIAG.TIME_LIMIT_DEFAULT,
    testcase_name      IN          VARCHAR2 := NULL,
    testcase           IN OUT NOCOPY CLOB,
    preserveSchemaMapping IN      BOOLEAN := FALSE);
```

This variant allow the SQL Test case to be generated from a cursor present in the cursor cache. Use V\$SQL to get the SQL identifier and the SQL hash value.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory          IN          VARCHAR2,
    sql_id             IN          VARCHAR2,
    plan_hash_value    IN          NUMBER := NULL,
    exportEnvironment IN          BOOLEAN := TRUE,
    exportMetadata     IN          BOOLEAN := TRUE,
    exportData         IN          BOOLEAN := TRUE,
    exportPkgbody      IN          BOOLEAN := FALSE,
    samplingPercent    IN          NUMBER := 100,
    ctrlOptions        IN          VARCHAR2 := NULL,
    timeLimit          IN          NUMBER :=
        DBMS_SQLDIAG.TIME_LIMIT_DEFAULT,
    testcase_name      IN          VARCHAR2 := NULL,
    testcase           IN OUT NOCOPY CLOB,
    preserveSchemaMapping IN      BOOLEAN := FALSE);
```

## Parameters

**Table 137–19 EXPORT\_SQL\_TESTCASE Procedure Parameters**

Parameter	Description
directory	Directory to store the various generated files
sql_text	Text of the SQL statement to export
incident_id	Incident ID containing the offending SQL
sql_id	Identifier of the statement in the cursor cache
username	Name of the user schema to use to parse the SQL, defaults to SYS
bind_list	List of bind values associated to the statement
exportEnvironment	TRUE if the compilation environment should be exported
exportMetadata	TRUE if the definition of the objects referenced in the SQL should be exported
exportData	TRUE if the data of the objects referenced in the SQL should be exported
exportPkgbody	TRUE if the body of the packages referenced in the SQL are exported
samplingPercent	If is TRUE, specify the sampling percentage to use to create the dump file
ctrlOptions	Opaque control parameters
timeLimit	How much time should we spend exporting the SQL test case
testcaseName	An optional name for the SQL test case. This is used to prefix all the generated scripts
testcase	Resulting testcase
preservesSchemaMapping	TRUE if the schema (or schemas) are not re-mapped from the original environment to the test environment

## Usage Notes

- A SQL test case generates a set of files needed to help reproduce a SQL failure on a different machine. It contains:
  - a dump file containing schemas objects and statistics (.dmp)
  - the explain plan for the statements (in advanced mode)
  - diagnostic information gathered on the offending statement
  - an import script to execute to reload the objects
  - a SQL script to replay system statistics of the source
  - a table of contents file describing the SQL test case
  - metadata.(xxxxxmain.xml)
- You should not run Test Case Builder (TCB) under user SYS. Instead, use another user who can be granted the DBA role.
- The default setting for TCB is that data is not exported. However, in some cases data is required, such as to diagnose an outcome with a result that is not optimal. To export data, call EXPORT\_SQL\_TESTCASE with exportData=>TRUE and the data will be imported by default, unless turned OFF by importData=>FALSE.

- TCB includes PL/SQL package spec by default, but not the PL/SQL package body. However, you may need to have the package body as well, for example, to invoke the PL/SQL functions, or because you have a Virtual Private Database (VPD) function defined in a package. To export a PL/SQL package body, call `EXPORT_SQL_TESTCASE` with `exportPkgbody=>TRUE`. To import a PL/SQL package body, call [IMPORT\\_SQL\\_TESTCASE Procedures](#) with `importPkgbody=>TRUE`.

## EXPORT\_SQL\_TESTCASE\_DIR\_BY\_INC Function

This function generates a SQL Test Case corresponding to the incident ID passed as an argument. It creates a set of scripts and dump file in the directory passed as an argument.

### Syntax

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_INC (
  incident_id      IN  NUMBER,
  directory        IN  VARCHAR2,
  exportEnvironment IN  VARCHAR2 := 'TRUE',
  exportMetadata  IN  VARCHAR2 := 'TRUE',
  exportData      IN  VARCHAR2 := 'FALSE',
  samplingPercent IN  VARCHAR2 := '100',
  ctrlOptions     IN  VARCHAR2 := NULL)
RETURN BOOLEAN;
```

### Parameters

**Table 137–20 EXPORT\_SQL\_TESTCASE\_DIR\_BY\_INC Function Parameters**

Parameter	Description
incident_id	Incident ID containing the offending SQL. For more information about Incidents, see <i>Oracle Database Performance Tuning Guide</i> .
directory	Directory to store the various generated files
exportEnvironment	TRUE if the compilation environment should be exported
exportMetadata	TRUE if the definition of the objects referenced in the SQL should be exported
exportData	TRUE if the data of the objects referenced in the SQL should be exported
samplingPercent	If is TRUE, specify the sampling percentage to use to create the dump file
ctrlOptions	Opaque control parameters

## EXPORT\_SQL\_TESTCASE\_DIR\_BY\_TXT Function

This function generates a SQL Test Case corresponding to the SQL passed as an argument. It creates a set of scripts and dump files in the directory passed as an argument.

### Syntax

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_TXT (
    incident_id      IN    NUMBER,
    directory        IN    VARCHAR2,
    sql_text         IN    CLOB,
    user_name        IN    VARCHAR2 := 'SYS',
    exportEnvironment IN  VARCHAR2 := 'TRUE',
    exportMetadata   IN  VARCHAR2 := 'TRUE',
    exportData       IN  VARCHAR2 := 'FALSE',
    samplingPercent  IN  VARCHAR2 := '100',
    ctrlOptions      IN  VARCHAR2 := NULL)
RETURN BOOLEAN;
```

### Parameters

**Table 137–21 EXPORT\_SQL\_TESTCASE\_DIR\_BY\_TXT Function Parameters**

Parameter	Description
incident_id	Incident ID containing the offending SQL
directory	Directory to store the various generated files
sql_text	Text of the SQL statement to explain
username	Name of the user schema to use to parse the SQL, defaults to SYS
exportEnvironment	TRUE if the compilation environment should be exported
exportMetadata	TRUE if the definition of the objects referenced in the SQL should be exported
exportData	TRUE if the data of the objects referenced in the SQL should be exported
samplingPercent	If is TRUE, specify the sampling percentage to use to create the dump file
ctrlOptions	Opaque control parameters



## GET\_FIX\_CONTROL Function

This function returns the value of fix control for a given bug number.

### Syntax

```
DBMS_SQLDIAG.GET_FIX_CONTROL (  
    bug_number IN NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 137-22** *GET\_FIX\_CONTROL Function Parameters*

Parameter	Description
bug_number	Bug number

## GET\_SQL Function

This function loads a `sql_setrow` from the trace file associated to an the given incident ID.

### Syntax

```
DBMS_SQLDIAG.GET_SQL (  
    incident_id IN    VARCHAR2)  
    RETURN SQLSET_ROW;
```

### Parameters

**Table 137–23** *GET\_SQL Function Parameters*

Parameter	Description
<code>incident_id</code>	Identifier of the incident

## IMPORT\_SQL\_TESTCASE Procedures

This procedure imports a SQL test case into a schema.

### Syntax

This variant requires a source directory and SQL Testcase metadata object (in XML format).

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (
  directory          IN  VARCHAR2,
  sqlTestCase        IN  CLOB,
  importEnvironment  IN  BOOLEAN := TRUE,
  importMetadata     IN  BOOLEAN := TRUE,
  importData         IN  BOOLEAN := TRUE,
  importPkgbody      IN  BOOLEAN := FALSE,
  importDiagnosis    IN  BOOLEAN := TRUE,
  ignoreStorage      IN  BOOLEAN := TRUE,
  ctrlOptions        IN  VARCHAR2 := NULL,
  preserveSchemaMapping IN BOOLEAN := FALSE);
```

This variant requires a source directory name of SQL Testcase metadata file.

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (
  directory          IN  VARCHAR2,
  filename           IN  VARCHAR2,
  importEnvironment  IN  BOOLEAN := TRUE,
  importMetadata     IN  BOOLEAN := TRUE,
  importData         IN  BOOLEAN := TRUE,
  importPkgbody      IN  BOOLEAN := FALSE,
  importDiagnosis    IN  BOOLEAN := TRUE,
  ignoreStorage      IN  BOOLEAN := TRUE,
  ctrlOptions        IN  VARCHAR2 := NULL,
  preserveSchemaMapping IN BOOLEAN := FALSE);
```

### Parameters

**Table 137–24** *IMPORT\_SQL\_TESTCASE Procedure Parameters*

Parameter	Description
directory	Directory containing test case files
filename	Name of a file containing an XML document describing the SQL test case
importEnvironment	TRUE if the compilation environment should be imported
importMetadata	TRUE if the definition of the objects referenced in the SQL should be imported
importData	TRUE if the data of the objects referenced in the SQL should be imported
importPkgbody	TRUE if the body of the packages referenced in the SQL are imported
importDiagnosis	TRUE if the diagnostic information associated to the task should be imported
ignoreStorage	TRUE if the storage attributes should be ignored
ctrlOptions	Opaque control parameters

**Table 137-24 (Cont.) IMPORT\_SQL\_TESTCASE Procedure Parameters**

Parameter	Description
<code>preservesSchemaMapping</code>	TRUE if the schema (or schemas) are not re-mapped from the original environment to the test environment

## Usage Notes

- A SQL test case generates a set of files needed to help reproduce a SQL failure on a different machine. It contains:
  - a dump file containing schemas objects and statistics (.dmp)
  - the explain plan for the statements (in advanced mode)
  - diagnostic information gathered on the offending statement
  - an import script to execute to reload the objects
  - a SQL script to replay system statistics of the source
  - a table of contents file describing the SQL test case
  - metadata. (xxxxmain.xml)
- You should not run Test Case Builder (TCB) under user SYS. Instead, use another user who can be granted the SYSDBA privilege
- The default setting for TCB is that data is not exported. However, in some cases data is required, such as to diagnose an outcome with a result that is not optimal. To export data, call [EXPORT\\_SQL\\_TESTCASE Procedures](#) with `exportData=>TRUE` and the data will be imported by default, unless turned OFF by `importData=>FALSE`.
- TCB includes PL/SQL package spec by default, but not the PL/SQL package body. However, you may need to have the package body as well, for example, to invoke the PL/SQL functions, or because you have a Virtual Private Database (VPD) function defined in a package. To export a PL/SQL package body, call [EXPORT\\_SQL\\_TESTCASE Procedures](#) with `exportPkgbody=>TRUE`. To import a PL/SQL package body, call [IMPORT\\_SQL\\_TESTCASE Procedures](#) with `importPkgbody=>TRUE`.

## INCIDENTID\_2\_SQL Procedure

This procedure initializes a `sql_setrow` from an incident ID.

### Syntax

```
DBMS_SQLDIAG.INCIDENTID_2_SQL (
  incident_id  IN   VARCHAR2,
  sql_stmt    OUT  SQLSET_ROW,
  problem_type OUT  NUMBER,
  err_code    OUT  BINARY_INTEGER,
  err_mesg    OUT  VARCHAR2);
```

### Parameters

**Table 137–25** *INCIDENTID\_2\_SQL Procedure Parameters*

Parameter	Description
<code>incident_id</code>	Identifier of the incident
<code>sql_stmt</code>	Resulting SQL
<code>problem_type</code>	Tentative type of SQL problem (currently among <code>PROBLEM_TYPE_COMPILATION_ERROR</code> and <code>PROBLEM_TYPE_EXECUTION_ERROR</code> )
<code>err_code</code>	Error code if any otherwise it is set to <code>NULL</code>
<code>err_msg</code>	Error message if any otherwise it is set to <code>NULL</code>

## INTERRUPT\_DIAGNOSIS\_TASK Procedure

This procedure interrupts a diagnostic task.

### Syntax

```
DBMS_SQLDIAG.INTERRUPT_DIAGNOSIS_TASK (  
    taskname          IN  VARCHAR2);
```

### Parameters

**Table 137–26** *INTERRUPT\_DIAGNOSIS\_TASK Procedure Parameters*

Parameter	Description
taskname	Name of task

## LOAD\_SQLSET\_FROM\_TCB Function

This function loads a SQLSET from a Test Case Builder file.

### Syntax

```
DBMS_SQLDIAG.LOAD_SQLSET_FROM_TCB (  
    directory      IN      VARCHAR2,  
    filename       IN      VARCHAR2,  
    sqlset_name    IN      VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 137-27** *LOAD\_SQLSET\_FROM\_TCB Function Parameters*

Parameter	Description
directory	Name of directory
filename	Name of file
sqlset_name	Name of SQLSET

## PACK\_STGTAB\_SQLPATCH Procedure

This procedure packs SQL patches into the staging table created by a call to the [CREATE\\_STGTAB\\_SQLPATCH Procedure](#).

### Syntax

```
DBMS_SQLDIAG.UPPACK_STGTAB_SQLPATCH (
  patch_name          IN  VARCHAR2 := '%',
  patch_category      IN  VARCHAR2 := 'DEFAULT',
  staging_table_name  IN  VARCHAR2,
  staging_schema_owner IN  VARCHAR2 := NULL);
```

### Parameters

**Table 137–28 UPPACK\_STGTAB\_SQLPATCH Procedure Parameters**

Parameter	Description
patch_name	Name of patch to pack (% wildcards acceptable, case-sensitive)
patch_category	Category to which to pack patches (% wildcards acceptable, case-insensitive)
staging_table_name	(Mandatory) Name of the table to use (case-sensitive)
staging_schema_owner	Schema where the table resides, or NULL for current schema (case-sensitive)

### Usage Notes

- Requires: ADMINISTER SQL PLAN MANAGEMENT OBJECT privilege and INSERT privilege on the staging table
- By default, we move all SQL patches in category DEFAULT. See the [Examples](#) on page 137-8 for details. Note that the subprogram issues a COMMIT after packing each SQL patch, so if an error is raised in mid-execution, some patches may be in the staging table.



## REPORT\_DIAGNOSIS\_TASK Function

This function reports on a diagnostic task. It returns a CLOB containing the desired report.

### Syntax

```
DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK (
  taskname          IN  VARCHAR2,
  type              IN  VARCHAR2 := TYPE_TEXT,
  level             IN  VARCHAR2 := LEVEL_TYPICAL,
  section          IN  VARCHAR2 := SECTION_ALL,
  object_id        IN  NUMBER   := NULL,
  result_limit     IN  NUMBER   := NULL,
  owner_name       IN  VARCHAR2 := NULL)
RETURN CLOB;
```

### Parameters

**Table 137–29** *REPORT\_DIAGNOSIS\_TASK Function Parameters*

Parameter	Description
taskname	Name of task to report
type	Type of the report. Possible values are: TEXT, HTML, XML (see <a href="#">Table 137–4, "DBMS_SQLDIAG Constants - Report Type (possible values) Constants"</a> ).
level	Format of the recommendations. Possible values are TYPICAL, BASIC, ALL ( <a href="#">Table 137–5, "DBMS_SQLDIAG Constants - Report Level (possible values) Constants"</a> ).
section	Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLAN, INFORMATION, ERROR, ALL ( <a href="#">Table 137–6, "DBMS_SQLDIAG Constants - Report Section (possible values) Constants"</a> ).
object_id	Identifier of the advisor framework object that represents a given statement in a SQL Tuning Set (STS).
result_limit	Number of statements in a STS for which the report is generated
owner_name	Name of the task execution to use. If NULL, the report will be generated for the last task execution.

## RESET\_DIAGNOSIS\_TASK Procedure

This procedure resets a diagnostic task.

### Syntax

```
DBMS_SQLDIAG.RESET_DIAGNOSIS_TASK (  
    taskname          IN  VARCHAR2);
```

### Parameters

**Table 137–30** RESET\_DIAGNOSIS\_TASK Procedure Parameters

Parameter	Description
taskname	Name of task

## RESUME\_DIAGNOSIS\_TASK Procedure

This procedure resumes a diagnostic path.

### Syntax

```
DBMS_SQLDIAG.RESUME_DIAGNOSIS_TASK (  
    taskname          IN  VARCHAR2);
```

### Parameters

**Table 137-31 RESUME\_DIAGNOSIS\_TASK Procedure Parameters**

Parameter	Description
taskname	Name of task

## SET\_DIAGNOSIS\_TASK\_PARAMETER Procedure

This procedure is called to update the value of a SQL diagnosis parameter of type VARCHAR2. The task must be set to its initial state before calling this procedure. The diagnosis parameters that can be set by this procedure are:

- MODE: diag scope (comprehensive, limited)
- \_SQLDIAG\_FINDING\_MODE: findings in the report (see "[DBMS\\_SQLDIAG Constants - Findings Filter Constants](#)" on page 137-7 for possible values)

### Syntax

```
DBMS_SQLDIAG.SET_DIAGNOSIS_TASK_PARAMETER (  
    taskname          IN  VARCHAR2,  
    parameter         IN  VARCHAR2,  
    value             IN  NUMBER);
```

### Parameters

**Table 137-32 SET\_DIAGNOSIS\_TASK\_PARAMETER Procedure Parameters**

Parameter	Description
taskname	Identifier of the task to execute
parameter	Name of the parameter to set
value	New value of the specified parameter

## UNPACK\_STGTAB\_SQLPATCH Procedure

This procedure unpacks from the staging table populated by a call to the [PACK\\_STGTAB\\_SQLPATCH Procedure](#). It uses the patch data stored in the staging table to create patches on this system. Users can opt to replace existing patches with patch data when they exist already. In this case, note that it is only possible to replace patches referring to the same statement if the names are the same (see the [ACCEPT\\_SQL\\_PATCH Function & Procedure](#)).

### Syntax

```
DBMS_SQLDIAG.UNPACK_STGTAB_SQLPATCH (
  patch_name          IN  VARCHAR2 := '%',
  patch_category     IN  VARCHAR2 := '%',
  replace            IN  BOOLEAN,
  staging_table_name IN  VARCHAR2,
  staging_schema_owner IN VARCHAR2 := NULL);
```

### Parameters

**Table 137–33 UPPACK\_STGTAB\_SQLPATCH Procedure Parameters**

Parameter	Description
patch_name	Name of patch to unpack (% wildcards acceptable, case-sensitive)
patch_category	Category from which to unpack patches (% wildcards acceptable, case-insensitive)
replace	Replace patches if they already exist. Note that patches cannot be replaced if there is one in the staging table with the same name as an active patch on different SQL. The subprogram raises an error if there an attempt to create a patch that already exists.
staging_table_name	(Mandatory) Name of the table to use (case-sensitive)
staging_schema_owner	Schema where the table resides, or NULL for current schema (case-sensitive)

### Usage Notes

- Requires: ADMINISTER SQL MANAGEMENT OBJECT privilege and SELECT privilege on the staging table
- By default, all SQL patches in the staging table are moved. The function commits after successfully loading each patch. If it fails in creating an individual patch, it raises an error and does not proceed to those remaining in the staging table.



The DBMS\_SQLPA package provides the interface to implement the SQL Performance Analyzer.

The chapter contains the following topics:

- [Using DBMS\\_SQLPA](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_SQLPA Subprograms](#)

## Using DBMS\_SQLPA

- [Overview](#)
- [Security Model](#)



## Overview

The DBMS\_SQLPA package provides a capacity to help users predict the impact of system environment changes on the performance of a SQL workload. The interface lets users build and then compare two different versions of the workload performance, analyze the differences between the two versions, and unmask the SQL statements that might be impacted by the changes.

The package provides a task-oriented interface to implement the SQL Performance Analyzer. For example

1. You use the [CREATE\\_ANALYSIS\\_TASK Functions](#) to create an analysis task for a single statement or a group of SQL statements.s
2. The [EXECUTE\\_ANALYSIS\\_TASK Function & Procedure](#) executes a previously created analysis task.
3. The [REPORT\\_ANALYSIS\\_TASK Function](#) displays the results of an analysis task.

## Security Model

This package is available to `PUBLIC` and performs its own security checking. All analysis task interfaces (`XXX_ANALYSIS_TASK`) require privilege `ADVISOR`.

---

## Summary of DBMS\_SQLPA Subprograms

**Table 138–1 DBMS\_SQLPA Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CANCEL_ANALYSIS_TASK Procedure</a> on page 138-6	Cancels the currently executing task analysis of one or more SQL statements
<a href="#">CREATE_ANALYSIS_TASK Functions</a> on page 138-7	Creates an advisor task to process and analyze one or more SQL statements
<a href="#">DROP_ANALYSIS_TASK Procedure</a> on page 138-10	Drops a SQL analysis task
<a href="#">EXECUTE_ANALYSIS_TASK Function &amp; Procedure</a> on page 138-11	Executes a previously created analysis task
<a href="#">INTERRUPT_ANALYSIS_TASK Procedure</a> on page 138-14	Interrupts the currently executing analysis task
<a href="#">REPORT_ANALYSIS_TASK Function</a> on page 138-15	Displays the results of an analysis task
<a href="#">RESET_ANALYSIS_TASK Procedure</a> on page 138-17	Resets the currently executing analysis task to its initial state
<a href="#">RESUME_ANALYSIS_TASK Procedure</a> on page 138-18	Resumes a previously interrupted analysis task that was created to process a SQL tuning set.
<a href="#">SET_ANALYSIS_TASK_PARAMETER Procedures</a> on page 138-19	Sets the SQL analysis task parameter value
<a href="#">SET_ANALYSIS_DEFAULT_PARAMETER Procedures</a> on page 138-21	Sets the SQL analysis task parameter default value

---

## CANCEL\_ANALYSIS\_TASK Procedure

This procedure cancels the currently executing analysis task. All intermediate result data is removed from the task.

### Syntax

```
DBMS_SQLPA.CANCEL_ANALYSIS_TASK(  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 138–2 CANCEL\_ANALYSIS\_TASK Procedure Parameters**

Parameter	Description
task_name	Name of the task to cancel

### Examples

Canceling a task when there is a need to stop it executing and it is not required to view any already-completed results:

```
EXEC DBMS_SQLPA.CANCEL_ANALYSIS_TASK(:my_task);
```

## CREATE\_ANALYSIS\_TASK Functions

These functions create an advisor task to process and analyze one or more SQL statements. You can use different forms of this function to:

- Create an analysis task for a single statement given its text.
- Create an analysis task for a single statement from the cursor cache given its identifier.
- Create an analysis task for a single statement from the workload repository given a range of snapshot identifiers.
- Create an analysis task for a SQL tuning set.

In all cases, the function creates an advisor task and sets its parameters.

### Syntax

SQL text format. This form of the function is called to prepare the analysis of a single statement given its text.

```
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
  sql_text          IN CLOB,
  bind_list         IN sql_binds := NULL,
  parsing_schema    IN VARCHAR2  := NULL,
  task_name         IN VARCHAR2  := NULL,
  description       IN VARCHAR2  := NULL)
RETURN VARCHAR2;
```

SQL ID format. This form of the function is called to prepare the analysis of a single statement from the cursor cache given its identifier.

```
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
  sql_id           IN VARCHAR2,
  plan_hash_value  IN NUMBER     := NULL,
  task_name        IN VARCHAR2  := NULL,
  description       IN VARCHAR2  := NULL)
RETURN VARCHAR2;
```

Workload Repository format. This form of the function is called to prepare the analysis of a single statement from the workload repository given a range of snapshot identifiers.

```
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
  begin_snap       IN NUMBER,
  end_snap         IN NUMBER,
  sql_id           IN VARCHAR2,
  plan_hash_value  IN NUMBER     := NULL,
  task_name        IN VARCHAR2  := NULL,
  description       IN VARCHAR2  := NULL)
RETURN VARCHAR2;
```

SQLSET format. This form of the function is called to prepare the analysis of a SQL tuning set.

```
DBMS_SQLPA.CREATE_ANALYSIS_TASK(
  sqlset_name      IN VARCHAR2,
  basic_filter     IN VARCHAR2  := NULL,
  order_by         IN VARCHAR2  := NULL,
  top_sql          IN VARCHAR2  := NULL,
  task_name        IN VARCHAR2  := NULL,
```

```

description      IN VARCHAR2 := NULL
sqlset_owner     IN VARCHAR2 := NULL)
RETURN VARCHAR2;

```

## Parameters

**Table 138–3 CREATE\_ANALYSIS\_TASK Function Parameters**

Parameter	Description
sql_text	Text of a SQL statement
bind_list	A set of bind values
parsing_schema	Name of the schema where the statement can be compiled
task_name	Optional analysis task name
description	Description of the SQL analysis task to a maximum of 256 characters
sql_id	Identifier of a SQL statement
plan_hash_value	Hash value of the SQL execution plan
begin_snap	Begin snapshot identifier
end_snap	End snapshot identifier
sqlset_name	SQL tuning set name
basic_filter	SQL predicate to filter the SQL from the SQL tuning set
order_by	Order-by clause on the selected SQL
top_sql	Top N SQL after filtering and ranking
sqlset_owner	The owner of the SQL tuning set, or NULL for the current schema owner

## Return Values

A SQL analysis task name that is unique by user (two different users can give the same name to their advisor tasks).

## Examples

```

variable stmt_task VARCHAR2(64);
variable sts_task  VARCHAR2(64);

-- Sql text format
EXEC :stmt_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sql_text => 'select quantity_sold from sales s, times t where s.time_id =
t.time_id and s.time_id = TO_DATE(''24-NOV-00'')');

-- Sql id format (cursor cache)
EXEC :stmt_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    sql_id    => 'ay1m3ssvtrh24');

-- Workload repository format
exec :stmt_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
    begin_snap => 1,
    end_snap   => 2,
    sql_id     => 'ay1m3ssvtrh24');

-- Sql tuning set format (first we need to load an STS, then analyze it)

```

```
EXEC :sts_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK( -  
    sqlset_name    => 'my_workload', -  
    order_by       => 'BUFFER_GETS', -  
    description    => 'process workload ordered by buffer gets');
```

## DROP\_ANALYSIS\_TASK Procedure

This procedure drops a SQL analysis task. The task and all its result data are deleted.

### Syntax

```
DBMS_SQLPA.DROP_ANALYSIS_TASK(  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 138–4** *DROP\_ANALYSIS\_TASK Procedure Parameters*

Parameter	Description
task_name	The name of the analysis task to drop



## EXECUTE\_ANALYSIS\_TASK Function & Procedure

This function and procedure executes a previously created analysis task, the function version returning the new execution name.

### Syntax

```
DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
    task_name          IN VARCHAR2,
    execution_type     IN VARCHAR2          := 'test execute',
    execution_name     IN VARCHAR2          := NULL,
    execution_params   IN dbms_advisor.argList := NULL,
    execution_desc     IN VARCHAR2          := NULL)
RETURN VARCHAR2;
```

```
DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
    task_name          IN VARCHAR2,
    execution_type     IN VARCHAR2          := 'test execute',
    execution_name     IN VARCHAR2          := NULL,
    execution_params   IN dbms_advisor.argList := NULL,
    execution_desc     IN VARCHAR2          := NULL);
```

### Parameters

**Table 138–5 EXECUTE\_ANALYSIS\_TASK Function & Procedure Parameters**

Parameter	Description
task_name	Identifier of the task to execute
execution_type	Type of the action to perform by the function. If NULL it will default to the value of the <code>DEFAULT_EXECUTION_TYPE</code> parameter. Possible values are: <ul style="list-style-type: none"> <li>▪ <code>[TEST] EXECUTE</code> - test-execute every SQL statement and collect its execution plans and execution statistics. The resulting plans and statistics will be stored in the advisor framework. This is default.</li> <li>▪ <code>EXPLAIN PLAN</code> - generate explain plan for every statement in the SQL workload. This is similar to the <code>EXPLAIN PLAN</code> command. The resulting plans will be stored in the advisor framework in association with the task.</li> <li>▪ <code>COMPARE [PERFORMANCE]</code> - analyze and compare two versions of SQL performance data. The performance data is generated by test-executing or generating explain plan of the SQL statements. Use this option when two executions of type <code>EXPLAIN_PLAN</code> or <code>TEST_EXECUTE</code> already exist in the task</li> <li>▪ <code>CONVERT SQLSET</code> - used to read the statistics captured in a SQL Tuning Set and model them as a task execution. This can be used when you wish to avoid executing the SQL statements because valid data for the experiment already exists in the SQL Tuning Set.</li> </ul>
execution_name	A name to qualify and identify an execution. If not specified, it will be generated by the advisor and returned by function.
execution_params	List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They will override the values for the parameters stored in the task (set through the <a href="#">SET_ANALYSIS_DEFAULT_PARAMETER Procedures</a> ).

**Table 138–5 (Cont.) EXECUTE\_ANALYSIS\_TASK Function & Procedure Parameters**

Parameter	Description
execution_desc	A 256-length string describing the execution

## Usage Notes

SQL performance analyzer task can be executed multiples times without having to reset it. For example, when a task is created to perform a change impact analysis on a SQL workload, the created task has to be executed before making any change in the system environment to build a version of the workload that will be used as a reference for performance analysis. Once the change has been made, a second execution is required to build the post-change version of the workload. Finally, the task has to be executed a third time to let the advisor analyze and compare the performance of the workload in both versions.

## Examples

### 1. Create a task with a purpose of change impact analysis

```
EXEC :tname := DBMS_SQLPA.CREATE_ANALYSIS_TASK (
    sqlset_name => 'my_sts');
```

### 2. Make baseline or the before change execution

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
    task_name      => :tname,
    execution_type => 'compare performance',
    execution_name => 'before_change');
```

### 3. Make change

...

### 4. Make the after change version of the workload performance

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
    task_name      => :tname, -
    execution_type => 'test execute ',
    execution_name => 'after_change');
```

### 5. Compare the two versions of the workload

By default we always compare the results of the two last executions. The SQL Performance Analyzer uses the `elapsed_time` as a default metric for comparison. Here we are changing it to `buffer_gets` instead.

```
EXEC DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER (
    :tname, 'comparison_metric', 'buffer_gets');
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
    task_name      => :tname, -
    execution_type => 'compare performance', -
    execution_name => 'after_change');
```

Use the following call if you would like to explicitly specify the two executions to compare as well as the comparison metric to use.

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
    task_name      => :tname, -
    execution_type => 'compare performance',
    execution_params => dbms_advisor.arglist(
```

```
'execution_name1',  
'before_change',  
'execution_name2',  
'after_change',  
'comparion_metric',  
'buffer_gets'));
```

## INTERRUPT\_ANALYSIS\_TASK Procedure

This procedure interrupts the currently executing analysis task. All intermediate result data will not be removed from the task.

### Syntax

```
DBMS_SQLPA.INTERRUPT_ANALYSIS_TASK(  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 138–6** *INTERRUPT\_ANALYSIS\_TASK Procedure Parameters*

Parameter	Description
task_name	Identifier of the analysis task to interrupt

### Examples

```
EXEC DBMS_SQLPA.INTERRUPT_ANALYSIS_TASK(:my_task);
```

## REPORT\_ANALYSIS\_TASK Function

This procedure displays the results of an analysis task.

### Syntax

```
DBMS_SQLPA.REPORT_ANALYSIS_TASK(
  task_name      IN  VARCHAR2,
  type           IN  VARCHAR2  := 'TEXT',
  level         IN  VARCHAR2  := 'TYPICAL',
  section       IN  VARCHAR2  := 'SUMMARY',
  object_id     IN  NUMBER     := NULL,
  top_sql       IN  NUMBER     := 100,
  execution_name IN  VARCHAR2  := NULL,
  task_owner    IN  VARCHAR2  := NULL,
  order_by     IN  VARCHAR2  := NULL)
RETURN CLOB;
```

### Parameters

**Table 138–7** REPORT\_ANALYSIS\_TASK Function Parameters

Parameter	Description
task_name	Name of the task to report
type	Type of the report to produce. Possible values are TEXT (default), HTML, XML and ACTIVE (see Usage Notes).
level	Level of detail in the report: <ul style="list-style-type: none"> <li>▪ ALL - details of all SQL</li> <li>▪ BASIC - currently the same as typical</li> <li>▪ CHANGED - only SQL with changed performance</li> <li>▪ CHANGED_PLANS - only SQL with plan changes</li> <li>▪ ERRORS - SQL with errors only</li> <li>▪ IMPROVED - only improved SQL</li> <li>▪ REGRESSED - only regressed SQL</li> <li>▪ TIMEOUT - only SQL which timed-out during execution</li> <li>▪ TYPICAL (default) - show information about every statement analyzed, including changing and errors</li> <li>▪ UNCHANGED - only SQL with unchanged performance</li> <li>▪ UNCHANGED_PLANS - only SQL with unchanged plans</li> <li>▪ UNSUPPORTED - only SQL not supported by SPAs</li> </ul>
section	Optionally limit the report to a single section (ALL for all sections): <ul style="list-style-type: none"> <li>▪ SUMMARY (default) - workload summary only</li> <li>▪ ALL - summary and details on SQL</li> </ul>
object_id	Identifier of the advisor framework object that represents a given SQL in a tuning set (STS)
top_sql	Number of SQL statements in a STS for which the report is generated
execution_name	Name of the task execution to use. If NULL, the report will be generated for the last task execution.

**Table 138–7 (Cont.) REPORT\_ANALYSIS\_TASK Function Parameters**

Parameter	Description
task_owner	Owner of the relevant analysis task. Defaults to the current schema owner.
order_by	How to sort SQL statements in the report (summary and body). Possible values: <ul style="list-style-type: none"> <li>■ CHANGE_DIFF - sort SQL statements by change difference in SQL performance in terms of the comparison Metric</li> <li>■ NULL (default) - order SQL statement by impact on workload</li> <li>■ SQL_IMPACT - order SQL statement by change impact on SQL</li> <li>■ WORKLOAD_IMPACT - same as NULL</li> <li>■ METRIC_DELTA - same as CHANGE_DIFF</li> </ul>

## Return Values

A CLOB containing the desired report.

## Usage Notes

ACTIVE reports have a rich, interactive user interface similar to Enterprise Manager while not requiring any EM installation. The report file built is in HTML format so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.

## Examples

```
-- Get the whole report for the single statement case.
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(:stmt_task) from dual;

-- Show me the summary for the sts case.
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(:sts_task, 'TEXT', 'TYPICAL', 'SUMMARY')
FROM DUAL;

-- Show me the findings for the statement I'm interested in.
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(:sts_task, 'TEXT', 'TYPICAL', 'ALL', 5)
from dual;
```

## RESET\_ANALYSIS\_TASK Procedure

This procedure is called on an analysis task that is not currently executing to prepare it for re-execution. All intermediate result data will be deleted.

### Syntax

```
DBMS_SQLPA.RESET_ANALYSIS_TASK(  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 138–8** *RESET\_ANALYSIS\_TASK Procedure Parameters*

Parameter	Description
task_name	Identifier of the analysis task to reset

### Examples

```
-- reset and re-execute a task  
EXEC DBMS_SQLPA.RESET_ANALYSIS_TASK(:sts_task);  
  
-- re-execute the task  
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(:sts_task);
```

## RESUME\_ANALYSIS\_TASK Procedure

This procedure resumes a previously interrupted or FAILED (with a fatal error) task execution.

### Syntax

```
DBMS_SQLPA.RESUME_ANALYSIS_TASK(
  task_name          IN VARCHAR2,
  basic_filter       IN VARCHAR2 := NULL);
```

### Parameters

**Table 138–9 RESUME\_ANALYSIS\_TASK Procedure Parameters**

Parameter	Description
task_name	Identifier of the analysis task to resume
basic_filter	A SQL predicate to filter the SQL from the SQL tuning set. Note that this filter will be applied in conjunction with the basic filter (parameter <code>basic_filter</code> ) that was specified when calling the <a href="#">CREATE_ANALYSIS_TASK Functions</a> .

### Usage Notes

Resuming a single SQL analysis task (a task that was created to analyze a single SQL statement as compared to a SQL Tuning Set) is not supported.

### Examples

```
-- Interrupt the task
EXEC DBMS_SQLPA.INTERRUPT_ANALYSIS_TASK(:conc_task);

-- Once a task is interrupted, we can elect to reset it, resume it, or check
-- out its results and then decide. For this example we will just resume.

EXEC DBMS_SQLPA.RESUME_ANALYSIS_TASK(:conc_task);
```



## SET\_ANALYSIS\_TASK\_PARAMETER Procedures

This procedure sets the SQL analysis task parameter value.

### Syntax

This form of the procedure updates the value of a SQL analysis parameter of type VARCHAR2.

```
DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(
  task_name      IN VARCHAR2,
  parameter      IN VARCHAR2,
  value          IN VARCHAR2);
```

This form of the procedure updates the value of a SQL analysis parameter of type NUMBER.

```
DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(
  task_name      IN VARCHAR2,
  parameter      IN VARCHAR2,
  value          IN NUMBER);
```

### Parameters

**Table 138–10 SET\_ANALYSIS\_TASK\_PARAMETER Procedure Parameters**

Parameter	Description
task_name	Identifier of the task to execute
parameter	Name of the parameter to set. The possible analysis parameters that can be set by this procedure are: <ul style="list-style-type: none"> <li>▪ APPLY_CAPTURED_COMPILEENV: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO).</li> <li>▪ BASIC_FILTER: basic filter for SQL tuning set</li> <li>▪ CELL_SIMULATION_ENABLED: Set it to 'TRUE' to simulate Exadata Cell effect on SQL tuning set. For more details, see the helper script tcellsim.sql in the ADMIN directory.</li> <li>▪ COMPARISON_METRIC: specify an expression of execution statistics to use in performance comparison (Example: buffer_gets, cpu_time + buffer_gets * 10)</li> <li>▪ DATABASE_LINK: can be set to the global name of a PUBLIC database link. When it is set, SQL Performance Analyzer will use the database link for all TEST EXECUTE and EXPLAIN PLAN operations by sending the SQL statements to the remote database to be processed remotely. The analysis results will still be stored on the local database.</li> <li>▪ DAYS_TO_EXPIRE: number of days until the task is deleted</li> <li>▪ DEFAULT_EXECUTION_TYPE: the task will default to this type of execution when none is specified by the <a href="#">EXECUTE_ANALYSIS_TASK Function &amp; Procedure</a>.</li> </ul>

**Table 138–10 (Cont.) SET\_ANALYSIS\_TASK\_PARAMETER Procedure Parameters**

Parameter	Description
parameter (contd.)	<ul style="list-style-type: none"> <li data-bbox="667 260 1333 390">■ <code>DISABLE_MULTI_EXEC</code>: SQL statements are executed multiple times and runtime statistics are then averaged. Set this parameter to 'TRUE' to disable this capability. In this case, each SQL in the SQL tuning set is executed only once.</li> <li data-bbox="667 407 1333 485">■ <code>EXECUTION_DAYS_TO_EXPIRE</code>: number of days until the task's executions will be deleted (without deleting the task)</li> <li data-bbox="667 501 1333 657">■ <code>EXECUTE_FULLDML</code>: TRUE to execute DML statement fully, including acquiring row locks and modifying rows; FALSE (default) to execute only the query part of the DML without modifying data. When TRUE, SQL Performance Analyzer will issue a rollback following DML execution to prevent persistent changes from being made by the DML.</li> <li data-bbox="667 674 1333 722">■ <code>EXECUTION_NAME1</code>: name of the first task execution to analyze</li> <li data-bbox="667 739 1333 787">■ <code>EXECUTION_NAME2</code>: name of the second task execution to analyze</li> <li data-bbox="667 804 1333 829">■ <code>LOCAL_TIME_LIMIT</code>: per-statement time out (seconds)</li> <li data-bbox="667 846 1333 921">■ <code>METRIC_DELTA_THRESHOLD</code>: threshold of the difference between the SQL performance metric before and after the change. The default value is zero.</li> <li data-bbox="667 938 1333 987">■ <code>PLAN_FILTER</code>: plan filter for SQL tuning set (see <code>SELECT_SQLSET</code> for possible values)</li> <li data-bbox="667 1003 1333 1052">■ <code>RANK_MEASURE1</code>: first ranking measure for SQL tuning set</li> <li data-bbox="667 1068 1333 1117">■ <code>RANK_MEASURE2</code>: second possible ranking measure for SQL tuning set</li> <li data-bbox="667 1134 1333 1182">■ <code>RANK_MEASURE3</code>: third possible ranking measure for SQL tuning set</li> <li data-bbox="667 1199 1333 1247">■ <code>RESUME_FILTER</code>: a extra filter for SQL tuning sets besides <code>BASIC_FILTER</code></li> <li data-bbox="667 1264 1333 1339">■ <code>SQL_IMPACT_THRESHOLD</code>: threshold of a change impact on a SQL statement. Same as the previous parameter, but at the level of the SQL statement.</li> <li data-bbox="667 1356 1333 1404">■ <code>SQL_LIMIT</code>: maximum number of SQL statements to process</li> <li data-bbox="667 1421 1333 1470">■ <code>SQL_PERCENTAGE</code>: percentage filter of SQL tuning set statements</li> <li data-bbox="667 1486 1333 1591">■ <code>SQLSET_NAME</code>: name of the SQL tuning set to associate to the specified task or task execution. This parameter is mainly using in comparing two SQL tuning sets using SPA.</li> <li data-bbox="667 1608 1333 1656">■ <code>SQLSET_OWNER</code>: owner of the SQL tuning set specified using task parameter <code>SQLSET_NAME</code>.</li> <li data-bbox="667 1673 1333 1698">■ <code>TIME_LIMIT</code>: global time out (seconds)</li> <li data-bbox="667 1715 1333 1850">■ <code>WORKLOAD_IMPACT_THRESHOLD</code>: threshold of a SQL statement impact on a workload. Statements which workload change impact is below the absolute value of this threshold will be ignored and not considered for improvement or regression.</li> </ul>
value	New value of the specified parameter

## SET\_ANALYSIS\_DEFAULT\_PARAMETER Procedures

This procedure sets the SQL analysis task parameter default value.

### Syntax

This form of the procedure updates the default value of an analyzer parameter of type VARCHAR2.

```
DBMS_SQLPA.SET_ANALYSIS_DEFAULT_PARAMETER (  
    parameter    IN  VARCHAR2,  
    value        IN  VARCHAR2);
```

This form of the procedure updates the default value of an analyzer parameter of type NUMBER.

```
DBMS_SQLPA.SET_ANALYSIS_DEFAULT_PARAMETER (  
    parameter    IN  VARCHAR2,  
    value        IN  NUMBER);
```

## Parameters

**Table 138–11 SET\_ANALYSIS\_DEFAULT\_PARAMETER Procedure Parameters**

Parameter	Description
parameter	<p>Name of the parameter to set. The possible analysis parameters that can be set by this procedure are:</p> <ul style="list-style-type: none"> <li>■ <code>APPLY_CAPTURED_COMPILEENV</code>: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO).</li> <li>■ <code>BASIC_FILTER</code>: basic filter for SQL tuning set</li> <li>■ <code>COMPARISON_METRIC</code>: specify an expression of execution statistics to use in performance comparison (Example: <code>buffer_gets, cpu_time + buffer_gets * 10</code>)</li> <li>■ <code>DATABASE_LINK</code>: can be set to the global name of a PUBLIC database link. When it is set, SQL Performance Analyzer will use the database link for all <code>TEST EXECUTE</code> and <code>EXPLAIN PLAN</code> operations by sending the SQL statements to the remote database to be processed remotely. The analysis results will still be stored on the local database.</li> <li>■ <code>DAYS_TO_EXPIRE</code>: number of days until the task is deleted</li> <li>■ <code>DEFAULT_EXECUTION_TYPE</code>: the task will default to this type of execution when none is specified by the <a href="#">EXECUTE_ANALYSIS_TASK Function &amp; Procedure</a>.</li> <li>■ <code>EXECUTE_FULLDML</code>: TRUE to execute DML statement fully, including acquiring row locks and modifying rows; FALSE (default) to execute only the query part of the DML without modifying data. When TRUE, SQL Performance Analyzer will issue a rollback following DML execution to prevent persistent changes from being made by the DML.</li> <li>■ <code>EXECUTION_DAYS_TO_EXPIRE</code>: number of days until the tasks's executions will be deleted (without deleting the task)</li> <li>■ <code>EXECUTION_NAME1</code>: name of the first task execution to analyze</li> <li>■ <code>EXECUTION_NAME2</code>: name of the second task execution to analyze</li> <li>■ <code>LOCAL_TIME_LIMIT</code>: per-statement time out (seconds)</li> </ul>

**Table 138–11 (Cont.) SET\_ANALYSIS\_DEFAULT\_PARAMETER Procedure Parameters**

Parameter	Description
parameter (contd.)	<ul style="list-style-type: none"> <li data-bbox="748 264 1317 317">■ PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</li> <li data-bbox="748 327 1382 380">■ RANK_MEASURE1: first ranking measure for SQL tuning set</li> <li data-bbox="748 390 1382 443">■ RANK_MEASURE2: second possible ranking measure for SQL tuning set</li> <li data-bbox="748 453 1414 506">■ RANK_MEASURE3: third possible ranking measure for SQL tuning set</li> <li data-bbox="748 516 1414 569">■ RESUME_FILTER: a extra filter for SQL tuning sets besides BASIC_FILTER</li> <li data-bbox="748 579 1398 674">■ SQL_IMPACT_THRESHOLD: threshold of a change impact on a SQL statement. Same as the previous parameter, but at the level of the SQL statement.</li> <li data-bbox="748 684 1357 737">■ SQL_LIMIT: maximum number of SQL statements to process</li> <li data-bbox="748 747 1373 800">■ SQL_PERCENTAGE: percentage filter of SQL tuning set statements</li> <li data-bbox="748 810 1211 842">■ TIME_LIMIT: global time out (seconds)</li> <li data-bbox="748 852 1382 989">■ WORKLOAD_IMPACT_THRESHOLD: threshold of a SQL statement impact on a workload. Statements which workload change impact is below the absolute value of this threshold will be ignored and not considered for improvement or regression.</li> </ul>
value	New value of the specified parameter



The DBMS\_SQLTUNE package is the interface for tuning SQL on demand. The related package DBMS\_AUTO\_SQLTUNE package provides the interface for SQL Tuning Advisor run as an automated task.

The chapter contains the following topics:

- [Using DBMS\\_SQLTUNE](#)
  - Overview
  - Security Model
- [Data Structures](#)
- [Subprogram Groups](#)
  - SQL Tuning Advisor Subprograms
  - SQL Profile Subprograms
  - SQL Tuning Set Subprograms
  - Real-time SQL Monitoring Subprograms
  - SQL Performance Reporting Subprograms
- [Summary of DBMS\\_SQLTUNE Subprograms](#)

---

## Using DBMS\_SQLTUNE

- [Overview](#)
- [Security Model](#)



## Overview

The DBMS\_SQLTUNE package provides a number interrelated areas of functionality:

- [SQL Tuning Advisor Subprograms](#)
- [SQL Profile Subprograms](#)
- [SQL Tuning Set Subprograms](#)
- [Real-time SQL Monitoring Subprograms](#)

### SQL Tuning Advisor

The SQL Tuning Advisor is one of a suite of Advisors, a set of expert systems that identifies and helps resolve database performance problems. Specifically, the SQL Tuning Advisor automates the tuning process of problematic SQL statements. That is, it takes one or more SQL statements as input and gives precise advice on how to tune the statements. The advice is provided in the form of precise SQL actions for tuning the SQL along with their expected performance benefit.

The group of [SQL Tuning Advisor Subprograms](#) provide a task-oriented interface that lets you access the Advisor. You can call the following subprograms in the order given to use some of the SQL Tuning Advisor's features:

1. You use the [CREATE\\_TUNING\\_TASK Functions](#) to create a tuning task for tuning a single statement or a group of SQL statements.
2. The [EXECUTE\\_TUNING\\_TASK Function & Procedure](#) executes a previously created tuning task.
3. The [REPORT\\_TUNING\\_TASK Function](#) displays the results of a tuning task.
4. You use the [SCRIPT\\_TUNING\\_TASK Function](#) to create a SQL\*PLUS script which can then be executed to implement a set of Advisor recommendations

### SQL Profile Subprograms

The SQL Tuning Advisor may recommend the creation of a SQL profile to improve the performance of a statement. SQL profiles consist of auxiliary statistics specific to the statement. The query optimizer makes estimates about cardinality, selectivity, and cost that can sometimes be off by a significant amount, resulting in poor execution plans. The SQL profile addresses this problem by collecting additional information using sampling and partial execution techniques to adjust these estimates.

The group of [SQL Profile Subprograms](#) provides a mechanism for delivering statistics to the optimizer that targets one particular SQL statement, and helps the optimizer make good decisions for that statement by giving it the most accurate statistical information possible. For example:

- You can use the [ACCEPT\\_SQL\\_PROFILE Procedure and Function](#) to accept a SQL profile recommended by the SQL Tuning Advisor.
- You can alter the STATUS, NAME, DESCRIPTION, and CATEGORY attributes of an existing SQL profile with the [ALTER\\_SQL\\_PROFILE Procedure](#).
- You can drop a SQL profile with the [DROP\\_SQL\\_PROFILE Procedure](#).

### SQL Tuning Sets

The SQL Tuning Advisor input can be a single SQL statement or a set of statements. When tuning multiple statements in one advisor task, you give the input in the form of a SQL tuning set (STS). A SQL tuning set is a database object that stores SQL

statements along with their execution context in a system-provided schema. SQL tuning sets provide an infrastructure for dealing with SQL workloads and simplify tuning of a large number of SQL statements.

SQL tuning sets store SQL statements along with

- The execution context, such as the parsing schema name and bind values
- Execution statistics such as average elapsed time and execution count
- Execution plans - which are the sequence of operations Oracle performs to run SQL statements
- Row source statistics such as the number of rows processed for each operation executed within the plan

SQL tuning sets can be created by filtering or ranking SQL statements from several sources:

- The cursor cache using the [SELECT\\_CURSOR\\_CACHE Function](#)
- Top SQL statements from the Automatic Workload Repository using the [SELECT\\_WORKLOAD\\_REPOSITORY Functions](#)
- Other SQL tuning sets using the [SELECT\\_SQLSET Function](#)
- SQL Performance Analyzer task comparison results using the [SELECT\\_SQLPA\\_TASK Function](#)
- SQL Trace files using the [SELECT\\_SQL\\_TRACE Function](#)
- A user-defined workload

The complete group of [SQL Tuning Set Subprograms](#) facilitates this functionality. As examples:

- You use the [CREATE\\_SQLSET Procedure and Function](#) to create a SQL tuning set object in the database
- The [LOAD\\_SQLSET Procedure](#) populates the SQL tuning set with a set of selected SQL
- The [CAPTURE\\_CURSOR\\_CACHE\\_SQLSET Procedure](#) collects SQL statements from the cursor cache over a specified time interval, attempting to build a realistic picture of system workload.

### **Import/Export SQL Tuning Sets and SQL Profiles**

You use DBMS\_SQLTUNE subprograms to move SQL profiles and SQL tuning sets from one system to another using a common programmatic model. In both cases, you create a staging table on the source system and populate that staging table with the relevant data. You then move that staging table to the destination system following the method of your choice (such as datapump, import/export, or database link), where it is used to reconstitute the objects in their original form. These steps are implemented by means of subprograms included in this package:

1. Call the [CREATE\\_STGTAB\\_SQLPROF Procedure](#) or the [CREATE\\_STGTAB\\_SQLSET Procedure](#) to create the staging table on the source system.
2. Call the [PACK\\_STGTAB\\_SQLPROF Procedure](#) or [PACK\\_STGTAB\\_SQLSET Procedure](#) to populate the staging table with information from the source system.

3. Once you have moved the staging table to the destination system, you call the [UNPACK\\_STGTAB\\_SQLPROF Procedure](#) or the [UNPACK\\_STGTAB\\_SQLSET Procedure](#) to recreate the object on the new system.

**See Also:** *Oracle Database Performance Tuning Guide* for more information about programmatic flow

### Automatic Tuning Task Functions

The automated system task `SYS_AUTO_SQL_TUNING_TASK` is created by the database as part of the catalog scripts. This task automatically chooses a set of high-load SQL from AWR and runs the SQL Tuning Advisor on this SQL. The automated task performs the same comprehensive analysis as any other SQL Tuning task.

You can obtain a report on the activity of the Automatic SQL Tuning task through the `DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK` API. See the `DBMS_AUTO_SQLTUNE` package for the list of subprograms that you can use to manage the automated SQL tuning task.

**See Also:** [Using DBMS\\_AUTO\\_SQLTUNE](#)

### Real-time SQL Monitoring

Real-time SQL Monitoring allows DBAs or performance analysts to monitor the execution of long-running SQL statements while they are executing. Both cursor statistics (such as CPU times and IO times) and execution plan statistics (such as number of output rows, memory and temp space used) are updated close to real-time during statement execution. These statistics are exposed by the `V$SQL_MONITOR` and `V$SQL_PLAN_MONITOR` views. In addition, `DBMS_SQLTUNE` provides a subprogram `REPORT_SQL_MONITOR` to report on monitoring information.

## Security Model

This package is available to PUBLIC and performs its own security checking:

- As the SQL Tuning advisor relies on the advisor framework, all tuning task interfaces (XXX\_TUNING\_TASK) require privilege ADVISOR.
- SQL tuning set subprograms (XXX\_SQLSET) require either the ADMINISTER SQL TUNING SET or the ADMINISTER ANY SQL TUNING SET privilege. Users having the ADMINISTER SQL TUNING SET privilege can only create and modify a SQL tuning set they own, while the ADMINISTER ANY SQL TUNING SET privilege allows them to operate upon all SQL tuning sets, even those owned by other users. For example, using the [CREATE\\_SQLSET Procedure and Function](#) you can create a SQL tuning set to be owned by another user. In this case, the user need not necessarily have the ADMINISTER SQL TUNING SET privilege to operate upon her tuning set.
- Previously, three different privileges were needed to invoke subprograms concerned with SQL profiles:
  - CREATE ANY SQL PROFILE
  - ALTER ANY SQL PROFILE
  - DROP ANY SQL PROFILE

These have now been deprecated in favor of ADMINISTER SQL MANAGEMENT OBJECT

## Data Structures

The DBMS\_SQLTUNE package defines the following OBJECT type

### Object Types

- [SQLSET\\_ROW Object Type](#)

## SQLSET\_ROW Object Type

The `SQLSET_ROW` object models the content of a SQL tuning set for the user. Logically, a SQL tuning set is a collection of `SQLSET_ROWS` where each `SQLSET_ROW` contains a single SQL statement along with its execution context, statistics, binds, and plan. The `SELECT_XXX` subprograms each model a data source as a collection of `SQLSET_ROWS`, unique by (`sql_id`, `plan_hash_value`). Similarly, the `LOAD_SQLSET` procedure takes as input a cursor whose row type is `SQLSET_ROW`, treating each `SQLSET_ROW` in isolation according to the policies requested by the user.

Several subprograms in the `DBMS_SQLTUNE` package accept basic filters on the content of a SQL tuning set or data source. These filters are expressed in terms of the attributes within the `SQLSET_ROW` as defined.

### Syntax

```
CREATE TYPE sqlset_row AS object (
  sql_id                VARCHAR(13),
  force_matching_signature NUMBER,
  sql_text              CLOB,
  object_list           sql_objects,
  bind_data             RAW(2000),
  parsing_schema_name  VARCHAR2(30),
  module                VARCHAR2(48),
  action                VARCHAR2(32),
  elapsed_time          NUMBER,
  cpu_time              NUMBER,
  buffer_gets           NUMBER,
  disk_reads            NUMBER,
  direct_writes         NUMBER,
  rows_processed        NUMBER,
  fetches               NUMBER,
  executions            NUMBER,
  end_of_fetch_count    NUMBER,
  optimizer_cost        NUMBER,
  optimizer_env         RAW(2000),
  priority              NUMBER,
  command_type          NUMBER,
  first_load_time       VARCHAR2(19),
  stat_period           NUMBER,
  active_stat_period    NUMBER,
  other                 CLOB,
  plan_hash_value       NUMBER,
  sql_plan              sql_plan_table_type,
  bind_list             sql_binds)
```

### Attributes

**Table 139–1** *SQLSET\_ROW Attributes*

Attribute	Description
<code>sql_id</code>	Unique SQL ID
<code>forcing_matching_signature</code>	Signature with literals, case, and whitespace removed
<code>sql_text</code>	Full text for the statement
<code>object_list</code>	Currently not implemented

**Table 139-1 (Cont.) SQLSET\_ROW Attributes**

<b>Attribute</b>	<b>Description</b>
bind_data	Bind data as captured for this SQL. Note that you cannot stipulate an argument for this parameter and also for bind_list - they are mutually exclusive.
parsing_schema_name	Schema where the SQL is parsed
module	Last application module for the SQL
action	Last application action for the SQL
elapsed_time	Sum total elapsed time for this SQL statement
cpu_time	Sum total CPU time for this SQL statement
buffer_gets	Sum total number of buffer gets
disk_reads	Sum total number of disk reads
direct_writes	Sum total number of direct writes
rows_processed	Sum total number of rows processed by this SQL
fetches	Sum total number of fetches
executions	Total executions of this SQL
end_of_fetch_count	Number of times the statement was fully executed with all of its rows fetched
optimizer_cost	Optimizer cost for this SQL
optimizer_env	Optimizer environment for this SQL statement
priority	User-defined priority (1,2,3)
command_type	Statement type, such as INSERT or SELECT.
first_load_time	Load time of parent cursor
stat_period	Period of time (seconds) when the statistics of this SQL statement were collected
active_stat_period	Effective period of time (in seconds) during which the SQL statement was active
other	Other column for user defined attributes
plan_hash_value	Plan hash value of the plan
sql_plan	Explain plan
bind_list	List of user specified binds for SQL This is used for user-specified workloads. Note that you cannot stipulate an argument for this parameter and also for bind_data - they are mutually exclusive.

---

## Subprogram Groups

DBMS\_SQLTUNE subprograms are grouped by function:

- [SQL Tuning Advisor Subprograms](#)
- [SQL Profile Subprograms](#)
- [SQL Tuning Set Subprograms](#)
- [Real-time SQL Monitoring Subprograms](#)
- [SQL Performance Reporting Subprograms](#)



## SQL Tuning Advisor Subprograms

This subprogram group provides an interface to manage SQL tuning tasks.

**Table 139–2 SQL Tuning Task Subprograms**

Subprogram	Description
<a href="#">CANCEL_TUNING_TASK Procedure</a> on page 139-25	Cancels the currently executing tuning task
<a href="#">CREATE_SQL_PLAN_BASELINE Procedure</a> on page 139-28	Creates a SQL plan baseline for an existing plan
<a href="#">CREATE_TUNING_TASK Functions</a> on page 139-33	Creates a tuning of a single statement or SQL tuning set for either the SQL Tuning Advisor
<a href="#">DROP_TUNING_TASK Procedure</a> on page 139-40	Drops a SQL tuning task
<a href="#">EXECUTE_TUNING_TASK Function &amp; Procedure</a> on page 139-41	Executes a previously created tuning task
<a href="#">IMPLEMENT_TUNING_TASK Function</a> on page 139-42	Implements a set of SQL profile recommendations made by the SQL Tuning Advisor
<a href="#">INTERRUPT_TUNING_TASK Procedure</a> on page 139-43	Interrupts the currently executing tuning task
<a href="#">REPORT_AUTO_TUNING_TASK Function</a> on page 139-54	Displays a report from the automatic tuning task, reporting on a range of executions
<a href="#">REPORT_TUNING_TASK Function</a> on page 139-66	Displays the results of a tuning task
<a href="#">RESET_TUNING_TASK Procedure</a> on page 139-68	Resets the currently executing tuning task to its initial state
<a href="#">RESUME_TUNING_TASK Procedure</a> on page 139-69	Resumes a previously interrupted task that was created to process a SQL tuning set
<a href="#">SCRIPT_TUNING_TASK Function</a> on page 139-70	Creates a SQL*PLUS script which can then be executed to implement a set of SQL Tuning Advisor recommendations
<a href="#">SET_TUNING_TASK_PARAMETER Procedures</a> on page 139-84	Updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER

The [Summary of DBMS\\_SQLTUNE Subprograms](#) contains a complete listing of all subprograms in the package.

## SQL Profile Subprograms

This subprogram group provides an interface to manage SQL profiles.

**Table 139–3 SQL Profile Subprograms**

Subprogram	Description
<a href="#">ACCEPT_SQL_PROFILE Procedure and Function</a> on page 139-19	Creates a SQL profile for the specified tuning task
<a href="#">ALTER_SQL_PROFILE Procedure</a> on page 139-23	Alters specific attributes of an existing SQL profile object
<a href="#">CREATE_STGTAB_SQLPROF Procedure</a> on page 139-30	Creates the staging table used for copying SQL profiles from one system to another
<a href="#">DROP_SQL_PROFILE Procedure</a> on page 139-38	Drops the named SQL profile from the database
<a href="#">PACK_STGTAB_SQLPROF Procedure</a> on page 139-48	Moves profile data out of the <code>SYS</code> schema into the staging table
<a href="#">REMAP_STGTAB_SQLPROF Procedure</a> on page 139-51	Changes the profile data values kept in the staging table prior to performing an unpack operation
<a href="#">SQLTEXT_TO_SIGNATURE Function</a> on page 139-86	Returns a SQL text's signature
<a href="#">UNPACK_STGTAB_SQLPROF Procedure</a> on page 139-87	Uses the profile data stored in the staging table to create profiles on this system

The [Summary of DBMS\\_SQLTUNE Subprograms](#) contains a complete listing of all subprograms in the package.

## SQL Tuning Set Subprograms

This subprogram group provides an interface to manage SQL tuning sets.

**Table 139–4 SQL Tuning Set Subprograms**

Subprogram	Description
<a href="#">ADD_SQLSET_REFERENCE Function</a> on page 139-22	Adds a new reference to an existing SQL tuning set to indicate its use by a client
<a href="#">CAPTURE_CURSOR_CACHE_SQLSET Procedure</a> on page 139-26	Over a specified time interval incrementally captures a workload from the cursor cache into a SQL tuning set
<a href="#">CREATE_SQLSET Procedure and Function</a> on page 139-29	Creates a SQL tuning set object in the database
<a href="#">CREATE_STGTAB_SQLSET Procedure</a> on page 139-31	Creates a staging table through which SQL Tuning Sets are imported and exported
<a href="#">DELETE_SQLSET Procedure</a> on page 139-37	Deletes a set of SQL statements from a SQL tuning set
<a href="#">DROP_SQLSET Procedure</a> on page 139-39	Drops a SQL tuning set if it is not active
<a href="#">LOAD_SQLSET Procedure</a> on page 139-44	Populates the SQL tuning set with a set of selected SQL
<a href="#">PACK_STGTAB_SQLSET Procedure</a> on page 139-49	Copies tuning sets out of the <code>SYS</code> schema into the staging table
<a href="#">REMOVE_SQLSET_REFERENCE Procedure</a> on page 139-53	Deactivates a SQL tuning set to indicate it is no longer used by the client
<a href="#">SELECT_CURSOR_CACHE Function</a> on page 139-72	Collects SQL statements from the cursor cache
<a href="#">SELECT_SQL_TRACE Function</a> on page 139-76	Reads the content of one or more trace files and returns the SQL statements it finds in the format of <code>sqlset_row</code>
<a href="#">SELECT_SQLPA_TASK Function</a> on page 139-78	Collects SQL statements from a SQL performance analyzer comparison task
<a href="#">SELECT_SQLSET Function</a> on page 139-80	Collects SQL statements from an existing SQL tuning set
<a href="#">SELECT_WORKLOAD_REPOSITORY Functions</a> on page 139-82	Collects SQL statements from the workload repository
<a href="#">UNPACK_STGTAB_SQLSET Procedure</a> on page 139-88	Copies one or more SQL tuning sets from the staging table
<a href="#">UPDATE_SQLSET Procedures</a> on page 139-90	Updates whether selected string fields for a SQL statement in a SQL tuning set or the set numerical attributes of a SQL in a SQL tuning set

The [Summary of DBMS\\_SQLTUNE Subprograms](#) contains a complete listing of all subprograms in the package.

## Real-time SQL Monitoring Subprograms

This subprogram group provides function to report on monitoring data collected in V\$SQL\_MONITOR and V\$SQL\_PLAN\_MONITOR.

**Table 139–5 SQL Tuning Set Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">REPORT_SQL_MONITOR Function</a> on page 139-59	Reports on real-time SQL Monitoring
<a href="#">REPORT_SQL_MONITOR_LIST Function</a> on page 139-64	Builds a report for all or a sub-set of statements monitored by Oracle

---

## SQL Performance Reporting Subprograms

This subprogram group provides detailed reports on SQL performance using statistics from the cursor cache and automatic workload repository (AWR).

**Table 139–6 SQL Performance Reporting Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">REPORT_SQL_DETAIL Function</a> on page 139-56	Reports on a specific SQLID

## Summary of DBMS\_SQLTUNE Subprograms

**Table 139–7 DBMS\_SQLTUNE Package Subprograms**

Subprogram	Description	Group
<a href="#">ACCEPT_SQL_PROFILE Procedure and Function</a> on page 139-19	Create a SQL profile for the specified tuning task	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">ADD_SQLSET_REFERENCE Function</a> on page 139-22	Adds a new reference to an existing SQL tuning set to indicate its use by a client	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">ALTER_SQL_PROFILE Procedure</a> on page 139-23	Alters specific attributes of an existing SQL profile object	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">CANCEL_TUNING_TASK Procedure</a> on page 139-25	Cancels the currently executing tuning task	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">CAPTURE_CURSOR_CACHE_SQLSET Procedure</a> on page 139-26	Over a specified time interval incrementally captures a workload from the cursor cache into a SQL tuning set	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">CREATE_SQL_PLAN_BASELINE Procedure</a> on page 139-28	Creates a SQL plan baseline for an existing plan	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">CREATE_SQLSET Procedure and Function</a> on page 139-29	Creates a SQL tuning set object in the database	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">CREATE_STGTAB_SQLPROF Procedure</a> on page 139-30	Creates the staging table used for copying SQL profiles from one system to another	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">CREATE_STGTAB_SQLSET Procedure</a> on page 139-31	Creates a staging table through which SQL tuning sets are imported and exported	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">CREATE_TUNING_TASK Functions</a> on page 139-33	Creates a tuning of a single statement or SQL tuning set for either the SQL Tuning Advisor	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">DELETE_SQLSET Procedure</a> on page 139-37	Deletes a set of SQL statements from a SQL tuning set	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">DROP_SQL_PROFILE Procedure</a> on page 139-38	Drops the named SQL profile from the database	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">DROP_SQLSET Procedure</a> on page 139-39	Drops a SQL tuning set if it is not active	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">DROP_TUNING_TASK Procedure</a> on page 139-40	Drops a SQL tuning task	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11

**Table 139-7 (Cont.) DBMS\_SQLTUNE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">EXECUTE_TUNING_TASK Function &amp; Procedure</a> on page 139-41	Executes a previously created tuning task	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">IMPLEMENT_TUNING_TASK Function</a> on page 139-42	implements a set of SQL profile recommendations made by the SQL Tuning Advisor	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">INTERRUPT_TUNING_TASK Procedure</a> on page 139-43	Interrupts the currently executing tuning task	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">LOAD_SQLSET Procedure</a> on page 139-44	Populates the SQL tuning set with a set of selected SQL	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">PACK_STGTAB_SQLPROF Procedure</a> on page 139-48	Moves profile data out of the SYS schema into the staging table	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">PACK_STGTAB_SQLSET Procedure</a> on page 139-49	Moves tuning sets out of the SYS schema into the staging table	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">REMAP_STGTAB_SQLPROF Procedure</a> on page 139-51	Changes the profile data values kept in the staging table prior to performing an unpack operation	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">REMAP_STGTAB_SQLSET Procedure</a> on page 139-52	Changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">REMOVE_SQLSET_REFERENCE Procedure</a> on page 139-53	Deactivates a SQL tuning set to indicate it is no longer used by the client	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">REPORT_AUTO_TUNING_TASK Function</a> on page 139-54	Displays a report from the automatic tuning task, reporting on a range of subtasks	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">REPORT_SQL_DETAIL Function</a> on page 139-56	Reports on a specific SQLID	<a href="#">SQL Performance Reporting Subprograms</a> on page 139-15
<a href="#">REPORT_SQL_MONITOR Function</a> on page 139-59	Displays a report on real-time SQL monitoring	<a href="#">Real-time SQL Monitoring Subprograms</a> on page 139-14
<a href="#">REPORT_SQL_MONITOR_LIST Function</a> on page 139-64	Builds a report for all or a sub-set of statements monitored by Oracle	<a href="#">Real-time SQL Monitoring Subprograms</a> on page 139-14
<a href="#">REPORT_TUNING_TASK Function</a> on page 139-66	Displays the results of a tuning task	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">RESET_TUNING_TASK Procedure</a> on page 139-68	Resets the currently executing tuning task to its initial state	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11

**Table 139-7 (Cont.) DBMS\_SQLTUNE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">RESUME_TUNING_TASK Procedure</a> on page 139-69	Resumes a previously interrupted task that was created to process a SQL tuning set	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">SCRIPT_TUNING_TASK Function</a> on page 139-70	Creates a SQL*PLUS script which can then be executed to implement a set of SQL Tuning Advisor recommendations	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">SELECT_CURSOR_CACHE Function</a> on page 139-72	Collects SQL statements from the cursor cache	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">SELECT_SQL_TRACE Function</a> on page 139-76	Reads the content of one or more trace files and returns the SQL statements it finds in the format of <code>sqlset_row</code>	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">SELECT_SQLSET Function</a> on page 139-80	Collects SQL statements from an existing SQL tuning set	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">SELECT_WORKLOAD_REPOSITORY Functions</a> on page 139-82	Collects SQL statements from the workload repository	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">SET_TUNING_TASK_PARAMETER Procedures</a> on page 139-84	Updates the value of a SQL tuning parameter of type <code>VARCHAR2</code> or <code>NUMBER</code>	<a href="#">SQL Tuning Advisor Subprograms</a> on page 139-11
<a href="#">SQLTEXT_TO_SIGNATURE Function</a> on page 139-86	Returns a SQL text's signature	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">UNPACK_STGTAB_SQLPROF Procedure</a> on page 139-87	Uses the profile data stored in the staging table to create profiles on this system	<a href="#">SQL Profile Subprograms</a> on page 139-12
<a href="#">UNPACK_STGTAB_SQLSET Procedure</a> on page 139-88	Moves one or more SQL tuning sets from the staging table	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13
<a href="#">UPDATE_SQLSET Procedures</a> on page 139-90	Updates selected fields for a SQL statement in a SQL tuning set	<a href="#">SQL Tuning Set Subprograms</a> on page 139-13



## ACCEPT\_SQL\_PROFILE Procedure and Function

This procedure creates a SQL profile recommended by the SQL Tuning Advisor. The SQL text is normalized for matching purposes though it is stored in the data dictionary in de-normalized form for readability. SQL text is provided through a reference to the SQL Tuning task. If the referenced SQL statement doesn't exist, an error is reported.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
  task_name      IN  VARCHAR2,
  object_id      IN  NUMBER      := NULL,
  name           IN  VARCHAR2    := NULL,
  description    IN  VARCHAR2    := NULL,
  category       IN  VARCHAR2    := NULL);
task_owner      IN  VARCHAR2    := NULL,
replace         IN  BOOLEAN      := FALSE,
force_match     IN  BOOLEAN      := FALSE,
profile_type    IN  VARCHAR2    := REGULAR_PROFILE);
```

```
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
  task_name      IN  VARCHAR2,
  object_id      IN  NUMBER      := NULL,
  name           IN  VARCHAR2    := NULL,
  description    IN  VARCHAR2    := NULL,
  category       IN  VARCHAR2    := NULL;
  task_owner     IN  VARCHAR2    := NULL,
  replace        IN  BOOLEAN      := FALSE,
  force_match    IN  BOOLEAN      := FALSE,
  profile_type   IN  VARCHAR2    := REGULAR_PROFILE)
RETURN VARCHAR2;
```

### Parameters

**Table 139–8** *ACCEPT\_SQL\_PROFILE Procedure and Function Parameters*

Parameter	Description
task_name	The (mandatory) name of the SQL tuning task
object_id	The identifier of the advisor framework object representing the SQL statement associated with the tuning task
name	The name of the SQL profile. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system generates a unique name for the SQL profile.
description	A user specified string describing the purpose of the SQL profile. The description is truncated if longer than 256 characters. The maximum size is 500 characters.

**Table 139–8 (Cont.) ACCEPT\_SQL\_PROFILE Procedure and Function Parameters**

Parameter	Description
category	This is the category name which must match the value of the <code>SQLTUNE_CATEGORY</code> parameter in a session for the session to use this SQL profile. It defaults to the value "DEFAULT". This is also the default of the <code>SQLTUNE_CATEGORY</code> parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name creates a unique key for a SQL profile. An <code>ACCEPT_SQL_PROFILE</code> fails if this combination is duplicated.
task_owner	Owner of the tuning task. This is an optional parameter that has to be specified to accept a SQL profile associated to a tuning task owned by another user. The current user is the default value.
replace	If the profile already exists, it is replaced if this argument is <code>TRUE</code> . It is an error to pass a name that is already being used for another signature/category pair, even with <code>replace</code> set to <code>TRUE</code> .
force_match	If <code>TRUE</code> this causes SQL profiles to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the <code>FORCE</code> option of the <code>cursor_sharing</code> parameter.  If <code>FALSE</code> , literals are not transformed. This is analogous to the matching algorithm used by the <code>EXACT</code> option of the <code>cursor_sharing</code> parameter.
profile_type	Options: <ul style="list-style-type: none"> <li>▪ <code>REGULAR_PROFILE</code> - profile without a change to parallel execution (Default, equivalent to <code>NULL</code>). Note that if the SQL statement currently has a parallel execution plan, the regular profile will cause the optimizer to choose a different, but still parallel, execution plan.</li> <li>▪ <code>PX_PROFILE</code> - regular profile with a change to parallel execution</li> </ul>

## Return Values

The name of the SQL profile.

## Usage Notes

The `CREATE ANY SQL PROFILE` privilege is required.

## Examples

You use both the procedure and the function versions of the subprogram in the same way except you must specify a return value to invoke the function. Here we give examples of the procedure only.

In this example, you tune a single SQL statement from the workload repository and you create the SQL profile recommended by SQL Tuning Advisor.

```
variable stmt_task VARCHAR2(64);
variable sts_task VARCHAR2(64);

-- create a tuning task tune the statement
```

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK (
  begin_snap => 1, -
  end_snap   => 2, -
  sql_id     => 'ay1m3ssvtrh24');
```

```
-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);
```

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(:stmt_task);
```

Note that you do not have to specify the ID (that is, `object_id`) for the advisor framework object created by SQL Tuning Advisor to represent the tuned SQL statement.

You might also want to accept the recommended SQL profile in a different category, (for example, `TEST`), so that it is not used by default.

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
  task_name => :stmt_task, -
  category  => 'TEST');
```

You can use command `ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST'` to see how this profile behaves.

The following call creates a SQL profile that targets any SQL statement with the same `force_matching_signature` as the tuned statement.

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (task_name => :stmt_task, -
                                       force_match => TRUE);
```

In the following example, you tune a SQL tuning set, and you create a SQL profile for only one of the SQL statements in the SQL tuning set. The SQL statement is represented by an advisor framework object with ID equal to '5'. Please notice that you must pass an object id to the `ACCEPT_SQL_PROFILE` procedure because there are potentially many SQL profiles for the tuning task. This object id is given along with the report.

```
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK ( -
  sqlset_name => 'my_workload', -
  rank1       => 'ELAPSED_TIME', -
  time_limit  => 3600,          -
  description => 'my workload ordered by elapsed time');
```

```
-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);
```

```
-- create the profile for the sql statement corresponding to object_id = 5.
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
  task_name => :sts_task, -
  object_id => 5);
```

## ADD\_SQLSET\_REFERENCE Function

This procedure adds a new reference to an existing SQL tuning set to indicate its use by a client.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.ADD_SQLSET_REFERENCE (
    sqlset_name IN VARCHAR2,
    description IN VARCHAR2 := NULL)
RETURN NUMBER;
```

### Parameters

**Table 139–9 ADD\_SQLSET\_REFERENCE Function Parameters**

Parameter	Description
sqlset_name	The SQL tuning set name
description	The description of the usage of SQL tuning set. The description is truncated if longer than 256 characters.

### Return Values

The identifier of the added reference.

### Examples

You can add reference to a SQL tuning set. This prevents the tuning set from being modified while it is being used. References are automatically added when you invoke SQL Tuning Advisor on the SQL tuning set, so you should use this function for custom purposes only.

The function returns a reference ID that is used to remove it later. You use the `REMOVE_SQLSET_REFERENCE` Procedure to delete references to a SQL tuning set.

```
variable rid number;
EXEC :rid := DBMS_SQLTUNE.ADD_SQLSET_REFERENCE( -
    sqlset_name => 'my_workload', -
    description => 'my sts reference');
```

You can use the views `USER/DBA_SQLSET_REFERENCES` to find all references on a given SQL tuning set.

## ALTER\_SQL\_PROFILE Procedure

This procedure alters specific attributes of an existing SQL profile object. The following attributes can be altered (using these attribute names):

- "STATUS" can be set to "ENABLED" or "DISABLED"
- "NAME" can be reset to a valid name which must be a valid Oracle identifier and must be unique.
- "DESCRIPTION" can be set to any string of size no more than 500 characters
- "CATEGORY" can be reset to a valid category name which must be a valid Oracle identifier and must be unique when combined with normalized SQL text)

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.ALTER_SQL_PROFILE (
  name           IN  VARCHAR2,
  attribute_name IN  VARCHAR2,
  value          IN  VARCHAR2);
```

### Parameters

**Table 139–10 ALTER\_SQL\_PROFILE Procedure Parameters**

Parameter	Description
name	The (mandatory) name of the existing SQL profile to alter
attribute_name	The (mandatory) attribute name to alter (case insensitive) using valid attribute names
value	The (mandatory) new value of the attribute using valid attribute values

### Usage Notes

Requires the ALTER ANY SQL PROFILE privilege.

### Examples

```
-- Disable a profile, so it is not be used by any sessions.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name           => :pname, -
                                     attribute_name => 'STATUS', -
                                     value          => 'DISABLED');

-- Enable it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name           => :pname, -
                                     attribute_name => 'STATUS', -
                                     value          => 'ENABLED');

-- Change the category of the profile so it is used only by sessions
-- with category set to TEST.
-- Use ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST' to see how this profile
-- behaves.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name           => :pname, -
                                     attribute_name => 'CATEGORY', -
                                     value          => 'TEST');
```

```
-- Change it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name      => :pname, -
                                     attribute_name => 'CATEGORY', -
                                     value         => 'DEFAULT');
```

## CANCEL\_TUNING\_TASK Procedure

This procedure cancels the currently executing tuning task. All intermediate result data is deleted.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.CANCEL_TUNING_TASK(  
  task_name          IN VARCHAR2);
```

### Parameters

**Table 139-11** CANCEL\_TUNING\_TASK Procedure Parameters

Parameter	Description
task_name	The name of the task to cancel

### Examples

You cancel a task when you need to stop it executing and do not require to view any already-completed results.

```
EXEC DBMS_SQLTUNE.CANCEL_TUNING_TASK(:my_task);
```

## CAPTURE\_CURSOR\_CACHE\_SQLSET Procedure

Over a specified time interval this procedure incrementally captures a workload from the cursor cache into a SQL tuning set. The procedure captures a workload from the cursor cache into a SQL tuning set, polling the cache multiple times over a time period and updating the workload data stored there. It can execute over as long a period as required to capture an entire system workload.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET (
  sqlset_name          IN VARCHAR2,
  time_limit           IN POSITIVE := 1800,
  repeat_interval      IN POSITIVE := 300,
  capture_option       IN VARCHAR2 := 'MERGE',
  capture_mode         IN NUMBER   := MODE_REPLACE_OLD_STATS,
  basic_filter         IN VARCHAR2 := NULL,
  sqlset_owner        IN VARCHAR2 := NULL,
  recursive_sql        IN VARCHAR2 := HAS_RECURSIVE_SQL);
```

### Parameters

**Table 139–12** CAPTURE\_CURSOR\_CACHE\_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
time_limit	The total amount of time, in seconds, to execute
repeat_interval	The amount of time, in seconds, to pause between sampling
capture_option	During capture, either insert new statements, update existing statements, or both. 'INSERT', 'UPDATE', or 'MERGE' just like load_option in load_sqlset
capture_mode	Capture mode (UPDATE and MERGE capture options). Possible values: <ul style="list-style-type: none"> <li>■ MODE_REPLACE_OLD_STATS - Replace statistics when the number of executions seen is greater than that stored in the SQL tuning set</li> <li>■ MODE_ACCUMULATE_STATS - Add new values to current values for SQL we already store. Note that this mode detects if a statement has been aged out, so the final value for a statistics is the sum of the statistics of all cursors that statement existed under.</li> </ul>
basic_filter	Filter to apply to cursor cache on each sampling (see SELECT_XXX subprograms). If basic_filter is not set by the caller, the subprogram captures only statements of the type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.
sqlset_owner	The owner of the SQL tuning set or NULL for current schema owner
recursive_sql	Filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).



## Examples

In this example capture takes place over a 30-second period, polling the cache once every five seconds. This captures all statements run during that period but not before or after. If the same statement appears a second time, the process replaces the stored statement with the new occurrence.

Note that in production systems the time limit and repeat interval would be set much higher. You should tune the `time_limit` and `repeat_interval` parameters based on the workload time and cursor cache turnover properties of your system.

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
    sqlset_name      => 'my_workload', -
    time_limit       => 30, -
    repeat_interval  => 5);
```

In the following call you accumulate execution statistics as you go. This option produces an accurate picture of the cumulative activity of each cursor, even across age-outs, but it is more expensive than the previous example.

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
    sqlset_name      => 'my_workload', -
    time_limit       => 30, -
    repeat_interval  => 5, -
    capture_mode     => dbms_sqltune.MODE_ACCUMULATE_STATS);
```

This call performs a very inexpensive capture where you only insert new statements and do not update their statistics once they have been inserted into the SQL tuning set

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
    sqlset_name      => 'my_workload', -
    time_limit       => 30, -
    repeat_interval  => 5, -
    capture_option   => 'INSERT');
```

## CREATE\_SQL\_PLAN\_BASELINE Procedure

This procedure creates a SQL plan baseline for an execution plan. It can be used in the context of an Alternative Plan Finding made by the SQL Tuning Advisor.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.CREATE_SQL_PLAN_BASELINE (  
    task_name          IN VARCHAR2,  
    object_id          IN NUMBER := NULL,  
    plan_hash_value    IN NUMBER,  
    owner_name         IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–13** CREATE\_SQL\_PLAN\_BASELINE Procedure Parameters

Parameter	Description
task_name	Name of the task for which to get a script
object_id	Object ID to which the SQL corresponds
plan_hash_value	Plan to create plan baseline
owner_name	Owner of the relevant tuning task. Defaults to the current schema owner.

## CREATE\_SQLSET Procedure and Function

The procedure creates a SQL tuning set object in the database.

The function causes the system to generate a name for the SQL tuning set.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.CREATE_SQLSET (
  sqlset_name IN VARCHAR2,
  description IN VARCHAR2 := NULL,
  sqlset_owner IN VARCHAR2 := NULL);
```

```
DBMS_SQLTUNE.CREATE_SQLSET (
  sqlset_name IN VARCHAR2 := NULL,
  description IN VARCHAR2 := NULL,
  sqlset_owner IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 139–14** CREATE\_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
description	The description of the SQL tuning set
sqlset_owner	The owner of the SQL tuning set, or NULL for the current schema owner

### Examples

```
EXEC DBMS_SQLTUNE.CREATE_SQLSET(-
  sqlset_name => 'my_workload', -
  description => 'complete application workload');
```

## CREATE\_STGTAB\_SQLPROF Procedure

This procedure creates the staging table used for copying SQL profiles from one system to another.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (
  table_name          IN VARCHAR2,
  schema_name        IN VARCHAR2 := NULL,
  tablespace_name     IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–15** CREATE\_STGTAB\_SQLPROF Procedure Parameters

Parameter	Description
table_name	The name of the table to create (case-insensitive unless double quoted). Required.
schema_name	The schema to create the table in, or NULL for current schema (case-insensitive unless double quoted)
tablespace_name	The tablespace to store the staging table within, or NULL for current user's default tablespace (case-insensitive unless double quoted)

### Usage Notes

- Call this procedure once before issuing a call to the [PACK\\_STGTAB\\_SQLPROF Procedure](#).
- This procedure can be called multiple times if you would like to have different SQL profiles in different staging tables.
- Note that this is a DDL operation, so it does not occur within a transaction.

### Examples

Create a staging table to store profile data that can be moved to another system.

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (table_name => 'PROFILE_STGTAB');
```

## CREATE\_STGTAB\_SQLSET Procedure

This procedure creates a staging table through which SQL tuning sets are imported and exported

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.CREATE_STGTAB_SQLSET (
  table_name          IN VARCHAR2,
  schema_name        IN VARCHAR2 := NULL,
  tablespace_name    IN VARCHAR2 := NULL,
  db_version         IN NUMBER   := NULL);
```

### Parameters

**Table 139–16** CREATE\_STGTAB\_SQLSET Procedure Parameters

Parameter	Description
table_name	Name of the table to create (case-sensitive)
schema_name	Schema in which to create the table in, or NULL for current schema (case-sensitive)
tablespace_name	Tablespace in which to store the staging table, or NULL for current user's default tablespace (case-sensitive)
db_version	Database (DB) version determining the format of the staging table. User can also create an older DB version staging table to export STS to an older DB version. One of the following values: <ul style="list-style-type: none"> <li>■ NULL (default) — current DB version</li> <li>■ STS_STGTAB_10_2_VERSION — 10.2 DB version</li> <li>■ STS_STGTAB_11_1_VERSION — 11.1 DB version</li> <li>■ STS_STGTAB_11_2_VERSION — 11.2 DB version</li> </ul>

### Usage Notes

- Call this procedure once before issuing a call to the [PACK\\_STGTAB\\_SQLSET Procedure](#).
- This procedure can be called multiple times if you would like to have different tuning sets in different staging tables.
- Note that this is a DDL operation, so it does not occur within a transaction.
- Users issuing the call must have permission to CREATE TABLE in the schema provided and the relevant tablespace.
- Please note that the staging table contains nested table columns and indexes, so it should not be renamed.

### Examples

**Create a staging table for packing and eventually exporting a SQL tuning sets**

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(table_name => 'STGTAB_SQLSET');
```

**Create a staging table to pack a SQL tuning set in Oracle Database 10g Release 2 (10.2) format**

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(  
    table_name => 'STGTAB_SQLSET',  
    db_version => DBMS_SQLTUNE.STS_STGTAB_10_2_VERSION)
```

**Create a staging table to pack a SQL tuning set in Oracle Database 11g Release 1 (11.1) format**

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(  
    table_name => 'STGTAB_SQLSET',  
    db_version => DBMS_SQLTUNE.STS_STGTAB_11_1_VERSION)
```

## CREATE\_TUNING\_TASK Functions

You can use different forms of this function to:

- Create a tuning task for a single statement given its text.
- Create a tuning task for a single statement from the Cursor Cache given its identifier.
- Create a tuning task for a single statement from the workload repository given a range of snapshot identifiers.
- Create a tuning task for a SQL tuning set.
- Create tuning task for a SQL Performance Analyzer

In all cases, the function mainly creates an advisor task and sets its parameters.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

SQL text format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sql_text          IN CLOB,
  bind_list         IN sql_binds := NULL,
  user_name         IN VARCHAR2 := NULL,
  scope             IN VARCHAR2 := SCOPE_COMPREHENSIVE,
  time_limit        IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name         IN VARCHAR2 := NULL,
  description       IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

SQL ID format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sql_id           IN VARCHAR2,
  plan_hash_value  IN NUMBER    := NULL,
  scope           IN VARCHAR2 := SCOPE_COMPREHENSIVE,
  time_limit      IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name       IN VARCHAR2 := NULL,
  description     IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

Workload Repository format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK(
  begin_snap      IN NUMBER,
  end_snap        IN NUMBER,
  sql_id          IN VARCHAR2,
  plan_hash_value IN NUMBER    := NULL,
  scope          IN VARCHAR2 := SCOPE_COMPREHENSIVE,
  time_limit     IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name      IN VARCHAR2 := NULL,
  description    IN VARCHAR2 := NULL)
RETURN VARCHAR2;
```

SQLSET format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK(
  sqlset_name     IN VARCHAR2,
```

```

basic_filter      IN VARCHAR2 := NULL,
object_filter     IN VARCHAR2 := NULL,
rank1            IN VARCHAR2 := NULL,
rank2            IN VARCHAR2 := NULL,
rank3            IN VARCHAR2 := NULL,
result_percentage IN NUMBER   := NULL,
result_limit     IN NUMBER   := NULL,
scope            IN VARCHAR2 := SCOPE_COMPREHENSIVE,
time_limit       IN NUMBER   := TIME_LIMIT_DEFAULT,
task_name        IN VARCHAR2 := NULL,
description      IN VARCHAR2 := NULL
plan_filter      IN VARCHAR2 := 'MAX_ELAPSED_TIME',
sqlset_owner     IN VARCHAR2 := NULL)
RETURN VARCHAR2;

```

#### SQL Performance Analyzer format:

```

DBMS_SQLTUNE.CREATE_TUNING_TASK(
spa_task_name     IN VARCHAR2,
spa_task_owner   IN VARCHAR2 := NULL,
spa_compare_exec IN VARCHAR2 := NULL,
basic_filter      IN VARCHAR2 := NULL,
time_limit       IN NUMBER   := TIME_LIMIT_DEFAULT,
task_name        IN VARCHAR2 := NULL,
description      IN VARCHAR2 := NULL)
RETURN VARCHAR2;

```

## Parameters

**Table 139–17 CREATE\_TUNING\_TASK Function Parameters**

Parameter	Description
sql_text	Text of a SQL statement
begin_snap	Begin snapshot identifier
end_snap	End snapshot identifier
sql_id	Identifier of a SQL statement
bind_list	An ordered list of bind values in ANYDATA type
plan_hash_value	Hash value of the SQL execution plan
sqlset_name	SQL tuning set name
basic_filter	SQL predicate to filter the SQL from the SQL tuning set
object_filter	Object filter
rank(i)	Order-by clause on the selected SQL
result_percentage	Percentage on the sum of a ranking measure
result_limit	Top L(imit) SQL from the (filtered/ranked) SQL
user_name	Username for whom the statement is to be tuned
scope	Tuning scope (limited/comprehensive)
time_limit	The maximum duration in seconds for the tuning session
task_name	Optional tuning task name
description	Description of the SQL tuning session to a maximum of 256 characters



**Table 139–17 (Cont.) CREATE\_TUNING\_TASK Function Parameters**

Parameter	Description
plan_filter	Plan filter. It is applicable in case there are multiple plans (plan_hash_value) associated with the same statement. This filter allows for selecting one plan (plan_hash_value) only. Possible values are: <ul style="list-style-type: none"> <li>■ LAST_GENERATED: plan with the most recent timestamp</li> <li>■ FIRST_GENERATED: plan with the earliest timestamp, the opposite to LAST_GENERATED</li> <li>■ LAST_LOADED: plan with the most recent first_load_time statistics information</li> <li>■ FIRST_LOADED: plan with the earliest first_load_time statistics information, the opposite to LAST_LOADED</li> <li>■ MAX_ELAPSED_TIME: plan with the maximum elapsed time</li> <li>■ MAX_BUFFER_GETS: plan with the maximum buffer gets</li> <li>■ MAX_DISK_READS: plan with the maximum disk reads</li> <li>■ MAX_DIRECT_WRITES: plan with the maximum direct writes</li> <li>■ MAX_OPTIMIZER_COST: plan with the maximum optimizer cost</li> </ul>
sqlset_owner	Owner of the SQL tuning set, or NULL for the current schema owner
spa_task_name	Name of the SQL Performance Analyzer task whose regressions are to be tuned
spa_task_owner	Owner of specified SQL Performance Analyzer task or NULL for current user
spa_compare_exec	Execution name of Compare Performance trial of SQL Performance Analyzer task. If NULL, we use the most recent execution of the given SQL Performance Analyzer task, of type COMPARE PERFORMANCE

## Return Values

A SQL tuning task name that is unique by user (two different users can give the same name to their advisor tasks).

## Usage Notes

With regard to the form of this subprogram that takes a SQL tuning set, filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

## Examples

```
variable stmt_task VARCHAR2(64);
variable sts_task  VARCHAR2(64);
variable spa_tune_task VARCHAR2(64);
```

### Create Tuning Task with SQL Text format

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
```

```
sql_text => 'select quantity_sold from sales s, times t where s.time_id =
t.time_id and s.time_id = TO_DATE(''24-NOV-00'')');
```

### Create Tuning Task with SQL ID format

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24');

-- tune in limited scope
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24', -
scope => 'LIMITED');

-- only give 10 minutes for tuning statement
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24', -
time_limit => 600);
```

### Create Tuning Task with AWR Snapshot format

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(begin_snap => 1, -
end_snap => 2, sql_id => 'ay1m3ssvtrh24');
```

### Create Tuning Task with SQL Tuning Set format

```
-- First we need to load an STS, then tune it
-- Tune our statements in order by buffer gets, time limit of one hour
-- the default ranking measure is elapsed time.
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
sqlset_name => 'my_workload', -
rank1 => 'BUFFER_GETS', -
time_limit => 3600, -
description => 'tune my workload ordered by buffer gets');
```

### Create Tuning Task with SPA Task format

```
-- Tune the SQLs that were reported as having regressed from the compare
-- performance execution of the SPA task named task_123
EXEC :spa_tune_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(
spa_task_name => 'task_123',
spa_task_owner => 'SCOTT',
spa_compare_exec => 'exec1');
```

## DELETE\_SQLSET Procedure

This procedure deletes a set of SQL statements from a SQL tuning set.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.DELETE_SQLSET (
  sqlset_name  IN  VARCHAR2,
  basic_filter IN  VARCHAR2 := NULL,
  sqlset_owner IN  VARCHAR2 := NULL);
```

### Parameters

**Table 139–18** *DELETE\_SQLSET Procedure Parameters*

Parameter	Description
sqlset_name	The SQL tuning set name
basic_filter	SQL predicate to filter the SQL from the SQL tuning set. This basic filter is used as a where clause on the SQL tuning set content to select a desired subset of SQL from the Tuning Set.
sqlset_owner	The owner of the SQL tuning set, or NULL for current schema owner

### Examples

```
-- Delete all statements in a sql tuning set.
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name => 'my_workload');

-- Delete all statements in a sql tuning set which ran for less than a second
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name => 'my_workload', -
                                basic_filter => 'elapsed_time < 1000000');
```

## DROP\_SQL\_PROFILE Procedure

This procedure drops the named SQL profile from the database.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.DROP_SQL_PROFILE (  
    name          IN VARCHAR2,  
    ignore       IN BOOLEAN := FALSE);
```

### Parameters

**Table 139–19** *DROP\_SQL\_PROFILE Procedure Parameters*

Parameter	Description
name	The (mandatory) name of SQL profile to be dropped. The name is case sensitive.
ignore	Ignores errors due to object not existing

### Usage Notes

Requires the "DROP ANY SQL PROFILE" privilege.

### Examples

```
-- Drop the profile:  
EXEC DBMS_SQLTUNE.DROP_SQL_PROFILE(:pname);
```

## DROP\_SQLSET Procedure

This procedure drops a SQL tuning set if it is not active.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.DROP_SQLSET (  
    sqlset_name    IN  VARCHAR2,  
    sqlset_owner   IN  VARCHAR2 := NULL);
```

### Parameters

**Table 139–20** *DROP\_SQLSET Procedure Parameters*

Parameter	Description
sqlset_name	The SQL tuning set name
sqlset_owner	The owner of the SQL tuning set, or NULL for current schema owner

### Usage Notes

You cannot drop a SQL tuning set when it is referenced by one or more clients.

### Examples

```
-- Drop the sqlset.  
EXEC DBMS_SQLTUNE.DROP_SQLSET ('my_workload');
```

## DROP\_TUNING\_TASK Procedure

This procedure drops a SQL tuning task. The task and all its result data are deleted.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.DROP_TUNING_TASK(  
    task_name          IN VARCHAR2);
```

### Parameters

**Table 139-21** *DROP\_TUNING\_TASK Procedure Parameters*

Parameter	Description
task_name	The name of the tuning task to drop

## EXECUTE\_TUNING\_TASK Function & Procedure

This function and procedure executes a previously created tuning task. Both the function and the procedure run in the context of a new task execution. The difference is that the function version returns that new execution name.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
    task_name          IN VARCHAR2,
    execution_name     IN VARCHAR2          := NULL,
    execution_params   IN dbms_advisor.argList := NULL,
    execution_desc     IN VARCHAR2          := NULL)
RETURN VARCHAR2;

DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
    task_name          IN VARCHAR2,
    execution_name     IN VARCHAR2          := NULL,
    execution_params   IN dbms_advisor.argList := NULL,
    execution_desc     IN VARCHAR2          := NULL);
```

### Parameters

**Table 139–22 EXECUTE\_TUNING\_TASK Function & Procedure Parameters**

Parameter	Description
task_name	Name of the tuning task to execute
execution_name	A name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by function.
execution_params	List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They override the values for the parameters stored in the task (set through the <a href="#">SET_TUNING_TASK_PARAMETER Procedures</a> ).
execution_desc	A 256-length string describing the execution

### Usage Notes

A tuning task can be executed multiples times without having to reset it.

### Examples

```
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK (:stmt_task);
```

## IMPLEMENT\_TUNING\_TASK Function

This function implements a set of SQL profile recommendations made by the SQL Tuning Advisor. Call this subprogram is equivalent to calling the [SCRIPT\\_TUNING\\_TASK Function](#) and then running the script.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.IMPLEMENT_TUNING_TASK(
    task_name      IN VARCHAR2,
    rec_type       IN VARCHAR2 := REC_TYPE_SQL_PROFILES,
    owner_name     IN VARCHAR2 := NULL,
    execution_name IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–23** *IMPLEMENT\_TUNING\_TASK Function Parameters*

Parameter	Description
task_name	Name of the tuning task for which to implement recommendations
rec_type	Filter the types of recommendations to implement. Only 'PROFILES' is supported.
owner_name	Owner of the relevant tuning task or NULL for the current user.
execution_name	name of the task execution to use. If NULL, recommendations from the last task execution are implemented.



## INTERRUPT\_TUNING\_TASK Procedure

This procedure interrupts the currently executing tuning task. The task ends its operations as it would at normal exit so that the user can access the intermediate results.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(  
  task_name          IN VARCHAR2);
```

### Parameters

**Table 139-24** *INTERRUPT\_TUNING\_TASK Procedure Parameters*

Parameter	Description
task_name	Name of the tuning task to interrupt

### Examples

```
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:my_task);
```

## LOAD\_SQLSET Procedure

This procedure populates the SQL tuning set with a set of selected SQL. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.LOAD_SQLSET (
  sqlset_name      IN  VARCHAR2,
  populate_cursor  IN  sqlset_cursor,
  load_option      IN  VARCHAR2 := 'INSERT',
  update_option    IN  VARCHAR2 := 'REPLACE',
  update_condition IN  VARCHAR2 := NULL,
  update_attributes IN VARCHAR2 := NULL,
  ignore_null      IN  BOOLEAN  := TRUE,
  commit_rows      IN  POSITIVE := NULL,
  sqlset_owner     IN  VARCHAR2 := NULL);
```

### Parameters

**Table 139–25** *LOAD\_SQLSET Procedure Parameters*

Parameter	Description
sqlset_name	The SQL tuning set name to populate
populate_cursor	The cursor reference from which to populate
load_option	Specifies how the statements are loaded into the SQL tuning set. The possible values are: <ul style="list-style-type: none"> <li>■ INSERT (default) - add only new statements</li> <li>■ UPDATE - update existing the SQL statements and ignores any new statements</li> <li>■ MERGE - this is a combination of the two other options. This option inserts new statements and updates the information of the existing ones.</li> </ul>
update_option	Specifies how the existing statements are updated. This parameter is considered only if load_option is specified with 'UPDATE'/'MERGE' as an option. The possible values are: <ul style="list-style-type: none"> <li>■ REPLACE (default) - update the statement using the new statistics, bind list, object list, and so on.</li> <li>■ ACCUMULATE - when possible combine attributes (for example, statistics like elapsed_time, and so on) otherwise just replace the old values (for example, module, action, and so on) by the new provided ones. The SQL statement attributes that can be accumulated are: elapsed_time, buffer_gets, direct_writes, disk_reads, row_processed, fetches, executions, end_of_fetch_count, stat_period and active_stat_period.</li> </ul>

**Table 139–25 (Cont.) LOAD\_SQLSET Procedure Parameters**

Parameter	Description
update_condition	Specifies a where clause to execute the update operation. The update is performed only if the specified condition is true. The condition can refer to either the data source or destination. The condition must use the following prefixes to refer to attributes from the source or the destination: <ul style="list-style-type: none"> <li>■ OLD - to refer to statement attributes from the SQL tuning set (destination)</li> <li>■ NEW - to refer to statements attributes from the input statements (source)</li> </ul>
update_attributes	Specifies the list of a SQL statement attributes to update during a merge or update operation. The possible values are: <ul style="list-style-type: none"> <li>■ NULL (default) - the content of the input cursor except the execution context. On other terms, it is equivalent to ALL without execution context like module, action, and so on.</li> <li>■ BASIC - statistics and binds only</li> <li>■ TYPICAL - BASIC + SQL plans (without row source statistics) and without object reference list</li> <li>■ ALL - all attributes including the execution context attributes like module, action, and so on.</li> <li>■ List of comma separated attribute names to update - EXECUTION_CONTEXT, EXECUTION_STATISTICS, BIND_LIST, OBJECT_LIST, SQL_PLAN, SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics)</li> </ul>
ignore_null	If TRUE do not update an attribute if the new value is NULL. That is, do not override with NULL values unless intentional.
commit_rows	If a value is provided, the load commits after each set of that many statements is inserted. If NULL is provided, the load commits only once, at the end of the operation. Providing a value for this argument allows you to monitor the progress of a SQL tuning set load operation in the DBA_/USER_SQLSET views. The STATEMENT_COUNT value increases as new SQL statements are loaded.
sqlset_owner	The owner of the SQL tuning set, or the current schema owner or NULL for current owner

## Exceptions

- This procedure returns an error when `sqlset_name` is invalid, or a corresponding SQL tuning set does not exist, or the `populate_cursor` is incorrect and cannot be executed.
- Exceptions are also raised when invalid filters are provided. Filters can be invalid either because they don't parse (for example, they refer to attributes not in `sqlset_row`), or because they violate the user's privileges.

## Usage Notes

Rows in the input `populate_cursor` must be of type `SQLSET_ROW`.

## Examples

In this example, you create and populate a SQL tuning set with all cursor cache statements with an elapsed time of 5 seconds or more excluding statements that

belong to `SYS` schema (to simulate an application user workload). You select all attributes of the SQL statements and load them in the tuning set using the default mode, which loads only new statements, since the SQL tuning set is empty.

```
-- create the tuning set
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_workload');
-- populate the tuning set from the cursor cache
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
      FROM table(
        DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
          'parsing_schema_name <> ''SYS'' AND elapsed_time > 5000000',
          NULL, NULL, NULL, NULL, 1, NULL,
          'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                           populate_cursor => cur);

END;
/
```

Suppose now you wish to augment this information with what is stored in the workload repository (AWR). You populate the tuning set with `'ACCUMULATE'` as your `update_option` because it is assumed the cursors currently in the cache had aged out since the snapshot was taken.

You omit the `elapsed_time` filter because it is assumed that any statement captured in AWR is important, but still you throw away the `SYS`-parsed cursors to avoid recursive SQL.

```
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
      FROM table(
        DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2,
          'parsing_schema_name <> ''SYS''',
          NULL, NULL, NULL, NULL,
          1,
          NULL,
          'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                           populate_cursor => cur,
                           Using DBMS_SQLTUNE
                           load_option => 'MERGE',
                           update_option => 'ACCUMULATE');

END;
```

The following example is a simple load that only inserts new statements from the workload repository, skipping existing ones (in the SQL tuning set). Note that `'INSERT'` is the default value for the `load_option` argument of the `LOAD_SQLSET` procedure.

```
DECLARE
  cur sys_refcursor;
BEGIN
```

```
OPEN cur FOR
SELECT VALUE(P)
FROM table(DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2)) P;

DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
populate_cursor => cur);
END;
/
```

The next example demonstrates a load with `UPDATE` option. This updates statements that already exist in the SQL tuning set but does not add new ones. By default, old statistics are replaced by their new values.

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
  SELECT VALUE(P)
  FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload',
                           populate_cursor => cur,
                           load_option => 'UPDATE');
END;
/
```

## PACK\_STGTAB\_SQLPROF Procedure

This procedure copies profile data from the SYS. schema into the staging table.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (
  profile_name      IN VARCHAR2 := '%',
  profile_category  IN VARCHAR2 := 'DEFAULT',
  staging_table_name IN VARCHAR2,
  staging_schema_owner IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–26** *PACK\_STGTAB\_SQLPROF Procedure Parameters*

Parameter	Description
profile_name	The name of the profile to pack (% wildcards acceptable, case-sensitive)
profile_category	The category to pack profiles from (% wildcards acceptable, case-sensitive)
staging_table_name	The name of the table to use (case-insensitive unless double quoted). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-insensitive unless double quoted)

### Usage Notes

- This procedure requires ADMINISTER SQL MANAGEMENT OBJECT privilege and INSERT privilege on the staging table.
- Note that this function issues a COMMIT after packing each SQL profile, so if an error is raised mid-execution, clear the staging table by deleting its rows.

### Examples

Put only those profiles in the DEFAULT category into the staging table. This corresponds to all profiles used by default on this system.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (staging_table_name => 'PROFILE_STGTAB');
```

This is another example where you put all profiles into the staging table. Note this moves profiles that are not currently being used by default but are in other categories, such as for testing purposes.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (profile_category => '%', -
                                         staging_table_name => 'PROFILE_STGTAB');
```

## PACK\_STGTAB\_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the SYS schema to a staging table created by the [CREATE\\_STGTAB\\_SQLSET Procedure](#).

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLSET (
  sqlset_name          IN VARCHAR2,
  sqlset_owner         IN VARCHAR2 := NULL,
  staging_table_name   IN VARCHAR2,
  staging_schema_owner IN VARCHAR2 := NULL,
  db_version           IN NUMBER := NULL);
```

### Parameters

**Table 139–27** *PACK\_STGTAB\_SQLSET Procedure Parameters*

Parameter	Description
sqlset_name	The name of the SQL tuning set to pack (% wildcards acceptable, case-sensitive)
sqlset_owner	The category from which to pack SQL tuning sets (% wildcards acceptable, case-sensitive)
staging_table_name	The name of the table to use (case-sensitive)
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-sensitive)
db_version	Database (DB) version determining the format of the staging table. User can also create an older DB version staging table to export STS to an older DB version. One of the following values: <ul style="list-style-type: none"> <li>■ NULL (default) — current DB version</li> <li>■ STS_STGTAB_10_2_VERSION — 10.2 DB version</li> <li>■ STS_STGTAB_11_1_VERSION — 11.1 DB version</li> <li>■ STS_STGTAB_11_2_VERSION — 11.2 DB version</li> </ul>

### Usage Notes

- This procedure can be called several times to move more than one SQL tuning set. Users can then move the populated staging table to another system using any method, such as database link or datapump. Users can then call the [UNPACK\\_STGTAB\\_SQLSET Procedure](#) create the SQL tuning set on the other system.
- Note that this function issues a COMMIT after packing each SQL tuning set, so if an error is raised mid-execution, clear the staging table by deleting its rows.

### Examples

#### Put all SQL tuning sets on the system in the staging table

-- to create a staging table, see the [CREATE\\_STGTAB\\_SQLSET Procedure](#)

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(sqlset_name => '%', -
```

```
sqlset_owner      => '%', -  
staging_table_name => 'STGTAB_SQLSET');
```

**Put only those SQL tuning sets owned by the current user in the staging table**

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(  
  sqlset_name      => '%',  
  staging_table_name => 'STGTAB_SQLSET');
```

**Pack a specific SQL tuning set**

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(  
  sqlset_name      => 'my_workload', -  
  staging_table_name => 'STGTAB_SQLSET');
```

**Pack a second SQL tuning set**

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(  
  sqlset_name      => 'workload_subset', -  
  staging_table_name => 'STGTAB_SQLSET');
```

**Pack the STS my\_workload to a staging table STGTAB\_SQLSET created for Oracle Database 10g Release 2 (10.2)**

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(  
  sqlset_name      => 'workload_subset',  
  staging_table_name => 'STGTAB_SQLSET',  
  db_version       => DBMS_SQLTUNE.STS_STGTAB_10_2_VERSION);
```

**Pack the STS my\_workload to a staging table STGTAB\_SQLSET created for Oracle Database 11g Release 1 (11.1)**

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLSET(  
  sqlset_name      => 'workload_subset',  
  staging_table_name => 'STGTAB_SQLSET',  
  db_version       => DBMS_SQLTUNE.STS_STGTAB_11_1_VERSION);
```



## REMAP\_STGTAB\_SQLPROF Procedure

This procedure allows DBAs to change the profile data values kept in the staging table prior to performing an unpack operation. The procedure can be used to change the category of a profile. It can be used to change the name of a profile if one already exists on the system with the same name.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF (
  old_profile_name      IN VARCHAR2,
  new_profile_name     IN VARCHAR2 := NULL,
  new_profile_category IN VARCHAR2 := NULL,
  staging_table_name    IN VARCHAR2,
  staging_schema_owner  IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–28** REMAP\_STGTAB\_SQLPROF Procedure Parameters

Parameter	Description
old_profile_name	The name of the profile to target for a remap operation (case-sensitive)
new_profile_name	The new name of the profile, or NULL to remain the same (case-sensitive)
new_profile_category	The new category for the profile, or NULL to remain the same (case-sensitive)
staging_table_name	The name of the table on which to perform the remap operation (case-sensitive). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-sensitive)

### Usage Notes

Using this procedure requires the UPDATE privilege on the staging table.

### Examples

Change the name of a profile before we unpack, to avoid conflicts

```
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(old_profile_name => :pname,      -
                                         new_profile_name => 'IMP' || :pname, -
                                         staging_table_name => 'PROFILE_STGTAB');
```

Change the SQL profile in the staging table to be 'TEST' category before we import it. This way users can test the profile on the new system before it is active.

```
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(old_profile_name => :pname,      -
                                         new_profile_category => 'TEST',    -
                                         staging_table_name => 'PROFILE_STGTAB');
```

## REMAP\_STGTAB\_SQLSET Procedure

This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLSET (
  old_sqlset_name      IN VARCHAR2,
  old_sqlset_owner     IN VARCHAR2 := NULL,
  new_sqlset_name      IN VARCHAR2 := NULL,
  new_sqlset_owner     IN VARCHAR2 := NULL,
  staging_table_name   IN VARCHAR2,
  tagging_schema_owner IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–29** REMAP\_STGTAB\_SQLSET Procedure Parameters

Parameter	Description
old_sqlset_name	The name of the tuning set to target for a remap operation. Wildcards are not supported.
old_sqlset_owner	The new name of the tuning set owner to target for a remap operation. NULL for current schema owner
new_sqlset_name	The new name for the tuning set, or NULL to keep the same tuning set name.
new_sqlset_owner	The new owner for the tuning set, or NULL to remain the same owner name.
staging_table_name	The name of the table on which to perform the remap operation (case-sensitive)
staging_schema_owner	The name of staging table owner, or NULL for current schema owner (case-sensitive)

### Usage Notes

You can call this procedure multiple times to remap more than one tuning set name or owner. Note that this procedure only handles one tuning set per call.

### Examples

```
-- Change the name of an STS in the staging table before we unpack it.
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(old_sqlset_name => 'my_workload', -
                                       old_sqlset_owner => 'SH', -
                                       new_sqlset_name => 'imp_workload', -
                                       staging_table_name => 'STGTAB_SQLSET');
```

```
-- Change the owner of an STS in the staging table before we unpack it.
EXEC DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(old_sqlset_name => 'imp_workload', -
                                       old_sqlset_owner => 'SH', -
                                       new_sqlset_owner => 'SYS', -
                                       staging_table_name => 'STGTAB_SQLSET');
```

## REMOVE\_SQLSET\_REFERENCE Procedure

This procedure deactivates a SQL tuning set to indicate it is no longer used by the client.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE (
  sqlset_name  IN  VARCHAR2,
  reference_id IN  NUMBER);
```

### Parameters

**Table 139–30 REMOVE\_SQLSET\_REFERENCE Procedure Parameters**

Parameter	Description
sqlset_name	The SQL tuning set name
reference_id	The identifier of the reference to remove

### Examples

You can remove references on a given SQL tuning set when you finish using it and want to make it writable again.

```
EXEC DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE( -
      sqlset_name => 'my_workload', -
      reference_id => :rid);
```

Use views USER/DBA\_SQLSET\_REFERENCES to find all references on a given SQL tuning set.

## REPORT\_AUTO\_TUNING\_TASK Function

This function displays a report from the automatic tuning task. This function reports on a range of task executions, whereas the [REPORT\\_TUNING\\_TASK Function](#) reports on a single execution. Note that this function is deprecated in Oracle Database 11g Release 2 (11.2) in favor of `DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK`.

### See Also:

- [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group
- [REPORT\\_AUTO\\_TUNING\\_TASK Function](#) on page 28-7

### Syntax

```
DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK (
  begin_exec      IN VARCHAR2 := NULL,
  end_exec        IN VARCHAR2 := NULL,
  type            IN VARCHAR2 := TYPE_TEXT,
  level           IN VARCHAR2 := LEVEL_TYPICAL,
  section         IN VARCHAR2 := SECTION_ALL,
  object_id       IN NUMBER   := NULL,
  result_limit    IN NUMBER   := NULL)
RETURN CLOB;
```

### Parameters

**Table 139-31** *REPORT\_AUTO\_TUNING\_TASK Function Parameters*

Parameter	Description
<code>begin_exec</code>	Name of execution from which to begin the report. NULL retrieves a report on the most recent run
<code>end_exec</code>	Name of execution at which to end the report. NULL retrieves a report on the most recent run.
<code>type</code>	Type of the report to produce. Possible values are <code>TYPE_TEXT</code> which produces a text report
<code>level</code>	Level of detail in the report: <ul style="list-style-type: none"> <li>■ <code>LEVEL_BASIC</code>: simple version of the report. Just show info about the actions taken by the advisor.</li> <li>■ <code>LEVEL_TYPICAL</code>: show information about every statement analyzed, including requests not implemented.</li> <li>■ <code>LEVEL_ALL</code>: highly detailed report level, also provides annotations about statements skipped over.</li> </ul>
<code>section</code>	Optionally limit the report to a single section ( <code>ALL</code> for all sections): <ul style="list-style-type: none"> <li>■ <code>SECTION_SUMMARY</code> - summary information</li> <li>■ <code>SECTION_FINDINGS</code> - tuning findings</li> <li>■ <code>SECTION_PLAN</code> - explain plans</li> <li>■ <code>SECTION_INFORMATION</code> - general information</li> <li>■ <code>SECTION_ERROR</code> - statements with errors</li> <li>■ <code>SECTION_ALL</code> - all statements</li> </ul>

**Table 139-31 (Cont.) REPORT\_AUTO\_TUNING\_TASK Function Parameters**

<b>Parameter</b>	<b>Description</b>
<code>object_id</code>	Advisor framework object id that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.
<code>result_limit</code>	Maximum number of SQL statements to show in the report

**Return Values**

A CLOB containing the desired report.

## REPORT\_SQL\_DETAIL Function

This function builds a report for a specific SQLID. For each SQLID it gives various statistics and details as obtained from the V\$ views and AWR.

**See Also:** [SQL Performance Reporting Subprograms](#) on page 139-15 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REPORT_SQL_DETAIL (
  sql_id          IN VARCHAR2  DEFAULT NULL,
  sql_plan_hash_value IN NUMBER  DEFAULT NULL,
  start_time      IN DATE      DEFAULT NULL,
  duration        IN NUMBER    DEFAULT NULL,
  inst_id         IN NUMBER    DEFAULT NULL,
  dbid            IN NUMBER    DEFAULT NULL,
  event_detail    IN VARCHAR2  DEFAULT 'YES',
  bucket_max_count IN NUMBER   DEFAULT 128,
  bucket_interval IN NUMBER    DEFAULT NULL,
  top_n           IN NUMBER    DEFAULT 10,
  report_level    IN VARCHAR2  DEFAULT 'TYPICAL',
  type            IN VARCHAR2  DEFAULT 'ACTIVE')
RETURN CLOB;
```

### Parameters

**Table 139-32** REPORT\_SQL\_DETAIL Function Parameters

Parameter	Description
sql_id	SQLID for which monitoring information should be displayed. If NULL (the default), display statistics for the SQLID of the last SQL statement executed in the current session.
sql_plan_hash_value	Displays SQL statistics and details for a specific plan_hash_value. If NULL (default), displays statistics and details for all plans of the SQL_ID.
start_time	If specified, shows SQL activity (from gv\$ACTIVE_SESSION_HISTORY) starting at this time. On Oracle RAC, the minimum start_time is the earliest sample_time of the in-memory ASH buffers across all instances. If NULL (default), one hour before the current time.
duration	Duration of activity in seconds for the report. If NULL (default) uses a value of 1 hour.
inst_id	Target instance to get SQL details from. If NULL, uses data from all instances. If 0 or -1, uses current instance.
dbid	DBID from which to get SQL details. If NULL, uses current DBID.
event_detail	When set to 'NO', the activity is aggregated by wait_class only. Use 'YES' (the default) to aggregate by (wait_class, event_name).
bucket_max_count	If specified, this should be the maximum number of histogram buckets created in the report. If not specified, a value of 128 is used.

**Table 139-32 (Cont.) REPORT\_SQL\_DETAIL Function Parameters**

Parameter	Description
bucket_interval	If specified, this represents the exact time interval in seconds, of all histogram buckets. If specified, bucket_max_count is ignored.
top_n	Controls the number of entries to display per dimension in the top dimensions section. If not specified, a default value of 10 is used.
report_level	<p>Level of detail for the report, either 'BASIC', 'TYPICAL' or 'ALL'. Default assumes 'TYPICAL'. Their meanings are explained below.</p> <p>In addition, individual report sections can also be enabled or disabled by using a +/- <i>section_name</i>. Several sections are defined:</p> <ul style="list-style-type: none"> <li>▪ 'TOP' - Show top values for the ASH dimensions for a SQL statement; ON by default</li> <li>▪ 'SPM' - Show existing plan baselines for a SQL statement; OFF by default</li> <li>▪ 'MISMATCH' - Show reasons for creating new child cursors (sharing criteria violations); OFF by default.</li> <li>▪ 'STATS' - Show SQL execution statistics per plan from GV\$SQLAREA_PLAN_HASH; ON by default</li> <li>▪ 'ACTIVITY' - Show top activity from ASH for each plan of a SQL statement; ON by default</li> <li>▪ 'ACTIVITY_ALL' - Show top activity from ASH for each line of the plan for a SQL statement; OFF by default</li> <li>▪ 'HISTOGRAM' - Show activity histogram for each plan of a SQL statement (plan timeline histogram); ON by default</li> <li>▪ 'SESSIONS' - Show activity for top sessions for each plan of a SQL statement; OFF by default</li> <li>▪ 'MONITOR' - Show show one monitored SQL execution per execution plan; ON by default</li> <li>▪ 'XPLAN' - Show execution plans; ON by default</li> <li>▪ 'BINDS' - show captured bind data; ON by default</li> </ul> <p>In addition, SQL text can be specified at different levels:</p> <ul style="list-style-type: none"> <li>▪ -SQL_TEXT - No SQL text in report</li> <li>▪ +SQL_TEXT - OK with partial SQL text up to the first 2000 chars as stored in GV\$SQL_MONITOR</li> <li>▪ -SQL_FULLTEXT - No full SQL text (+SQL_TEXT)</li> <li>▪ +SQL_FULLTEXT - Show full SQL text (default value)</li> </ul> <p>The meanings of the three top-level report levels are:</p> <ul style="list-style-type: none"> <li>▪ NONE - minimum possible</li> <li>▪ BASIC - SQL_TEXT+STATS+ACTIVITY+HISTOGRAM</li> <li>▪ TYPICAL - SQL_FULLTEXT+TOP+STATS+ACTIVITY+HISTOGRAM+XPLAN+MONITOR</li> <li>▪ ALL - everything</li> </ul> <p>Only one of these 4 levels can be specified and, if it is, it has to be at the start of the REPORT_LEVEL string</p>
type	Report format: 'ACTIVE' by default. Can also be 'XML' (see Usage Notes).

## Return Values

A CLOB containing the desired report.

## Usage Notes

- ACTIVE reports have a rich, interactive user interface similar to Enterprise Manager while not requiring any EM installation. The report file built is in HTML format, so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.
- The invoker needs the SELECT privilege on the following views:
  - V\$SESSION
  - DBA\_ADVISOR\_FINDINGS
  - V\$DATABASE
  - GV\$ASH\_INFO
  - GV\$ACTIVE\_SESSION\_HISTORY
  - GV\$SQLAREA\_PLAN\_HASH
  - GV\$SQL
  - DBA\_HIST\_SNAPSHOT
  - DBA\_HIST\_WR\_CONTROL
  - DBA\_HIST\_ACTIVE\_SESS\_HISTORY
  - DBA\_HIST\_SQLSTAT
  - DBA\_HIST\_SQL\_BIND\_METADATA
  - DBA\_HIST\_SQLTEXT
  - DBA\_SQL\_PLAN\_BASELINES
  - DBA\_SQL\_PROFILES
  - DBA\_ADVISOR\_TASKS
  - DBA\_SERVICES
  - DBA\_USERS
  - DBA\_OBJECTS
  - DBA\_PROCEDURES
- The invoker needs the EXECUTE privilege on the [DBMS\\_XPLAN](#) package.
- Finally, the invoker requires all privileges required by `DBMS_SQLTUNE.REPORT_SQL_MONITOR` and `DBMS_SQLTUNE.REPORT_SQL_MONITOR_LIST` as it calls these functions.



## REPORT\_SQL\_MONITOR Function

This function builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution.

**See Also:** [Real-time SQL Monitoring Subprograms](#) on page 139-14 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REPORT_SQL_MONITOR(
  sql_id              IN VARCHAR2  DEFAULT NULL,
  session_id         IN NUMBER    DEFAULT NULL,
  session_serial     IN NUMBER    DEFAULT NULL,
  sql_exec_start     IN DATE      DEFAULT NULL,
  sql_exec_id        IN NUMBER    DEFAULT NULL,
  inst_id            IN NUMBER    DEFAULT NULL,
  start_time_filter  IN DATE      DEFAULT NULL,
  end_time_filter    IN DATE      DEFAULT NULL,
  instance_id_filter IN NUMBER    DEFAULT NULL,
  parallel_filter    IN VARCHAR2  DEFAULT NULL,
  plan_line_filter   IN NUMBER    DEFAULT NULL,
  event_detail       IN VARCHAR2  DEFAULT 'YES',
  bucket_max_count  IN NUMBER    DEFAULT 128,
  bucket_interval   IN NUMBER    DEFAULT NULL,
  base_path          IN VARCHAR2  DEFAULT NULL,
  last_refresh_time  IN DATE      DEFAULT NULL,
  report_level       IN VARCHAR2  DEFAULT 'TYPICAL',
  type              IN VARCHAR2  DEFAULT 'TEXT',
  sql_plan_hash_value IN NUMBER    DEFAULT NULL)
RETURN CLOB;
```

### Parameters

**Table 139–33** *REPORT\_SQL\_MONITOR Function Parameters*

Parameter	Description
sql_id	SQL_ID for which monitoring information should be displayed. Use NULL (the default) to report on the last statement monitored by Oracle.
session_id	If not NULL, this parameters targets only the sub-set of statements executed by the specified session. Default is NULL. Use USERENV('SID') for current session.
session_serial	In addition to the session_id parameter, one can also specify its session serial to ensure that the desired session incarnation is targeted. This parameter is ignored when session_id is NULL.
sql_exec_start	This parameter, along with sql_exec_id, is only applicable when sql_id is also specified. Jointly, they can be used to display monitoring information associated to any execution of the statement identified by sql_id, assuming that this statement was monitored. When NULL (the default), the last monitored execution of SQL sql_id is shown.

**Table 139-33 (Cont.) REPORT\_SQL\_MONITOR Function Parameters**

Parameter	Description
<code>sql_exec_id</code>	This parameter, along with <code>sql_exec_start</code> , is only applicable when <code>sql_id</code> is also specified. Jointly, they can be used to display monitoring information associated to any execution of the statement identified by <code>sql_id</code> , assuming that this statement was monitored. When <code>NULL</code> (the default), the last monitored execution of SQL <code>sql_id</code> is shown.
<code>inst_id</code>	Only considers statements started on the specified instance. Use <code>-1</code> to target the login instance. <code>NULL</code> (default) targets all instances.
<code>start_time_filter</code>	If not <code>NULL</code> , the report considers only the activity (from <code>GV\$ACTIVE_SESSION_HISTORY</code> ) recorded after the specified date. If <code>NULL</code> , the reported activity starts when the execution of the targeted SQL statement has started.
<code>end_time_filter</code>	If not <code>NULL</code> , the report shows only the activity (from <code>GV\$ACTIVE_SESSION_HISTORY</code> ) collected before the date <code>end_time_filter</code> . If <code>NULL</code> , the reported activity ends when the targeted SQL statement execution has ended or is the current time if the statement is still executing.
<code>instance_id_filter</code>	Only applies when the execution runs parallel across multiple Oracle Real Application Cluster (Oracle RAC) instances. This parameter allows to only report the activity of the specified instance. Use a <code>NULL</code> value (the default) to include the activity on all instances where the parallel query was executed.
<code>parallel_filter</code>	Applies only to parallel execution and allows reporting the activity of only a subset of the processes involved in the parallel execution (Query Coordinator and/or Parallel eXecution servers). The value of this parameter can be: <ul style="list-style-type: none"> <li>▪ <code>NULL</code> to target all processes</li> <li>▪ <code>[qc] [servers (&lt;svr_grp&gt;[ , ] &lt;svr_set&gt;[ , ] &lt;srv_num&gt;)]</code>: 'qc' stands for query coordinator and <code>servers()</code> stipulate which PX servers to consider.</li> </ul> <p>The following examples show how to target a subset of the parallel processes:</p> <ul style="list-style-type: none"> <li>▪ <code>qc</code>: targets only the query coordinator</li> <li>▪ <code>servers(1)</code>: targets all parallel execution servers in group number 1. Note that statement running parallel have one main server group (group number 1) plus one additional group for each nested sub-query running parallel.</li> <li>▪ <code>servers(, 2)</code>: targets all parallel execution servers from any group but only running in set 1 of each group (each group has at most two set of parallel execution servers)</li> <li>▪ <code>servers(1, 1)</code>: consider only group 1, set 1</li> <li>▪ <code>servers(1, 2, 4)</code>: consider only group 1, set 2, server number 4. This reports for a single parallel server process</li> <li>▪ <code>qc servers(1, 2, 4)</code>: same as above by also including the query coordinator</li> </ul>
<code>event_detail</code>	When value is 'YES' (the default), reported activity from <code>GV\$ACTIVE_SESSION_HISTORY</code> is aggregated by ( <code>wait_class</code> , <code>event_name</code> ). Use 'NO' to only aggregate by <code>wait_class</code> .
<code>bucket_max_count</code>	If specified, this should be the maximum number of histogram buckets created in the report

**Table 139-33 (Cont.) REPORT\_SQL\_MONITOR Function Parameters**

Parameter	Description
bucket_interval	If specified, this represents the exact time interval in seconds, of all histogram buckets. If specified, bucket_max_count is ignored.
base_path	URL path for flex HTML resources since flex HTML format is required to access external files (java scripts and the flash SWF file itself)
last_refresh_time	If not NULL (default is NULL), the time when the report was last retrieved (see SYSDATE attribute of the report tag). Use this option to display the report of a running query, and when the report is refreshed on a regular basis. This optimizes the size of the report since only the new or changed information is returned. In particular, the following are optimized: <ul style="list-style-type: none"> <li>■ SQL text is not returned when this option is specified</li> <li>■ activity histogram starts at the bucket that intersect at that time. The entire content of the bucket is returned, even if last_refresh_time is after the start of that bucket</li> </ul>
report_level	Level of detail for the report, either 'NONE', 'BASIC', 'TYPICAL' or 'ALL'. Default assumes 'TYPICAL'. Their meanings are explained below.  In addition, individual report sections can also be enabled or disabled by using a +/- <i>section_name</i> . Several sections are defined: <ul style="list-style-type: none"> <li>■ 'XPLAN' - Show explain plan; ON by default</li> <li>■ 'PLAN' - Show plan monitoring statistics; ON by default</li> <li>■ 'SESSIONS' - Show session details. Applies only to parallel queries; ON by default</li> <li>■ 'INSTANCE' - Show instance details. Applies only to parallel and cross instance; ON by default</li> <li>■ 'PARALLEL' - An umbrella parameter for specifying sessions+instance details</li> <li>■ 'ACTIVITY' - Show activity summary at global level, plan line level and session or instance level (if applicable); ON by default</li> <li>■ 'BINDS' - Show bind information when available; ON by default</li> <li>■ 'METRICS' - Show metric data (CPU, IOs, ...) over time; ON by default</li> <li>■ 'ACTIVITY_HISTOGRAM' - Show an histogram of the overall query activity; ON by default</li> <li>■ 'PLAN_HISTOGRAM' - Show activity histogram at plan line level; OFF by default</li> <li>■ 'OTHER' - Other info; ON by default</li> </ul> In addition, SQL text can be specified at different levels: <ul style="list-style-type: none"> <li>■ SQL_TEXT - No SQL text in report</li> <li>■ +SQL_TEXT - OK with partial SQL text up to the first 2000 chars as stored in GV\$SQL_MONITOR</li> <li>■ -SQL_FULLTEXT - No full SQL text (+SQL_TEXT)</li> <li>■ +SQL_FULLTEXT - Show full SQL text (default value)</li> </ul>

**Table 139-33 (Cont.) REPORT\_SQL\_MONITOR Function Parameters**

Parameter	Description
report_level (contd.)	<p>The meanings of the three top-level report levels are:</p> <ul style="list-style-type: none"> <li>■ NONE - minimum possible</li> <li>■ +BASIC - SQL_TEXT-PLAN-XPLAN-SESSIONS-INSTANCE-ACTIVITY_HISTOGRAM-PLAN_HISTOGRAM-METRICS</li> <li>■ TYPICAL - everything but PLAN_HISTOGRAM</li> <li>■ ALL - everything</li> </ul> <p>Only one of these 4 levels can be specified and, if it is, it has to be at the start of the REPORT_LEVEL string</p>
type	Report format, 'TEXT' by default. Can be 'TEXT', 'HTML', 'XML' or 'ACTIVE' (see Usage Notes).
sql_plan_hash_value	Target only those SQL executions with the specified plan_hash_value. Default is NULL.

## Return Values

A CLOB containing the desired report.

## Usage Notes

- The target SQL statement for this report can be:
  - The last SQL monitored by Oracle (this is the default behavior, so there is no need to specify any parameter)
  - The last SQL executed by a specific session and monitored by Oracle. The session is identified by its session id and optionally its serial number. For example, use `session_id => USERENV ('SID')` for the current session or `session_id=>20, session_serial=>103` for session ID 20, serial number 103.
  - The last execution of a specific statement identified by its `sql_id`.
  - A specific execution of a SQL statement identified by its execution key (`sql_id, sql_exec_start` and `sql_exec_id`).
- This report produces performance data exposed by several fixed views, listed below. For this reason, the invoker of the report function must have privilege to select data from these fixed views (such as the `SELECT_CATALOG` role).
  - `GV$SQL_MONITOR`
  - `GV$SQL_PLAN_MONITOR`
  - `GV$SQL_PLAN`
  - `GV$ACTIVE_SESSION_HISTORY`
  - `GV$SESSION_LONGOPS`
  - `GV$SQL`
- The `bucket_max_count` and `bucket_interval` parameters control the activity histogram. By default, the maximum number of buckets is set to 128 and the RDBMS derives the `bucket_interval` based on this. The `bucket_interval` (value is in seconds) is computed such that it is the smallest possible power of 2 value (starting at 1s) without exceeding the maximum number of buckets. For example, if the query has executed for 600s, the RDBMS selects a `bucket_interval` of 8s

(a power of two) given that  $600/8 = 74$  which is less than 128 buckets maximum. Smaller than 8s would be 4s which would lead to more buckets than the 128 maximum. If `bucket_interval` is specified, the RDBMS uses that value instead of deriving it from `bucket_max_count`.

- ACTIVE reports have a rich, interactive user interface similar to Enterprise Manager while not requiring any EM installation. The report file built is in HTML format, so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.

**See Also:** *Oracle Database Performance Tuning Guide* for more information about SQL real-time monitoring.

## REPORT\_SQL\_MONITOR\_LIST Function

This function builds a report for all or a sub-set of statements monitored by Oracle. For each statement, the subprogram gives key information and associated global statistics.

Use the [REPORT\\_SQL\\_MONITOR\\_LIST Function](#) to get detail monitoring information for a single SQL statement

**See Also:** [SQL Performance Reporting Subprograms](#) on page 139-15 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REPORT_SQL_MONITOR_LIST (
  sql_id           IN VARCHAR2  DEFAULT NULL,
  session_id      IN NUMBER     DEFAULT NULL,
  session_serial  IN NUMBER     DEFAULT NULL,
  inst_id         IN NUMBER     DEFAULT NULL,
  active_since_date IN DATE      DEFAULT NULL,
  active_since_sec IN NUMBER     DEFAULT NULL,
  last_refresh_time IN DATE      DEFAULT NULL,
  report_level    IN VARCHAR2   DEFAULT 'TYPICAL',
  auto_refresh    IN NUMBER     DEFAULT NULL,
  base_path       IN VARCHAR2   DEFAULT NULL,
  type            IN VARCHAR2   DEFAULT 'TEXT')
RETURN CLOB;
```

### Parameters

**Table 139-34** REPORT\_SQL\_MONITOR\_LIST Function Parameters

Parameter	Description
sql_id	SQL_ID for which monitoring information should be displayed. Use NULL (the default) to report on the last statement monitored by Oracle.
session_id	If not NULL, this parameter targets only the sub-set of statements executed by the specified session. Default is NULL. Use -1 or USERENV('SID') for current session.
session_serial	In addition to the session_id parameter, you can also specify its session serial to ensure that the desired session incarnation is targeted. This parameter is ignored when session_id is NULL.
inst_id	Only considers statements started on the specified instance. Use -1 to target the login instance. NULL (default) targets all instances.
active_since_date	If not NULL (default), returns only monitored statements active since the specified time. This includes all statements that are still executing along with all statements that have completed their execution after the specified date and time.
active_since_sec	Same as active_since_date but with the date specified relative to the current sysdate minus a specified number of seconds. For example, use 3600 to apply a limit of 1 hour.

**Table 139-34 (Cont.) REPORT\_SQL\_MONITOR\_LIST Function Parameters**

Parameter	Description
last_refresh_time	If not NULL (default), the date and time when the list report was last retrieved. This optimizes the case where an application shows the list and refreshes the report on a regular basis (such as once every 5 seconds). In this case, the report shows detail about the execution of monitored queries that active since the specified last_refresh_time. For other queries, the report returns the execution key (sql_id, sql_exec_start, sql_exec_id). For queries with a first refresh time after the specified date, only the SQL execution key and statistics are returned.
report_level	Level of detail for the report. The level can be either: <ul style="list-style-type: none"> <li>■ BASIC - SQL text up to 200 characters</li> <li>■ TYPICAL - include full SQL text assuming that cursor has not aged out, in which case the SQL text is included up to 2000 characters</li> <li>■ ALL - currently the same as TYPICAL</li> </ul>
auto_refresh	Currently non-operational, reserved for future use
base_path	URL path for flex HTML resources since flex HTML format is required to access external files (java scripts and the flash SWF file itself)
type	Report format, 'TEXT' by default. Can be 'TEXT', 'HTML' or 'XML'.

## Return Values

A report (XML, text, HTML) for the list of SQL statements that have been monitored.

## Usage Notes

The user tuning this function needs to have privilege to access the following fixed views: GV\$SQL\_MONITOR and GV\$SQL.

**See Also:** *Oracle Database Performance Tuning Guide* for more information about SQL real-time monitoring.

## REPORT\_TUNING\_TASK Function

This procedure displays the results of a tuning task.

**See Also:** [SQL Performance Reporting Subprograms](#) on page 139-15 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.REPORT_TUNING_TASK (
  task_name      IN   VARCHAR2,
  type           IN   VARCHAR2  := 'TEXT',
  level         IN   VARCHAR2  := 'TYPICAL',
  section       IN   VARCHAR2  := ALL,
  object_id     IN   NUMBER     := NULL,
  result_limit  IN   NUMBER     := NULL,
  owner_name    IN   VARCHAR2  := NULL,
  execution_name IN  VARCHAR2  := NULL)
RETURN CLOB;
```

### Parameters

**Table 139–35** REPORT\_TUNING\_TASK Function Parameters

Parameter	Description
task_name	Name of the tuning task to report
type	Type of the report to produce. Possible values are TEXT which produces a text report.
level	Level of detail in the report: <ul style="list-style-type: none"> <li>■ BASIC: simple version of the report. Just show info about the actions taken by the advisor.</li> <li>■ TYPICAL: show information about every statement analyzed, including requests not implemented.</li> <li>■ ALL: highly detailed report level, also provides annotations about statements skipped over.</li> </ul>
section	Optionally limit the report to a single section (ALL for all sections): <ul style="list-style-type: none"> <li>■ SUMMARY - summary information</li> <li>■ FINDINGS - tuning findings</li> <li>■ PLAN - explain plans</li> <li>■ INFORMATION - general information</li> <li>■ ERROR - statements with errors</li> <li>■ ALL - all statements</li> </ul>
object_id	Advisor framework object id that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.
result_limit	Maximum number of SQL statements to show in the report
owner_name	Owner of the relevant tuning task. Defaults to the current schema owner.
execution_name	Name of the task execution to use. If NULL, the report is generated for the last task execution.



## Return Values

A CLOB containing the desired report.

## Examples

```
-- Get the whole report for the single statement case.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:stmt_task) from dual;

-- Show me the summary for the sts case.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL', 'SUMMARY')
FROM DUAL;

-- Show me the findings for the statement I'm interested in.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL', 'FINDINGS',
5) from dual;
```

## RESET\_TUNING\_TASK Procedure

This procedure is called on a tuning task that is not currently executing to prepare it for re-execution.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.RESET_TUNING_TASK(  
  task_name          IN VARCHAR2);
```

### Parameters

**Table 139–36** RESET\_TUNING\_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to reset

### Examples

```
-- reset and re-execute a task  
EXEC DBMS_SQLTUNE.RESET_TUNING_TASK(:sts_task);  
  
-- re-execute the task  
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);
```

## RESUME\_TUNING\_TASK Procedure

This procedure resumes a previously interrupted task that was created to process a SQL tuning set.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.RESUME_TUNING_TASK(
  task_name          IN VARCHAR2,
  basic_filter       IN VARCHAR2 := NULL);
```

### Parameters

**Table 139–37 RESUME\_TUNING\_TASK Procedure Parameters**

Parameter	Description
task_name	The name of the tuning task to resume
basic_filter	A SQL predicate to filter the SQL from the SQL tuning set. Note that this filter is applied in conjunction with the basic filter (i.e., parameter <code>basic_filter</code> ) when calling <a href="#">CREATE_TUNING_TASK Functions</a> .

### Usage Notes

Resuming a single SQL tuning task (a task that was created to tune a single SQL statement as compared to a SQL tuning set) is not supported.

### Examples

```
-- Interrupt the task
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:conc_task);

-- Once a task is interrupted, we can elect to reset it, resume it, or check
-- out its results and then decide. For this example we will just resume.

EXEC DBMS_SQLTUNE.RESUME_TUNING_TASK(:conc_task);
```

## SCRIPT\_TUNING\_TASK Function

This function creates a SQL\*PLUS script which can then be executed to implement a set of Advisor recommendations.

**See Also:** [SQL Tuning Advisor Subprograms](#) on page 139-11 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SCRIPT_TUNING_TASK(
  task_name          IN VARCHAR2,
  rec_type           IN VARCHAR2  := REC_TYPE_ALL,
  object_id          IN NUMBER    := NULL,
  result_limit       IN NUMBER    := NULL,
  owner_name         IN VARCHAR2  := NULL,
  execution_name     IN VARCHAR2  := NULL)
RETURN CLOB;
```

### Parameters

**Table 139–38** *SCRIPT\_TUNING\_TASK Function Parameters*

Parameter	Description
task_name	Name of the tuning task for which to apply a script
rec_type	Filter the script by types of recommendations to include. Any subset of the following separated by commas: or 'ALL: 'PROFILES' 'STATISTICS' 'INDEXES'. For example, a script with profiles and statistics: 'PROFILES, STATISTICS'
object_id	Optionally filters by a single object ID
result_limit	Optionally shows commands for only top N SQL (ordered by object_id and ignored if an object_id is also specified)
owner_name	Owner of the relevant tuning task. Defaults to the current schema owner
excution_name	Name of the task execution to use. If NULL, the script is generated for the last task execution.

### Return Values

Returns a script in the form of a CLOB.

### Usage Notes

- Once the script is returned, it should then be checked by the DBA and executed.
- Wrap with a call to `DBMS_ADVISOR.CREATE_FILE` to put it into a file.

### Examples

```
SET LINESIZE 140

-- Get a script for all actions recommended by the task.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task) FROM DUAL;

-- Get a script of just the sql profiles we should create.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'PROFILES') FROM DUAL;
```

```
-- get a script of just stale / missing stats
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'STATISTICS') FROM DUAL;

-- Get a script with recommendations about just one SQL statement when we have
-- tuned an entire STS.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:sts_task, 'ALL', 5) FROM DUAL;
```

## SELECT\_CURSOR\_CACHE Function

This function collects SQL statements from the SQL Cursor Cache.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SELECT_CURSOR_CACHE (
  basic_filter      IN  VARCHAR2 := NULL,
  object_filter    IN  VARCHAR2 := NULL,
  ranking_measure1 IN  VARCHAR2 := NULL,
  ranking_measure2 IN  VARCHAR2 := NULL,
  ranking_measure3 IN  VARCHAR2 := NULL,
  result_percentage IN NUMBER := 1,
  result_limit     IN  NUMBER := NULL,
  attribute_list   IN  VARCHAR2 := NULL,
  recursive_sql    IN  VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;
```

### Parameters

**Table 139–39** *SELECT\_CURSOR\_CACHE Function Parameters*

Parameter	Description
<code>basic_filter</code>	The SQL predicate to filter the SQL from the cursor cache defined on attributes of the <code>SQLSET_ROW</code> . If <code>basic_filter</code> is not set by the caller, the subprogram captures only statements of the type <code>CREATE TABLE</code> , <code>INSERT</code> , <code>SELECT</code> , <code>UPDATE</code> , <code>DELETE</code> , and <code>MERGE</code> .
<code>object_filter</code>	Currently not supported.
<code>ranking_measure (n)</code>	An order-by clause on the selected SQL.
<code>result_percentage</code>	A filter which picks the top N% according to the ranking measure given. Note that this applies only if one ranking measure is given.
<code>result_limit</code>	The top L(imit) SQL from the (filtered) source ranked by the ranking measure
<code>attribute_list</code>	List of SQL statement attributes to return in the result. The possible values are: <ul style="list-style-type: none"> <li>■ <code>TYPICAL - BASIC + SQL plan</code> (without row source statistics) and without object reference list (default)</li> <li>■ <code>BASIC</code> - all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result.</li> <li>■ <code>ALL</code> - return all attributes</li> <li>■ Comma separated list of attribute names this allows to return only a subset of SQL attributes: <code>EXECUTION_STATISTICS</code>, <code>BIND_LIST</code>, <code>OBJECT_LIST</code>, <code>SQL_PLAN</code>, <code>SQL_PLAN_STATISTICS</code>: similar to <code>SQL_PLAN + row source statistics</code></li> </ul>
<code>recursive_sql</code>	Filter that includes recursive SQL in the SQL tuning set ( <code>HAS_RECURSIVE_SQL</code> ) or excludes it ( <code>NO_RECURSIVE_SQL</code> ).

## Return Values

This function returns a one SQLSET\_ROW per SQL\_ID or PLAN\_HASH\_VALUE pair found in each data source.

## Usage Notes

- Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.
- Users need privileges on the cursor cache views.

## Examples

```
-- Get sql ids and sql text for statements with 500 buffer gets.
SELECT sql_id, sql_text
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('buffer_gets > 500'))
ORDER BY sql_id;

-- Get all the information we have about a particular statement.
SELECT *
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('sql_id = ''4rm4183czbs7j''));

-- Notice that some statements can have multiple plans. The output of the
-- SELECT_XXX table functions is unique by (sql_id, plan_hash_value). This is
-- because a data source can store multiple plans per sql statement.
SELECT sql_id, plan_hash_value
FROM table(dbms_sqltune.select_cursor_cache('sql_id = ''ay1m3ssvtrh24''))
ORDER BY sql_id, plan_hash_value;

-- PL/SQL examples: load_sqlset is called after opening a cursor, along the
-- lines given below

-- Select all statements in the cursor cache.
DECLARE
    cur sys_refcursor;
BEGIN
    OPEN cur FOR
        SELECT value(P)
        FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;

    -- Process each statement (or pass cursor to load_sqlset).

    CLOSE cur;
END;
/

-- Look for statements not parsed by SYS.
DECLARE
    cur sys_refcursor;
BEGIN
    OPEN cur for
        SELECT VALUE(P)
        FROM table(
            DBMS_SQLTUNE.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS''')) P;

    -- Process each statement (or pass cursor to load_sqlset).

    CLOSE cur;
```

```
end;
/

-- All statements from a particular module/action.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
        'module = 'MY_APPLICATION' and action = 'MY_ACTION') P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/

-- all statements that ran for at least five seconds
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('elapsed_time > 5000000')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;
/

-- select all statements that pass a simple buffer_gets threshold and
-- are coming from an APPS user
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
        'buffer_gets > 100 and parsing_schema_name = 'APPS')P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;
/

-- select all statements exceeding 5 seconds in elapsed time, but also
-- select the plans (by default we only select execution stats and binds
-- for performance reasons - in this case the SQL_PLAN attribute of sqlset_row
-- is NULL)
DECLARE
  cur sys_refcursor;
```



```

BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(dbms_sqлтune.select_cursor_cache(
      'elapsed_time > 5000000', NULL, NULL, NULL, NULL, 1, NULL,
      'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/

-- Select the top 100 statements in the cursor cache ordering by elapsed_time.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
      NULL,
      'ELAPSED_TIME', NULL, NULL,
      1,
      100)) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;
/

-- Select the set of statements which cumulatively account for 90% of the
-- buffer gets in the cursor cache. This means that the buffer gets of all
-- of these statements added up is approximately 90% of the sum of all
-- statements currently in the cache.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
      NULL,
      'BUFFER_GETS', NULL, NULL,
      .9)) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
END;
/

```

## SELECT\_SQL\_TRACE Function

This table function reads the content of one or more trace files and returns the SQL statements it finds in the format of `sqlset_row`.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SELECT_SQL_TRACE (
  directory          IN VARCHAR2,
  file_name         IN VARCHAR2 := NULL,
  mapping_table_name IN VARCHAR2 := NULL,
  mapping_table_owner IN VARCHAR2 := NULL, ,
  select_mode       IN POSITIVE := SINGLE_EXECUTION,
  options           IN BINARY_INTEGER := LIMITED_COMMAND_TYPE,
  pattern_start     IN VARCHAR2 := NULL,
  pattern_end       IN VARCHAR2 := NULL,
  result_limit      IN POSITIVE := NULL)
RETURN sys.sqlset PIPELINED;
```

### Parameters

**Table 139–40** *SELECT\_SQL\_TRACE Function Parameters*

Parameter	Description
<code>directory</code>	The directory object containing the trace file(s). This field is mandatory.
<code>file_name</code>	All or part of name of the trace file(s) to process. If <code>NULL</code> then the current or most recent file in the specified location or path is used. '%' wildcards are supported for matching trace file names.
<code>mapping_table_name</code>	The mapping table name. Note that the mapping table name is case insensitive. If the mapping table name is <code>NULL</code> , the mappings in the current database is used.
<code>mapping_table_owner</code>	the mapping table owner. If it is <code>NULL</code> , the current user is used.
<code>select_mode</code>	the mode for selecting SQL from the trace. By default, it is <code>SINGLE_EXECUTION</code> . <ul style="list-style-type: none"> <li>■ <code>SINGLE_EXECUTION</code> - return one execution of a SQL.(default).</li> <li>■ <code>ALL_EXECUTIONS</code> - return all executions.</li> </ul>
<code>options</code>	The options. By default, it is <code>LIMITED_COMMAND_TYPE</code> which can be specified to include SQL statements from all Oracle command types. <ul style="list-style-type: none"> <li>■ <code>LIMITED_COMMAND_TYPE</code> - returns the SQL statements with the command types <code>CREATE</code>, <code>INSERT</code>, <code>SELECT</code>, <code>UPDATE</code>, <code>DELETE</code>, <code>UPSERT</code>. It is the default.</li> <li>■ <code>ALL_COMMAND_TYPE</code> - returns the SQL statements with all command types.</li> </ul>
<code>pattern_start</code>	Opening delimiting pattern of the trace file section(s) to consider. CURRENTLY INOPERABLE.
<code>pattern_end</code>	closing delimiting pattern of the trace file section(s) to process. CURRENTLY INOPERABLE.

**Table 139-40 (Cont.) SELECT\_SQL\_TRACE Function Parameters**

Parameter	Description
result_limit	Top SQL from the (filtered) source. Default to MAXSB4 if NULL.

## Return Values

This function returns a SQLSET\_ROW object.

## Examples

The following code shows how to enable SQL trace for a few SQL statements and load the results into a SQL tuning set:

```
-- turn on the SQL trace in the capture database
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 4'

-- run sql statements
SELECT 1 FROM DUAL;
SELECT COUNT(*) FROM dba_tables WHERE table_name = :mytab;

ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';

-- create mapping table from the capture database
CREATE TABLE mapping AS
SELECT object_id id, owner, substr(object_name, 1, 30) name
   FROM dba_objects
   WHERE object_type NOT IN ('CONSUMER GROUP', 'EVALUATION CONTEXT',
                             'FUNCTION', 'INDEXTYPE', 'JAVA CLASS',
                             'JAVA DATA', 'JAVA RESOURCE', 'LIBRARY',
                             'LOB', 'OPERATOR', 'PACKAGE',
                             'PACKAGE BODY', 'PROCEDURE', 'QUEUE',
                             'RESOURCE PLAN', 'TRIGGER', 'TYPE',
                             'TYPE BODY')

UNION ALL
SELECT user_id id, username owner, NULL name
   FROM dba_users;

-- create the directory object where the SQL traces are stored
CREATE DIRECTORY SQL_TRACE_DIR as '/home/foo/trace';

-- create the STS
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_sts', 'test purpose');

-- load the SQL statements into STS from SQL TRACE
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
  SELECT value(p)
     FROM TABLE(
       DBMS_SQLTUNE.SELECT_SQL_TRACE(
         directory=>'SQL_TRACE_DIR',
         file_name=>'%trc',
         mapping_table_name=>'mapping')) p;
  DBMS_SQLTUNE.LOAD_SQLSET('my_sts', cur);
  CLOSE cur;
END;
/
```

## SELECT\_SQLPA\_TASK Function

This function collects SQL statements from a SQL Performance Analyzer comparison task.

### See Also:

- [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group
- *Oracle Database Real Application Testing User's Guide*

### Syntax

```
DBMS_SQLTUNE.SELECT_SQLPA_TASK(
    task_name          IN VARCHAR2,
    task_owner        IN VARCHAR2 := NULL,
    execution_name    IN VARCHAR2 := NULL,
    level_filter      IN VARCHAR2 := 'REGRESSED',
    basic_filter      IN VARCHAR2 := NULL,
    object_filter     IN VARCHAR2 := NULL,
    attribute_list    IN VARCHAR2 := 'TYPICAL')
RETURN sys.sqlset PIPELINED;
```

### Parameters

**Table 139–41** SELECT\_SQLPA\_TASK Function Parameters

Parameter	Description
task_name	Name of the SQL Performance Analyzer task
task_owner	Owner of the SQL Performance Analyzer task. If NULL, then assume the current user.
execution_name	Name of the SQL Performance Analyzer task execution (type COMPARE PERFORMANCE) from which the provided filters will be applied. If NULL, then assume the most recent COMPARE PERFORMANCE execution.
level_filter	Filter to specify which subset of SQLs to include. Same format as DBMS_SQLPA.REPORT_ANALYSIS_TASK.LEVEL, with some possible strings removed. <ul style="list-style-type: none"> <li>■ IMPROVED includes only improved SQL.</li> <li>■ REGRESSED includes only regressed SQL (default).</li> <li>■ CHANGED includes only SQL with changed performance.</li> <li>■ UNCHANGED includes only SQL with unchanged performance.</li> <li>■ CHANGED_PLANS includes only SQL with plan changes.</li> <li>■ UNCHANGED_PLANS includes only SQL with unchanged plans.</li> <li>■ ERRORS includes only SQL with errors only.</li> <li>■ MISSING_SQL includes only missing SQL statements (across STS).</li> <li>■ NEW_SQL includes only new SQL statements (across STS).</li> </ul>
basic filter	SQL predicate to filter the SQL in addition to the level filters.
object_filter	Currently not supported.

**Table 139-41 (Cont.) SELECT\_SQLPA\_TASK Function Parameters**

Parameter	Description
<code>attribute_list</code>	<p>List of SQL statement attributes to return in the result. The possible values are:</p> <ul style="list-style-type: none"> <li>▪ <code>TYPICAL - BASIC</code> + SQL plan (without row source statistics) and without object reference list (default)</li> <li>▪ <code>BASIC</code> - all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result.</li> <li>▪ <code>ALL</code> - return all attributes</li> <li>▪ Comma-separated list of attribute names this allows to return only a subset of SQL attributes: <code>EXECUTION_STATISTICS</code>, <code>SQL_BINDS</code>, <code>SQL_PLAN_STATISTICS</code> (similar to <code>SQL_PLAN</code> + row source statistics).</li> </ul>

### Return Values

This function returns a SQL tuning set object.

### Usage Notes

For example, you can use this function to create a SQL tuning set containing the subset of SQL statements that regressed during a SQL Performance Analyzer (SPA) experiment. You can also specify other arbitrary filters.

## SELECT\_SQLSET Function

This function reads SQLSET contents.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SELECT_SQLSET (
  sqlset_name      IN  VARCHAR2,
  basic_filter     IN  VARCHAR2 := NULL,
  object_filter    IN  VARCHAR2 := NULL,
  ranking_measure1 IN  VARCHAR2 := NULL,
  ranking_measure2 IN  VARCHAR2 := NULL,
  ranking_measure3 IN  VARCHAR2 := NULL,
  result_percentage IN NUMBER := 1,
  result_limit     IN  NUMBER := NULL,
  attribute_list   IN  VARCHAR2 := NULL,
  plan_filter      IN  VARCHAR2 := NULL,
  sqlset_owner     IN  VARCHAR2 := NULL,
  recursive_sql    IN  VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;
```

### Parameters

**Table 139-42** *SELECT\_SQLSET Function Parameters*

Parameter	Description
sqlset_name	The SQL tuning set name
basic_filter	The SQL predicate to filter the SQL from the SQL tuning set defined on attributes of the SQLSET_ROW
object_filter	Currently not supported.
ranking_measure (n)	An order-by clause on the selected SQL
result_percentage	A filter which picks the top N% according to the ranking measure given. Note that this applies only if one ranking measure is given.
result_limit	The top L(imit) SQL from the (filtered) source ranked by the ranking measure
attribute_list	List of SQL statement attributes to return in the result. The possible values are: <ul style="list-style-type: none"> <li>■ TYPICAL - BASIC + SQL plan (without row source statistics) and without object reference list (default)</li> <li>■ BASIC - all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result.</li> <li>■ ALL - return all attributes</li> <li>■ Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics).</li> </ul>
plan_filter	The plan filter

**Table 139-42 (Cont.) SELECT\_SQLSET Function Parameters**

Parameter	Description
sqlset_owner	The owner of the SQL tuning set, or NULL for the current schema owner
recursive_sql	Filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).

**Return Values**

This function returns a one SQLSET\_ROW per SQL\_ID or PLAN\_HASH\_VALUE pair found in each data source.

**Usage Notes**

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

**Examples**

```
-- select from a sql tuning set
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_sqlset('my_workload')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/
```

## SELECT\_WORKLOAD\_REPOSITORY Functions

This function collects SQL statements from the workload repository. The overloaded forms let you:

- Collect SQL statements from all snapshots between `begin_snap` and `end_snap`.
- Collect SQL statements from a workload repository baseline.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY (
  begin_snap      IN NUMBER,
  end_snap        IN NUMBER,
  basic_filter    IN VARCHAR2 := NULL,
  object_filter   IN VARCHAR2 := NULL,
  ranking_measure1 IN VARCHAR2 := NULL,
  ranking_measure2 IN VARCHAR2 := NULL,
  ranking_measure3 IN VARCHAR2 := NULL,
  result_percentage IN NUMBER := 1,
  result_limit    IN NUMBER := NULL,
  attribute_list  IN VARCHAR2 := NULL,
  recursive_sql   IN VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;
```

```
DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY (
  baseline_name  IN VARCHAR2,
  basic_filter    IN VARCHAR2 := NULL,
  object_filter   IN VARCHAR2 := NULL,
  ranking_measure1 IN VARCHAR2 := NULL,
  ranking_measure2 IN VARCHAR2 := NULL,
  ranking_measure3 IN VARCHAR2 := NULL,
  result_percentage IN NUMBER := 1,
  result_limit    IN NUMBER := NULL,
  attribute_list  IN VARCHAR2 := NULL,
  recursive_sql   IN VARCHAR2 := HAS_RECURSIVE_SQL)
RETURN sys.sqlset PIPELINED;
```

### Parameters

**Table 139–43** *SELECT\_WORKLOAD\_REPOSITORY Function Parameters*

Parameter	Description
<code>begin_snap</code>	Begin snapshot (non-inclusive).
<code>end_snap</code>	End snapshot (inclusive).
<code>baseline_name</code>	The name of the baseline period.
<code>basic_filter</code>	The SQL predicate to filter the SQL from the workload repository defined on attributes of the <code>SQLSET_ROW</code> . If <code>basic_filter</code> is not set by the caller, the subprogram captures only statements of the type <code>CREATE TABLE</code> , <code>INSERT</code> , <code>SELECT</code> , <code>UPDATE</code> , <code>DELETE</code> , and <code>MERGE</code> .
<code>object_filter</code>	Currently not supported.
<code>ranking_measure (n)</code>	An order-by clause on the selected SQL.



**Table 139-43 (Cont.) SELECT\_WORKLOAD\_REPOSITORY Function Parameters**

Parameter	Description
result_percentage	A filter which picks the top N% according to the ranking measure given. Note that this applies only if one ranking measure is given.
result_limit	The top L(imit) SQL from the (filtered) source ranked by the ranking measure.
attribute_list	List of SQL statement attributes to return in the result. The possible values are: <ul style="list-style-type: none"> <li>■ TYPICAL - BASIC + SQL plan (without row source statistics) and without object reference list (default)</li> <li>■ BASIC - all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result.</li> <li>■ ALL - return all attributes</li> <li>■ Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics).</li> </ul>
recursive_sql	Filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).

## Return Values

This function returns a one SQLSET\_ROW per SQL\_ID or PLAN\_HASH\_VALUE pair found in each data source.

## Usage Notes

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

## Examples

```
-- select statements from snapshots 1-2
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_workload_repository(1,2)) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/
```

## SET\_TUNING\_TASK\_PARAMETER Procedures

This procedure updates the value of a SQL tuning parameter of type VARCHAR2 or NUMBER.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(  
    task_name    IN  VARCHAR2,  
    parameter    IN  VARCHAR2,  
    value        IN  VARCHAR2);
```

```
DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(  
    task_name    IN  VARCHAR2,  
    parameter    IN  VARCHAR2,  
    value        IN  NUMBER);
```

### Parameters

**Table 139–44** SET\_TUNING\_TASK\_PARAMETER Procedure Parameters

Parameter	Description
task_name	Identifier of the task to execute

**Table 139–44 (Cont.) SET\_TUNING\_TASK\_PARAMETER Procedure Parameters**

Parameter	Description
parameter	<p>Name of the parameter to set. The possible tuning parameters that can be set by this procedure using the parameter in the form VARCHAR2:</p> <ul style="list-style-type: none"> <li>▪ APPLY_CAPTURED_COMPILEENV: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO).</li> <li>▪ BASIC_FILTER: basic filter for SQL tuning set</li> <li>▪ DAYS_TO_EXPIRE: number of days until the task is deleted</li> <li>▪ DEFAULT_EXECUTION_TYPE: the task defaults to this type of execution when none is specified by the <a href="#">EXECUTE_TUNING_TASK Function &amp; Procedure</a></li> <li>▪ EXECUTION_DAYS_TO_EXPIRE: number of days until the tasks's executions is deleted (without deleting the task)</li> <li>▪ LOCAL_TIME_LIMIT: per-statement time out (seconds)</li> <li>▪ MODE: tuning scope (comprehensive, limited)</li> <li>▪ OBJECT_FILTER: object filter for SQL tuning set</li> <li>▪ PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values)</li> <li>▪ RANK_MEASURE1: first ranking measure for SQL tuning set</li> <li>▪ RANK_MEASURE2: second possible ranking measure for SQL tuning set</li> <li>▪ RANK_MEASURE3: third possible ranking measure for SQL tuning set</li> <li>▪ RESUME_FILTER: a extra filter for SQL tuning sets besides BASIC_FILTER</li> <li>▪ SQL_LIMIT: maximum number of SQL statements to tune</li> <li>▪ SQL_PERCENTAGE: percentage filter of SQL tuning set statements</li> <li>▪ TEST_EXECUTE: FULL/AUTO/OFF. <ul style="list-style-type: none"> <li>* FULL - test-execute for as much time as necessary, up to the local time limit for the SQL (or the global task time limit if no SQL time limit is set)</li> <li>* AUTO - test-execute for an automatically-chosen time proportional to the tuning time</li> <li>* OFF - do not test-execute</li> </ul> </li> <li>▪ TIME_LIMIT: global time out (seconds)</li> <li>▪ USERNAME: username under which the statement is parsed</li> </ul>
value	New value of the specified parameter

## Usage Notes

When setting automatic tuning task parameters, use the [SET\\_AUTO\\_TUNING\\_TASK\\_PARAMETER Procedures](#) in the [DBMS\\_AUTO\\_SQLTUNE](#) package.

## SQLTEXT\_TO\_SIGNATURE Function

This function returns a SQL text's signature. The signature can be used to identify SQL text in `dba_sql_profiles`.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE (  
  sql_text      IN CLOB,  
  force_match  IN BOOLEAN := FALSE)  
RETURN NUMBER;
```

### Parameters

**Table 139–45** *SQLTEXT\_TO\_SIGNATURE Function Parameters*

Parameter	Description
<code>sql_text</code>	SQL text whose signature is required. Required.
<code>force_match</code>	If TRUE, this returns a signature that supports SQL matching with literal values transformed into bind variables. If FALSE, returns the signature based on the text with literals not transformed

### Return Values

This function returns the signature of the specified SQL text.

## UNPACK\_STGTAB\_SQLPROF Procedure

This procedure copies profile data stored in the staging table to create profiles on the system.

**See Also:** [SQL Profile Subprograms](#) on page 139-12 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF (
  profile_name      IN VARCHAR2 := '%',
  profile_category IN VARCHAR2 := 'DEFAULT',
  replace           IN BOOLEAN,
  staging_table_name IN VARCHAR2,
  staging_schema_owner IN VARCHAR2 := NULL);
```

### Parameters

**Table 139-46 UNPACK\_STGTAB\_SQLPROF Procedure Parameters**

Parameter	Description
profile_name	The name of the profile to unpack (% wildcards acceptable, case-sensitive)
profile_category	The category from which to unpack profiles (% wildcards acceptable, case-sensitive)
replace	The option to replace profiles if they already exist. Note that profiles cannot be replaced if one in the staging table has the same name as an active profile in a different SQL statement. If FALSE, this function raises errors if you try to create a profile that already exists
staging_table_name	The name of the table on which to perform the remap operation (case-insensitive unless double quoted). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-insensitive unless double quoted)

### Usage Notes

Using this procedure requires the CREATE ANY SQL PROFILE privilege and the SELECT privilege on staging table.

### Examples

```
-- Unpack all profiles stored in a staging table
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(replace           => FALSE, -
                                         staging_table_name => 'PROFILE_STGTAB');
```

-- If there is a failure during the unpack operation, users can find the profile  
-- we failed on and perform a remap\_stgtab\_sqlprof operation targeting it. Then  
-- they can resume the unpack operation by setting replace to TRUE so that  
-- the profiles that were already created are replaced

```
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(replace           => TRUE, -
                                         staging_table_name => 'PROFILE_STGTAB');
```

## UNPACK\_STGTAB\_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the staging table into the SQL tuning sets schema, making them proper SQL tuning sets.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET (
  sqlset_name          IN VARCHAR2 := '%',
  sqlset_owner        IN VARCHAR2 := NULL,
  replace              IN BOOLEAN,
  staging_table_name   IN VARCHAR2,
  staging_schema_owner IN VARCHAR2 := NULL);
```

### Parameters

**Table 139-47 UNPACK\_STGTAB\_SQLSET Procedure Parameters**

Parameter	Description
sqlset_name	The name of the tuning set to unpack (not NULL). Wildcard characters (%) are supported to unpack multiple tuning sets in a single call. For example, just specify '%' to unpack all tuning sets from the staging table.
sqlset_owner	The name of tuning set owner, or NULL for current schema owner. Wildcards supported.
replace	Replaces tuning set if they already exist. If FALSE, raises errors if you try to create a tuning set that already exists
staging_table_name	The name of the staging table, moved after a call to the <a href="#">PACK_STGTAB_SQLSET Procedure</a> (case-sensitive)
staging_schema_owner	The name of staging table owner, or NULL for current schema owner (case-sensitive)

### Usage Notes

- Users can drop the staging table after this procedure completes successfully.
- The unpack procedure commits after successfully loading each SQL tuning set. If it fails with one tuning set, no part of that tuning set will have been unpacked, but those which the subprogram had already apprehended continue to exist.
- When failures occur due to SQL tuning set name or owner conflicts, users should use the [REMAP\\_STGTAB\\_SQLSET Procedure](#) to patch the staging table, and then call this procedure again to unpack those tuning sets that remain.

### Examples

```
-- unpack all STS in the staging table
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name    => '%', -
                                       sqlset_owner    => '%', -
                                       replace         => FALSE, -
                                       staging_table_name => 'STGTAB_SQLSET');
```

-- errors can arise during STS unpack when a STS in the staging table has the  
-- same name/owner as STS on the system. In this case, users should call

```
-- remap_stgtab_sqlset to patch the staging table and with which to call unpack
-- Replace set to TRUE.
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name      => '%', -
                                       sqlset_owner     => '%', -
                                       replace           => TRUE, -
                                       staging_table_name => 'STGTAB_SQLSET');
```

## UPDATE\_SQLSET Procedures

This procedure updates selected fields for SQL statement in a SQL tuning set.

**See Also:** [SQL Tuning Set Subprograms](#) on page 139-13 for other subprograms in this group

### Syntax

```
DBMS_SQLTUNE.UPDATE_SQLSET (
  sqlset_name      IN  VARCHAR2,
  sql_id           IN  VARCHAR2,
  attribute_name   IN  VARCHAR2,
  attribute_value  IN  VARCHAR2 := NULL);
```

```
DBMS_SQLTUNE.UPDATE_SQLSET (
  sqlset_name      IN  VARCHAR2,
  sql_id           IN  VARCHAR2,
  attribute_name   IN  VARCHAR2,
  attribute_value  IN  NUMBER := NULL);
```

### Parameters

**Table 139–48** UPDATE\_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	The SQL tuning set name
sql_id	The identifier of the statement to update
attribute_name	The name of the attribute to modify
attribute_value	The new value of the attribute



---

---

## DBMS\_STAT\_FUNCS

The DBMS\_STAT\_FUNCS package provides statistical functions.

This chapter contains the following topic:

- [Summary of DBMS\\_STAT\\_FUNCS Subprograms](#)

## Summary of DBMS\_STAT\_FUNCS Subprograms

**Table 140–1** *DBMS\_STAT\_FUNCS Package Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">EXPONENTIAL_DIST_FIT Procedure</a> on page 140-3	Tests how well a sample of values fits an exponential distribution
<a href="#">NORMAL_DIST_FIT Procedure</a> on page 140-4	Tests how well a sample of values fits a normal distribution
<a href="#">POISSON_DIST_FIT Procedure</a> on page 140-5	Tests how well a sample of values fits a Poisson distribution
<a href="#">SUMMARY Procedure</a> on page 140-6	Summarizes a numerical column of a table
<a href="#">UNIFORM_DIST_FIT Procedure</a> on page 140-7	Tests how well a sample of values fits a uniform distribution
<a href="#">WEIBULL_DIST_FIT Procedure</a> on page 140-8	Tests how well a sample of values fits a Weibull distribution

## EXPONENTIAL\_DIST\_FIT Procedure

This procedure tests how well a sample of values fits an exponential distribution.

### Syntax

```
DBMS_STAT_FUNCS.EXPONENTIAL_DIST_FIT (
  ownername   IN   VARCHAR2,
  tablename   IN   VARCHAR2,
  columnname  IN   VARCHAR2,
  test_type   IN   VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
  lambda      IN   NUMBER,
  mu          IN   NUMBER,
  sig         OUT  NUMBER);
```

### Parameters

**Table 140–2 EXPONENTIAL\_DIST\_FIT Procedure Parameters**

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
lambda	The scale parameter.
mu	The location parameter.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the exponential distribution. A number close to 1 indicates a close match.

## NORMAL\_DIST\_FIT Procedure

This procedure tests how well a sample of values fits a normal distribution.

### Syntax

```
DBMS_STAT_FUNCS.NORMAL_DIST_FIT (  
    ownername    IN    VARCHAR2,  
    tablename    IN    VARCHAR2,  
    columnname   IN    VARCHAR2,  
    test_type    IN    VARCHAR2 DEFAULT 'SHAPIRO_WILKS',  
    mean         IN    NUMBER,  
    stdev        IN    NUMBER,  
    sig          OUT   NUMBER);
```

### Parameters

**Table 140–3** *NORMAL\_DIST\_FIT Procedure Parameters*

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV', 'ANDERSON_DARLING' or 'SHAPIRO_WILKS'.
mean	The mean of the distribution against which to compare.
stdev	The standard deviation of the distribution against which to compare.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the normal distribution. A number close to 1 indicates a close match.

## POISSON\_DIST\_FIT Procedure

This procedure tests how well a sample of values fits a Poisson distribution.

### Syntax

```
DBMS_STAT_FUNCS.POISSON_DIST_FIT (
  ownername   IN   VARCHAR2,
  tablename   IN   VARCHAR2,
  columnname  IN   VARCHAR2,
  test_type   IN   VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
  lambda      IN   NUMBER,
  sig         OUT  NUMBER);
```

### Parameters

**Table 140–4 POISSON\_DIST\_FIT Procedure Parameters**

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
lambda	The lambda parameter is the shape parameter.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the Poisson distribution. A number close to 1 indicates a close match.

## SUMMARY Procedure

This procedure summarizes the numerical column specified in the `columnname` of `tablename`. The summary is returned as a Summary Type. Note that most of the output of `SUMMARY` can be obtained with currently available SQL.

### Syntax

```
DBMS_STAT_FUNCS.SUMMARY (
  ownername   IN   VARCHAR2,
  tablename   IN   VARCHAR2,
  columnname  IN   VARCHAR2,
  sigma_value IN   NUMBER DEFAULT 3,
  s           OUT  SummaryType);
```

### Parameters

**Table 140–5** *SUMMARY Procedure Parameters*

Parameter	Description
<code>ownername</code>	The schema where the table resides.
<code>tablename</code>	The table where the column resides.
<code>columnname</code>	The column of the table to be summarized.
<code>sigma_value</code>	The number of sigmas for the set of extreme values, defaults to 3.
<code>s</code>	The Record containing summary information about given column.

### Definition of SummaryType

```
TYPE n_arr IS VARRAY(5) of NUMBER;
TYPE num_table IS TABLE of NUMBER;
TYPE summaryType IS RECORD (
  count          NUMBER,
  min            NUMBER,
  max           NUMBER,
  range         NUMBER,
  mean         NUMBER,
  cmode       num_table,
  variance    NUMBER,
  stddev     NUMBER,
  quantile_5  NUMBER,
  quantile_25 NUMBER,
  median     NUMBER,
  quantile_75 NUMBER,
  quantile_95 NUMBER,
  plus_x_sigma NUMBER,
  minus_x_sigma NUMBER,
  extreme_values num_table,
  top_5_values  n_arr,
  bottom_5_values n_arr);
```

## UNIFORM\_DIST\_FIT Procedure

This procedure tests well a sample of values fits a uniform distribution.

### Syntax

```
DBMS_STAT_FUNCS.UNIFORM_DIST_FIT (
  ownername   IN   VARCHAR2,
  tablename   IN   VARCHAR2,
  columnname  IN   VARCHAR2,
  var_type    IN   VARCHAR2 DEFAULT 'CONTINUOUS',
  test_type   IN   VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
  paramA      IN   NUMBER,
  paramB      IN   NUMBER,
  sig         OUT  NUMBER);
```

### Parameters

**Table 140–6 UNIFORM\_DIST\_FIT Procedure Parameters**

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
var_type	The type of distribution: 'CONTINUOUS' (the default) or 'DISCRETE'
test_type	The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
paramA	Parameter A estimated from the sample (the location parameter).
paramB	Parameter B estimated from the sample (the scale parameter).
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the uniform distribution. A number close to 1 indicates a close match.

## WEIBULL\_DIST\_FIT Procedure

This procedure tests how well a sample of values fits a Weibull distribution.

### Syntax

```
DBMS_STAT_FUNCS.WEIBULL_DIST_FIT (
  ownername   IN   VARCHAR2,
  tablename   IN   VARCHAR2,
  columnname  IN   VARCHAR2,
  test_type   IN   VARCHAR2 DEFAULT 'KOLMOGOROV_SMIRNOV',
  alpha       IN   NUMBER,
  mu          IN   NUMBER,
  beta        IN   NUMBER,
  sig         OUT  NUMBER);
```

### Parameters

**Table 140–7 WEIBULL\_DIST\_FIT Procedure Parameters**

Parameter	Description
ownername	The schema where the table resides.
tablename	The table where the column resides.
columnname	The column of the table against which to run the test.
test_type	The type of test to use: 'CHI_SQUARED', 'KOLMOGOROV_SMIRNOV' or 'ANDERSON_DARLING'.
alpha	The scale parameter.
mu	The location parameter.
beta	The slope/shape parameter.
sig	The goodness of fit value, based on test type. A small value indicates a significant difference between the sample and the Weibull distribution. A number close to 1 indicates a close match.



With the `DBMS_STATS` package you can view and modify optimizer statistics gathered for database objects.

**See Also:** *Oracle Database Performance Tuning Guide*

This chapter contains the following topics:

- [Using DBMS\\_STATS](#)
  - Overview
  - Types
  - Constants
  - Operational Notes
  - Deprecated Subprograms
  - Examples
- [Summary of DBMS\\_STATS Subprograms](#)

---

## Using DBMS\_STATS

This section contains topics which relate to using the DBMS\_STATS package.

- [Overview](#)
- [Types](#)
- [Constants](#)
- [Operational Notes](#)
- [Deprecated Subprograms](#)
- [Examples](#)

## Overview

The Oracle RDBMS allows you to collect statistics of many different kinds as an aid to improving performance. This package is concerned with optimizer statistics only. Given that Oracle sets automatic statistics collection of this kind on by default, this package is intended for only specialized cases.

The statistics of interest to be viewed or modified can reside in the dictionary or in a table created in the user's schema for this purpose. You can also collect and manage user-defined statistics for tables and domain indexes using this package.

For example, if the `DELETE_COLUMN_STATS` procedure is invoked on a column for which an association is defined, user-defined statistics for that column are deleted in addition to deletion of the standard statistics.

Only statistics stored in the dictionary have an impact on the cost-based optimizer. You can also use `DBMS_STATS` to gather statistics in parallel

**See Also:** *Oracle Database Performance Tuning Guide* for more information about "Managing Optimizer Statistics".

## Types

Types for the minimum and maximum values and histogram endpoints include:

```
TYPE numarray IS VARRAY(256) OF NUMBER;
TYPE datearray IS VARRAY(256) OF DATE;
TYPE chararray IS VARRAY(256) OF VARCHAR2(4000);
TYPE rawarray IS VARRAY(256) OF RAW(2000);
TYPE fltarray IS VARRAY(256) OF BINARY_FLOAT;
TYPE dblarray IS VARRAY(256) OF BINARY_DOUBLE;
```

```
TYPE StatRec IS RECORD (
    epc NUMBER,
    minval RAW(2000),
    maxval RAW(2000),
    bkvals NUMARRAY,
    novals NUMARRAY,
    chvals CHARARRAY,
    eavs NUMBER);
```

Types for listing stale tables include:

```
TYPE ObjectElem IS RECORD (
    ownname VARCHAR2(30), -- owner
    objtype VARCHAR2(6), -- 'TABLE' or 'INDEX'
    objname VARCHAR2(30), -- table/index
    partname VARCHAR2(30), -- partition
    subpartname VARCHAR2(30)); -- subpartition
type ObjectTab is TABLE of ObjectElem;
```

Type for displaying statistics difference report:

```
TYPE DiffRepElem IS RECORD (
    report CLOB, -- stats difference report
    maxdiffpct number); -- max stats difference (percentage)
type DiffRepTab is table of DiffRepElem;
```

## Constants

The DBMS\_STATS package uses the constants shown in [Table 141-1](#):

**Table 141-1 DBMS\_STATS Constants**

Name	Type	Value	Description
AUTO_CASCADE	BOOLEAN	NULL	Lets Oracle decide whether to collect statistics for indexes or not
AUTO_DEGREE	NUMBER	32768	Lets Oracle select the degree of parallelism based on size of the object, number of CPUs and initialization parameters. For definition of default parallel degree, see "Degree of Parallelism" in <i>Oracle Database VLDB and Partitioning Guide</i> .
AUTO_INVALIDATE	BOOLEAN	NULL	Lets Oracle decide when to invalidate dependent cursors
AUTO_SAMPLE_SIZE	NUMBER	0	Indicates that auto-sample size algorithms should be used

## Operational Notes

The `DBMS_STATS` subprograms perform the following general operations:

- [Gathering Optimizer Statistics](#)
- [Setting or Getting Statistics](#)
- [Deleting Statistics](#)
- [Transferring Statistics](#)
- [Locking or Unlocking Statistics](#)
- [Restoring and Purging Statistics History](#)
- [User-Defined Statistics](#)
- [Pending Statistics](#)
- [Comparing Statistics](#)
- [Extended Statistics](#)

Most of the `DBMS_STATS` procedures include the three parameters `statown`, `stattab`, and `statid`. These parameters allow you to store statistics in your own tables (outside of the dictionary), which does not affect the optimizer. Therefore, you can maintain and experiment with *sets* of statistics.

The `stattab` parameter specifies the name of a table in which to hold statistics, and it is assumed that it resides in the same schema as the object for which statistics are collected (unless the `statown` parameter is specified). You can create multiple tables with different `stattab` identifiers to hold separate sets of statistics.

Additionally, you can maintain different sets of statistics within a single `stattab` by using the `statid` parameter, which avoids cluttering the user's schema.

For the `SET` and `GET` procedures, if `stattab` is not provided (that is, `NULL`), then the operation works directly on the dictionary statistics; therefore, you do not need to create these statistics tables if they only plan to modify the dictionary directly. However, if `stattab` is not `NULL`, then the `SET` or `GET` operation works on the specified user statistics table, and not the dictionary.

You can change the default values of some of the parameters of `DBMS_STATS` procedures using the [SET\\_DATABASE\\_PREFS Procedure](#), [SET\\_GLOBAL\\_PREFS Procedure](#), [SET\\_SCHEMA\\_PREFS Procedure](#) and [SET\\_TABLE\\_PREFS Procedure](#).

Most of the procedures in this package commit the current transaction, perform the operation, and then commit again.

Most of the procedures have a parameter, `force` which allows you to override any lock on statistics.

Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoring.

### Gathering Optimizer Statistics

Use the following subprograms to gather certain classes of optimizer statistics, with possible performance improvements over the `ANALYZE` command:

[GATHER\\_DATABASE\\_STATS Procedures](#)  
[GATHER\\_DICTIONARY\\_STATS Procedure](#)  
[GATHER\\_FIXED\\_OBJECTS\\_STATS Procedure](#)

[GATHER\\_INDEX\\_STATS Procedure](#)  
[GATHER\\_SCHEMA\\_STATS Procedures](#)  
[GATHER\\_SYSTEM\\_STATS Procedure](#)  
[GATHER\\_TABLE\\_STATS Procedure](#)

The `GATHER_*` procedures also collect user-defined statistics for columns and domain indexes.

The `statown`, `stattab`, and `statid` parameters instruct the package to back up current statistics in the specified table before gathering new statistics.

Oracle also provides the following procedure for generating statistics for derived objects when you have sufficient statistics on related objects:

[GENERATE\\_STATS Procedure](#)

## Setting or Getting Statistics

Use the following subprograms to store and retrieve individual column-related, index-related, and table-related statistics:

[PREPARE\\_COLUMN\\_VALUES Procedures](#)  
[PREPARE\\_COLUMN\\_VALUES\\_NVARCHAR2 Procedure](#)  
[PREPARE\\_COLUMN\\_VALUES\\_ROWID Procedure](#)

[SEED\\_COL\\_USAGE Procedure](#)  
[SET\\_INDEX\\_STATS Procedures](#)  
[SET\\_SYSTEM\\_STATS Procedure](#)  
[SET\\_TABLE\\_STATS Procedure](#)

[GET\\_COLUMN\\_STATS Procedures](#)  
[GET\\_INDEX\\_STATS Procedures](#)  
[GET\\_SYSTEM\\_STATS Procedure](#)  
[GET\\_TABLE\\_STATS Procedure](#)

In the special versions of the `SET_*_STATS` procedures for setting user-defined statistics, the following, if provided, are stored in the dictionary or external statistics table:

- User-defined statistics (`extstats`)
- The statistics type schema name (`statsschema`)
- The statistics type name (`statsname`)

The user-defined statistics and the corresponding statistics type are inserted into the `USTATS$` dictionary table. You can specify user-defined statistics without specifying the statistics type name.

The special versions of the `GET_*_STATS` procedures return user-defined statistics and the statistics type owner and name as `OUT` arguments corresponding to the schema object specified. If user-defined statistics are not collected, `NULL` values are returned.

## Deleting Statistics

The `DELETE_*` procedures delete both user-defined statistics and the standard statistics for the given schema object.

[DELETE\\_COLUMN\\_STATS Procedure](#)  
[DELETE\\_DATABASE\\_STATS Procedure](#)

[DELETE\\_DICTIONARY\\_STATS Procedure](#)  
[DELETE\\_FIXED\\_OBJECTS\\_STATS Procedure](#)  
[DELETE\\_INDEX\\_STATS Procedure](#)  
[DELETE\\_SCHEMA\\_STATS Procedure](#)  
[DELETE\\_SYSTEM\\_STATS Procedure](#)  
[DELETE\\_TABLE\\_STATS Procedure](#)

## Transferring Statistics

Use the following procedures for creating and dropping the user statistics table.

[CREATE\\_STAT\\_TABLE Procedure](#)  
[DROP\\_STAT\\_TABLE Procedure](#)

Use the following procedures to transfer statistics

- from the dictionary to a user statistics table ([EXPORT\\_\\*](#))
- from a user statistics table to the dictionary ([IMPORT\\_\\*](#))

[EXPORT\\_COLUMN\\_STATS Procedure](#)  
[EXPORT\\_DATABASE\\_STATS Procedure](#)  
[EXPORT\\_DICTIONARY\\_STATS Procedure](#)  
[EXPORT\\_FIXED\\_OBJECTS\\_STATS Procedure](#)  
[EXPORT\\_INDEX\\_STATS Procedure](#)  
[EXPORT\\_SCHEMA\\_STATS Procedure](#)  
[EXPORT\\_SYSTEM\\_STATS Procedure](#)  
[EXPORT\\_TABLE\\_STATS Procedure](#)

[IMPORT\\_COLUMN\\_STATS Procedure](#)  
[IMPORT\\_DATABASE\\_STATS Procedure](#)  
[IMPORT\\_DICTIONARY\\_STATS Procedure](#)  
[IMPORT\\_FIXED\\_OBJECTS\\_STATS Procedure](#)  
[IMPORT\\_INDEX\\_STATS Procedure](#)  
[IMPORT\\_SCHEMA\\_STATS Procedure](#)  
[IMPORT\\_SYSTEM\\_STATS Procedure](#)  
[IMPORT\\_TABLE\\_STATS Procedure](#)

---

---

**Note:** Oracle does not support export or import of statistics across databases of different character sets.

---

---

## Locking or Unlocking Statistics

Use the following procedures to lock and unlock statistics on objects.

[LOCK\\_SCHEMA\\_STATS Procedure](#)  
[LOCK\\_TABLE\\_STATS Procedure](#)

[UNLOCK\\_SCHEMA\\_STATS Procedure](#)  
[UNLOCK\\_TABLE\\_STATS Procedure](#)

The [LOCK\\_\\*](#) procedures either freeze the current set of the statistics or to keep the statistics empty (uncollected). When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.



## Restoring and Purging Statistics History

Use the following procedures to restore statistics as of a specified timestamp. This is useful in case newly collected statistics leads to some sub-optimal execution plans and the administrator wants to revert to the previous set of statistics.

[RESET\\_GLOBAL\\_PREF\\_DEFAULTS Procedure](#)

[RESTORE\\_DICTIONARY\\_STATS Procedure](#)

[RESTORE\\_FIXED\\_OBJECTS\\_STATS Procedure](#)

[RESTORE\\_SCHEMA\\_STATS Procedure](#)

[RESTORE\\_SYSTEM\\_STATS Procedure](#)

[RESTORE\\_TABLE\\_STATS Procedure](#)

Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoring. The old statistics are purged automatically at regular intervals based on the statistics history retention setting and the time of recent statistics gathering performed in the system. Retention is configurable using the [ALTER\\_STATS\\_HISTORY\\_RETENTION Procedure](#).

The other DBMS\_STATS procedures related to restoring statistics are:

- [PURGE\\_STATS Procedure](#): This procedure lets you manually purge old versions beyond a time stamp.
- [GET\\_STATS\\_HISTORY\\_RETENTION Function](#): This function gets the current statistics history retention value.
- [GET\\_STATS\\_HISTORY\\_AVAILABILITY Function](#): This function gets the oldest time stamp where statistics history is available. Users cannot restore statistics to a time stamp older than the oldest time stamp.

RESTORE\_\* operations are not supported for user defined statistics.

## User-Defined Statistics

The DBMS\_STATS package supports operations on user-defined statistics. When a domain index or column is associated with a statistics type (using the `associate` statement), operations on the index or column manipulate user-defined statistics. For example, gathering statistics for a domain index (for which an association with a statistics type exists) using the [GET\\_INDEX\\_STATS Procedures](#) invokes the user-defined statistics collection method of the associated statistics type. Similarly, delete, transfer, import, and export operations manipulate user-defined statistics.

SET\_\* and GET\_\* operations for user-defined statistics are also supported using a special version of the SET and GET interfaces for columns and indexes.

EXPORT\_\*, IMPORT\_\* and RESTORE\_\* operations are not supported for user defined statistics.

## Pending Statistics

The package gather statistics and stores it in the dictionary by default. User's can store these statistics in the system's private area instead of the dictionary by turning the PUBLISH option to FALSE using the SET\*PREFS procedures. The default value for PUBLISH is TRUE. The statistics stored in private area are not used by Cost Based Optimizer unless parameter `optimizer_use_pending_statistics` is set to TRUE. The default value of this parameter is FALSE and this boolean parameter can be set at the session/system level. Users can verify the impact of the new statistics on query plans by using the pending statistics on a session.

Pending statistics provide a mechanism to verify the impact of the new statistics on query plans before making them available for general use. There are two scenarios to verify the query plans:

- Export the pending statistics (use the [EXPORT\\_PENDING\\_STATS Procedure](#)) to a test system, then run the query workload and check the performance or plans.
- Set `optimizer_use_pending_statistics` to `TRUE` in a session on the system where pending statistics have been gathered, run the workload, and check the performance or plans.

Once the performance or query plans have been verified, the pending statistics can be published (run the [PUBLISH\\_PENDING\\_STATS Procedure](#)) if the performance is acceptable or delete (run the [DELETE\\_PENDING\\_STATS Procedure](#)) if not.

Pending statistics can be published, exported, or deleted. The following procedures are provided to manage pending statistics:

- [DELETE\\_PENDING\\_STATS Procedure](#)
- [EXPORT\\_PENDING\\_STATS Procedure](#)
- [PUBLISH\\_PENDING\\_STATS Procedure](#)

## Comparing Statistics

The `DIFF_TABLE_STATS_*` statistics can be used to compare statistics for a table from two different sources. The statistics can be from:

- two different user statistics tables
- a single user statistics table containing two sets of statistics that can be identified using `stats`
- a user statistics table and dictionary history
- pending statistics

The functions also compare the statistics of the dependent objects (indexes, columns, partitions). They display statistics of the object(s) from both sources if the difference between those statistics exceeds a certain threshold (%). The threshold can be specified as an argument to the function, with a default of 10%. The statistics corresponding to the first source (`stattab1` or `time1`) will be used as basis for computing the differential percentage.

## Extended Statistics

This package allows you to collect statistics for column groups and expressions (known as "statistics extensions"). The statistics collected for column groups and expressions are called "extended statistics".

Statistics on Column groups are used by optimizer for accounting correlation between columns. For example, if a query has predicates `c1=1` and `c2=1` and if there are statistics on (c1, c2), the optimizer will use this statistics for estimating the combined selectivity of the predicates. The expression statistics are used by optimizer for estimating selectivity of predicates on those expressions. The extended statistics are similar to column statistics and the procedures that take columns names will accept extension names in place of column names.

Related subprograms:

- [CREATE\\_EXTENDED\\_STATS Function](#)
- [DROP\\_EXTENDED\\_STATS Procedure](#)

- [SHOW\\_EXTENDED\\_STATS\\_NAME](#) Function

## Deprecated Subprograms

The following subprograms are obsolete with Release 11g:

- [GET\\_PARAM](#) Function  
Instead, use [GET\\_PREFS](#) Function
- [SET\\_PARAM](#) Procedure  
Instead, use [SET\\_GLOBAL\\_PREFS](#) Procedure
- [RESET\\_PARAM\\_DEFAULTS](#) Procedure  
Instead use [RESET\\_GLOBAL\\_PREF\\_DEFAULTS](#) Procedure

## Examples

- [Using Pending Statistics](#)
- [Gathering Daytime System Statistics](#)

### Using Pending Statistics

Assume many modifications have been made to the `employees` table since the last time statistics were gathered. To ensure that the cost-based optimizer is still picking the best plan, statistics should be gathered once again; however, the user is concerned that new statistics will cause the optimizer to choose bad plans when the current ones are acceptable. The user can do the following:

```
EXEC DBMS_STATS.SET_TABLE_PREFS('hr', 'employees', 'PUBLISH', 'false');
```

By setting the `employees` tables publish preference to `FALSE`, any statistics gather from now on will not be automatically published. The newly gathered statistics will be marked as pending.

```
EXEC DBMS_STATS.GATHER_TABLE_STATS ('hr', 'employees');
```

To test the newly gathered statistics, set `optimizer_pending_statistics` to `TRUE` in a session and run sample queries.

```
ALTER SESSION SET optimizer_use_pending_statistics = TRUE;
```

If the pending statistics generate sound execution plans, they can be published:

```
EXEC DBMS_STATS.PUBLISH_PENDING_STATS('hr', 'employees');
EXEC DBMS_STATS.SET_TABLE_PREF('hr', 'employees', 'PUBLISH', 'true');
```

### Gathering Daytime System Statistics

Assume that you want to perform database application processing OLTP transactions during the day and run reports at night.

To collect daytime system statistics, gather statistics for 720 minutes. Store the statistics in the `MYSTATS` table.

```
BEGIN
  DBMS_STATS.GATHER_SYSTEM_STATS (
    interval => 720,
    stattab  => 'mystats',
    statid   => 'OLTP');
END;
```

To collect nighttime system statistics, gather statistics for 720 minutes. Store the statistics in the `MYSTATS` table.

```
BEGIN
  DBMS_STATS.GATHER_SYSTEM_STATS (
    interval => 720,
    stattab  => 'mystats',
    statid   => 'OLAP');
END;
```

Update the dictionary with the gathered statistics.

```
VARIABLE jobno number;
BEGIN
  DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS
```

```
        ('mystats','OLTP');'  
        sysdate, 'sysdate + 1');  
        COMMIT;  
    END;  
  
    BEGIN  
        DBMS_JOB.SUBMIT (:jobno, 'DBMS_STATS.IMPORT_SYSTEM_STATS  
        ('mystats','OLAP');'  
        sysdate + 0.5, 'sysdate + 1');  
        COMMIT;  
    END;
```

---

## Summary of DBMS\_STATS Subprograms

**Table 141–2 DBMS\_STATS Package Subprograms**

Subprogram	Description
<a href="#">ALTER_STATS_HISTORY_RETENTION Procedure</a> on page 141-20	Changes the statistics history retention value
<a href="#">CONVERT_RAW_VALUE Procedures</a> on page 141-21	Convert the internal representation of a minimum or maximum value into a datatype-specific value
<a href="#">CONVERT_RAW_VALUE_NVARCHAR Procedure</a> on page 141-22	Convert the internal representation of a minimum or maximum value into a datatype-specific value
<a href="#">CONVERT_RAW_VALUE_ROWID Procedure</a> on page 141-23	Convert the internal representation of a minimum or maximum value into a datatype-specific value
<a href="#">COPY_TABLE_STATS Procedure</a> on page 141-24	Copies the statistics of the source [sub] partition to the destination [sub] partition after scaling
<a href="#">CREATE_EXTENDED_STATS Function</a> on page 141-26	Creates a virtual column for a user specified column group or an expression in a table
<a href="#">CREATE_STAT_TABLE Procedure</a> on page 141-28	Creates a table with name <code>stattab</code> in ownname's schema which is capable of holding statistics
<a href="#">DELETE_COLUMN_STATS Procedure</a> on page 141-29	Deletes column-related statistics
<a href="#">DELETE_DATABASE_PREFS Procedure</a> on page 141-31	Deletes the statistics preferences of all the tables, excluding the tables owned by Oracle
<a href="#">DELETE_DATABASE_STATS Procedure</a> on page 141-34	Deletes statistics for the entire database
<a href="#">DELETE_DICTIONARY_STATS Procedure</a> on page 141-35	Deletes statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas)
<a href="#">DELETE_FIXED_OBJECTS_STATS Procedure</a> on page 141-34	Deletes statistics of all fixed tables
<a href="#">DELETE_INDEX_STATS Procedure</a> on page 141-37	Deletes index-related statistics
<a href="#">DELETE_PENDING_STATS Procedure</a> on page 141-38	Deletes the private statistics that have been collected but have not been published
<a href="#">DELETE_SCHEMA_PREFS Procedure</a> on page 141-39	Deletes the statistics preferences of all the tables owned by the specified owner name
<a href="#">DELETE_SCHEMA_STATS Procedure</a> on page 141-42	Deletes schema-related statistics
<a href="#">DELETE_SYSTEM_STATS Procedure</a> on page 141-43	Deletes system statistics
<a href="#">DELETE_TABLE_PREFS Procedure</a> on page 141-44	Deletes statistics preferences of the specified table in the specified schema
<a href="#">DELETE_TABLE_STATS Procedure</a> on page 141-47	Deletes table-related statistics
<a href="#">DIFF_TABLE_STATS_IN_HISTORY Function</a> on page 141-49	Compares statistics for a table from two timestamps in past and compare the statistics as of that timestamps

**Table 141–2 (Cont.) DBMS\_STATS Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DIFF_TABLE_STATS_IN_PENDING Function</a> on page 141-50	Compares pending statistics and statistics as of a timestamp or statistics from dictionary
<a href="#">DIFF_TABLE_STATS_IN_STATTAB Function</a> on page 141-51	Compares statistics for a table from two different sources
<a href="#">DROP_EXTENDED_STATS Procedure</a> on page 141-52	Drops the statistics entry that is created for the user specified extension
<a href="#">DROP_STAT_TABLE Procedure</a> on page 141-53	Drops a user statistics table created by CREATE_STAT_TABLE
<a href="#">EXPORT_COLUMN_STATS Procedure</a> on page 141-54	Retrieves statistics for a particular column and stores them in the user statistics table identified by <code>stattab</code>
<a href="#">EXPORT_DATABASE_PREFS Procedure</a> on page 141-55	Exports the statistics preferences of all the tables, excluding the tables owned by Oracle
<a href="#">EXPORT_DATABASE_STATS Procedure</a> on page 141-56	Retrieves statistics for all objects in the database and stores them in the user statistics table identified by <code>statown.stattab</code>
<a href="#">EXPORT_DICTIONARY_STATS Procedure</a> on page 141-57	Retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) and stores them in the user statistics table identified by <code>stattab</code>
<a href="#">EXPORT_FIXED_OBJECTS_STATS Procedure</a> on page 141-58	Retrieves statistics for fixed tables and stores them in the user statistics table identified by <code>stattab</code>
<a href="#">EXPORT_INDEX_STATS Procedure</a> on page 141-59	Retrieves statistics for a particular index and stores them in the user statistics table identified by <code>stattab</code>
<a href="#">EXPORT_PENDING_STATS Procedure</a> on page 141-60	Exports the statistics gathered and stored as pending
<a href="#">EXPORT_SCHEMA_PREFS Procedure</a> on page 141-61	Exports the statistics preferences of all the tables owned by the specified owner name
<a href="#">EXPORT_SCHEMA_STATS Procedure</a> on page 141-62	Retrieves statistics for all objects in the schema identified by <code>ownname</code> and stores them in the user statistics table identified by <code>stattab</code>
<a href="#">EXPORT_SYSTEM_STATS Procedure</a> on page 141-63	Retrieves system statistics and stores them in the user statistics table
<a href="#">EXPORT_TABLE_PREFS Procedure</a> on page 141-64	Exports statistics preferences of the specified table in the specified schema into the specified statistics table
<a href="#">EXPORT_TABLE_STATS Procedure</a> on page 141-65	Retrieves statistics for a particular table and stores them in the user statistics table
<a href="#">FLUSH_DATABASE_MONITORING_INFO Procedure</a> on page 141-66	Flushes in-memory monitoring information for all the tables to the dictionary
<a href="#">GATHER_DATABASE_STATS Procedures</a> on page 141-67	Gathers statistics for all objects in the database
<a href="#">GATHER_DICTIONARY_STATS Procedure</a> on page 141-67	Gathers statistics for dictionary schemas 'SYS', 'SYSTEM' and schemas of RDBMS components
<a href="#">GATHER_FIXED_OBJECTS_STATS Procedure</a> on page 141-75	Gathers statistics of fixed objects
<a href="#">GATHER_INDEX_STATS Procedure</a> on page 141-76	Gathers index statistics



**Table 141-2 (Cont.) DBMS\_STATS Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GATHER_SCHEMA_STATS Procedures</a> on page 141-78	Gathers statistics for all objects in a schema
<a href="#">GATHER_SYSTEM_STATS Procedure</a> on page 141-82	Gathers system statistics
<a href="#">GATHER_TABLE_STATS Procedure</a> on page 141-84	Gathers table and column (and index) statistics
<a href="#">GENERATE_STATS Procedure</a> on page 141-88	Generates object statistics from previously collected statistics of related objects
<a href="#">GET_COLUMN_STATS Procedures</a> on page 141-89	Gets all column-related information
<a href="#">GET_INDEX_STATS Procedures</a> on page 141-91	Gets all index-related information
<a href="#">GET_PARAM Function</a> on page 141-94	Gets the default value of parameters of DBMS_STATS procedures [see <a href="#">Deprecated Subprograms</a> on page 141-12]
<a href="#">GET_PREFS Function</a> on page 141-95	Gets the default value of the specified preference
<a href="#">GET_STATS_HISTORY_AVAILABILITY Function</a> on page 141-98	Gets the oldest timestamp where statistics history is available
<a href="#">GET_STATS_HISTORY_RETENTION Function</a> on page 141-99	Returns the current retention value
<a href="#">GET_SYSTEM_STATS Procedure</a> on page 141-100	Gets system statistics from stattab, or from the dictionary if stattab is NULL
<a href="#">GET_TABLE_STATS Procedure</a> on page 141-102	Gets all table-related information
<a href="#">IMPORT_COLUMN_STATS Procedure</a> on page 141-104	Retrieves statistics for a particular column from the user statistics table identified by stattab and stores them in the dictionary
<a href="#">IMPORT_DATABASE_PREFS Procedure</a> on page 141-106	Imports the statistics preferences of all the tables, excluding the tables owned by Oracle
<a href="#">IMPORT_DATABASE_STATS Procedure</a> on page 141-107	Retrieves statistics for all objects in the database from the user statistics table and stores them in the dictionary
<a href="#">IMPORT_DICTIONARY_STATS Procedure</a> on page 141-108	Retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) from the user statistics table and stores them in the dictionary
<a href="#">IMPORT_FIXED_OBJECTS_STATS Procedure</a> on page 141-110	Retrieves statistics for fixed tables from the user statistics table identified by stattab and stores them in the dictionary
<a href="#">IMPORT_INDEX_STATS Procedure</a> on page 141-111	Retrieves statistics for a particular index from the user statistics table identified by stattab and stores them in the dictionary
<a href="#">IMPORT_SCHEMA_PREFS Procedure</a> on page 141-112	Imports the statistics preferences of all the tables owned by the specified owner name
<a href="#">IMPORT_SCHEMA_STATS Procedure</a> on page 141-113	Retrieves statistics for all objects in the schema identified by ownname from the user statistics table and stores them in the dictionary

**Table 141–2 (Cont.) DBMS\_STATS Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">IMPORT_SYSTEM_STATS Procedure</a> on page 141-115	Retrieves system statistics from the user statistics table and stores them in the dictionary
<a href="#">IMPORT_TABLE_PREFS Procedure</a> on page 141-116	Sets the statistics preferences of the specified table in the specified schema
<a href="#">IMPORT_TABLE_STATS Procedure</a> on page 141-117	Retrieves statistics for a particular table from the user statistics table identified by <code>stattab</code> and stores them in the dictionary
<a href="#">LOCK_PARTITION_STATS Procedure</a> on page 141-119	Locks statistics for a partition
<a href="#">LOCK_SCHEMA_STATS Procedure</a> on page 141-120	Locks the statistics of all tables of a schema
<a href="#">LOCK_TABLE_STATS Procedure</a> on page 141-121	Locks the statistics on the table
<a href="#">MERGE_COL_USAGE Procedure</a> on page 141-122	Merges column usage information from a source database, by means of a <code>dblink</code> , into the local database
<a href="#">PREPARE_COLUMN_VALUES Procedures</a> on page 141-123	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using the <a href="#">SEED_COL_USAGE Procedure</a>
<a href="#">PREPARE_COLUMN_VALUES_NVARCHAR2 Procedure</a> on page 141-125	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using the <a href="#">SEED_COL_USAGE Procedure</a>
<a href="#">PREPARE_COLUMN_VALUES_ROWID Procedure</a> on page 141-127	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using the <a href="#">SEED_COL_USAGE Procedure</a>
<a href="#">PUBLISH_PENDING_STATS Procedure</a> on page 141-129	Publishes the statistics gathered and stored as pending
<a href="#">PURGE_STATS Procedure</a> on page 141-130	Purges old versions of statistics saved in the dictionary
<a href="#">RESET_GLOBAL_PREF_DEFAULTS Procedure</a> on page 141-131	Resets the default values of all parameters to Oracle recommended values
<a href="#">RESET_PARAM_DEFAULTS Procedure</a> on page 141-132	Resets global preferences to default values [see <a href="#">Deprecated Subprograms</a> on page 141-12]
<a href="#">RESTORE_DICTIONARY_STATS Procedure</a> on page 141-134	Restores statistics of all dictionary tables (tables of 'SYS', 'SYSTEM' and RDBMS component schemas) as of a specified timestamp
<a href="#">RESTORE_FIXED_OBJECTS_STATS Procedure</a> on page 141-135	Restores statistics of all fixed tables as of a specified timestamp
<a href="#">RESTORE_SCHEMA_STATS Procedure</a> on page 141-136	Restores statistics of all tables of a schema as of a specified timestamp
<a href="#">RESTORE_SYSTEM_STATS Procedure</a> on page 141-137	Restores statistics of all tables of a schema as of a specified timestamp
<a href="#">RESTORE_TABLE_STATS Procedure</a> on page 141-138	Restores statistics of a table as of a specified timestamp ( <code>as_of_timestamp</code> ), as well as statistics of associated indexes and columns

**Table 141–2 (Cont.) DBMS\_STATS Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SEED_COL_USAGE Procedure</a> on page 141-139	Iterates over the SQL statements in the specified SQL tuning set, compiles them and seeds column usage information for the columns that appear in these statements
<a href="#">SET_COLUMN_STATS Procedures</a> on page 141-140	Sets column-related information
<a href="#">SET_DATABASE_PREFS Procedure</a> on page 141-142	Sets the statistics preferences of all the tables, excluding the tables owned by Oracle
<a href="#">SET_GLOBAL_PREFS Procedure</a> on page 141-139	Sets the global statistics preferences
<a href="#">SET_INDEX_STATS Procedures</a> on page 141-148	Sets index-related information
<a href="#">SET_PARAM Procedure</a> on page 141-151	Sets default values for parameters of DBMS_STATS procedures [see <a href="#">Deprecated Subprograms</a> on page 141-12]
<a href="#">SET_SCHEMA_PREFS Procedure</a> on page 141-153	Sets the statistics preferences of all the tables owned by the specified owner name
<a href="#">SET_SYSTEM_STATS Procedure</a> on page 141-154	Sets system statistics
<a href="#">SET_TABLE_PREFS Procedure</a> on page 141-156	Sets the statistics preferences of the specified table in the specified schema
<a href="#">SET_TABLE_STATS Procedure</a> on page 141-159	Sets table-related information
<a href="#">SHOW_EXTENDED_STATS_NAME Function</a> on page 141-161	Returns the name of the virtual column that is created for the user-specified extension
<a href="#">UNLOCK_PARTITION_STATS Procedure</a> on page 141-162	Unlocks the statistics for a partition
<a href="#">UNLOCK_SCHEMA_STATS Procedure</a> on page 141-163	Unlocks the statistics on all the table in a schema
<a href="#">UNLOCK_TABLE_STATS Procedure</a> on page 141-164	Unlocks the statistics on the table
<a href="#">UPGRADE_STAT_TABLE Procedure</a> on page 141-165	Upgrades user statistics on an older table

## ALTER\_STATS\_HISTORY\_RETENTION Procedure

This procedure changes the statistics history retention value. Statistics history retention is used by both the automatic purge and [PURGE\\_STATS Procedure](#).

### Syntax

```
DBMS_STATS.ALTER_STATS_HISTORY_RETENTION (  
    retention          IN          NUMBER);
```

### Parameters

**Table 141–3 ALTER\_STATS\_HISTORY\_RETENTION Procedure Parameters**

Parameter	Description
retention	<p>The retention time in days. The statistics history will be retained for at least these many number of days. The valid range is [1,365000]. Also you can use the following values for special purposes:</p> <ul style="list-style-type: none"><li>▪ -1 :Statistics history is never purged by automatic purge</li><li>▪ 0 : Old statistics are never saved. The automatic purge will delete all statistics history</li><li>▪ NULL : Change statistics history retention to default value</li></ul>

### Usage Notes

To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

### Exceptions

ORA-20000: Insufficient privileges

## CONVERT\_RAW\_VALUE Procedures

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The `minval` and `maxval` fields of the `StatRec` structure as filled in by `GET_COLUMN_STATS` or `PREPARE_COLUMN_VALUES` are appropriate values for input.

### Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE (
    rawval     RAW,
    resval OUT BINARY_FLOAT);
```

```
DBMS_STATS.CONVERT_RAW_VALUE (
    rawval     RAW,
    resval OUT BINARY_DOUBLE);
```

```
DBMS_STATS.CONVERT_RAW_VALUE (
    rawval     RAW,
    resval OUT DATE);
```

```
DBMS_STATS.CONVERT_RAW_VALUE (
    rawval     RAW,
    resval OUT NUMBER);
```

```
DBMS_STATS.CONVERT_RAW_VALUE (
    rawval     RAW,
    resval OUT VARCHAR2);
```

### Pragmas

```
pragma restrict_references(convert_raw_value, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 141–4** *CONVERT\_RAW\_VALUE Procedure Parameters*

Parameter	Description
<code>rawval</code>	The raw representation of a column minimum or maximum datatype-specific output parameters
<code>resval</code>	The converted, type-specific value

## CONVERT\_RAW\_VALUE\_NVARCHAR Procedure

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The `minval` and `maxval` fields of the `StatRec` structure as filled in by `GET_COLUMN_STATS` or `PREPARE_COLUMN_VALUES` are appropriate values for input.

### Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE_NVARCHAR (
    rawval      RAW,
    resval OUT NVARCHAR2);
```

### Pragmas

```
pragma restrict_references(convert_raw_value_nvarchar, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 141–5** *CONVERT\_RAW\_VALUE\_NVARCHAR Procedure Parameters*

Parameter	Description
<code>rawval</code>	The raw representation of a column minimum or maximum datatype-specific output parameters
<code>resval</code>	The converted, type-specific value

## CONVERT\_RAW\_VALUE\_ROWID Procedure

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The minval and maxval fields of the StatRec structure as filled in by GET\_COLUMN\_STATS or PREPARE\_COLUMN\_VALUES are appropriate values for input.

### Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE_ROWID (
    rawval      RAW,
    resval OUT ROWID);
```

### Pragmas

```
pragma restrict_references(convert_raw_value_rowid, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 141–6** *CONVERT\_RAW\_VALUE\_ROWID Procedure Parameters*

Parameter	Description
rawval	The raw representation of a column minimum or maximum datatype-specific output parameters
resval	The converted, type-specific value

## COPY\_TABLE\_STATS Procedure

This procedure copies the statistics of the source [sub] partition to the destination [sub] partition. It also copies statistics of all dependent object such as columns and local indexes. If the statistics for source are not available then nothing is copied. It can optionally scale the statistics (such as the number of blks, or number of rows) based on the given `scale_factor`.

### Syntax

```
DBMS_STATS.COPY_TABLE_STATS (
  ownname          VARCHAR2,
  tablename        VARCHAR2,
  srcpartname      VARCHAR2,
  dstpartname      VARCHAR2,
  scale_factor     VARCHAR2 DEFAULT 1,
  flags            NUMBER DEFAULT,
  force            BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–7 COPY\_TABLE\_STATS Procedure Parameters**

Parameter	Description
<code>ownname</code>	Schema of the table of source and destination [sub] partitions
<code>tablename</code>	Table name of source and destination [sub] partitions
<code>srcpartname</code>	Source [sub] partition
<code>dstpartname</code>	Destination [sub] partition
<code>scale_factor</code>	Scale factor to scale <code>nblks</code> , <code>nrows</code> etc. in <code>dstpartname</code>
<code>flags</code>	For internal Oracle use (should be left as <code>NULL</code> )
<code>force</code>	When value of this argument is <code>TRUE</code> , copy statistics even if locked

### Exceptions

ORA-20000: Invalid partition name

ORA-20001: Bad input value

### Usage Notes

This procedure updates the minimum and maximum values of destination partition for the first partitioning column as follows:

- If the partitioning type is `HASH` the minimum and maximum values of the destination partition are same as that of the source partition.
- If the partitioning type is `LIST` then
  - if the destination partition is a `NOT DEFAULT` partition then
    - \* the minimum value of the destination partition is set to the minimum value of the value list that describes the destination partition
    - \* the maximum value of the destination partition is set to the maximum value of the value list that describes the destination partition



- alternatively, if the destination partition is a `DEFAULT` partition, then
  - \* the minimum value of the destination partition is set to the minimum value of the source partition
  - \* the maximum value of the destination partition is set to the maximum value of the source partition
- If the partitioning type is `RANGE` then
  - the minimum value of the destination partition is set to the high bound of previous partition
  - the maximum value of the destination partition is set to the high bound of the destination partition unless the high bound of the destination partition is `MAXVALUE`, in which case the maximum value of the destination partition is set to the high bound of the previous partition

Note that if the destination partition is the first partition then minimum values are equal to maximum values.

The procedure is extended to handle the second and subsequent key columns with two additional rules (which do not apply to the first key column) :

- If the source partition column's minimum value is equal to its maximum value, and both are equal to the source partition's lower bound, and it has a single distinct value, then the destination partition column's minimum and maximum values are both set to the destination partition's lower bound.
- If the above condition does not apply, the destination partition column's maximum value is set to the greater of the destination partition upper bound and the source partition column's maximum value, with the following exception: if the destination partition is `D` and its preceding partition is `D-1` and the key column to be adjusted is `Cn`, the maximum value for `Cn` is set to the upper bound of `D` (ignoring the maximum value of the source partition column) provided that the upper bounds of the previous key column `Cn-1` are the same in partitions `D` and `D-1`.

## CREATE\_EXTENDED\_STATS Function

This function creates a column statistics entry in the system for a user specified column group or an expression in a table. Statistics for this extension will be gathered when user or auto statistics gathering job gathers statistics for the table. We call statistics for such an extension, "extended statistics". This function returns the name of this newly created entry for the extension.

### Syntax

```
DBMS_STATS.CREATE_EXTENDED_STATS (
    ownname    VARCHAR2,
    tabname    VARCHAR2,
    extension  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 141–8 CREATE\_EXTENDED\_STATS Function Parameters**

Parameter	Description
ownname	Owner name of a table
tabname	Name of the table
extension	Can be either a column group or an expression. Suppose the specified table has two column <i>c1</i> , <i>c2</i> . An example column group can be "( <i>c1</i> , <i>c2</i> )" and an example expression can be "( <i>c1</i> + <i>c2</i> )".

### Return Values

This function returns the name of this newly created entry for the extension.

### Exceptions

ORA-20000: Insufficient privileges / creating extension is not supported

ORA-20001: Error when processing extension

ORA-20007: Extension already exists

ORA-20008: Reached the upper limit on number of extensions

### Usage Notes

There are nine restrictions on the extension:

- The extension cannot contain a virtual column.
- Extensions cannot be created on tables owned by *SYS*.
- Extensions cannot be created on cluster tables, index organized tables, temporary tables or external tables.
- The total number of extensions in a table cannot be greater than a maximum of (20, 10% of number of non-virtual columns in the table).
- The number of columns in a column group must be in the range [2, 32].
- A column can not appear more than once in a column group.
- A column group can not contain expressions.

- An expression must contain at least one column.
- An expression can not contain a subquery.
- The COMPATIBLE parameter needs to be 11.0.0.0 or greater

## CREATE\_STAT\_TABLE Procedure

This procedure creates a table with name `stattab` in `ownname`'s schema which is capable of holding statistics. The columns and types that compose this table are not relevant as it should be accessed solely through the procedures in this package.

### Syntax

```
DBMS_STATS.CREATE_STAT_TABLE (  
    ownname VARCHAR2,  
    stattab VARCHAR2,  
    tblspace VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–9 CREATE\_STAT\_TABLE Procedure Parameters**

Parameter	Description
<code>ownname</code>	Name of the schema
<code>stattab</code>	Name of the table to create. This value should be passed as the <code>stattab</code> parameter to other procedures when the user does not want to modify the dictionary statistics directly.
<code>tblspace</code>	Tablespace in which to create the statistics tables. If none is specified, then they are created in the user's default tablespace.

### Exceptions

ORA-20000: Table already exists or insufficient privileges

ORA-20001: Tablespace does not exist

## DELETE\_COLUMN\_STATS Procedure

This procedure deletes column-related statistics.

### Syntax

```
DBMS_STATS.DELETE_COLUMN_STATS (
  ownname      VARCHAR2,
  tabname      VARCHAR2,
  colname      VARCHAR2,
  partname     VARCHAR2 DEFAULT NULL,
  statab       VARCHAR2 DEFAULT NULL,
  statid       VARCHAR2 DEFAULT NULL,
  cascade_parts BOOLEAN DEFAULT TRUE,
  statown      VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
  force        BOOLEAN DEFAULT FALSE,
  col_stat_type VARCHAR2 DEFAULT 'ALL');
```

### Parameters

**Table 141–10** *DELETE\_COLUMN\_STATS Procedure Parameters*

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
colname	Name of the column or extension
partname	Name of the table partition for which to delete the statistics. If the table is partitioned and if <code>partname</code> is <code>NULL</code> , then global column statistics are deleted.
statab	User statistics table identifier describing from where to delete the statistics. If <code>statab</code> is <code>NULL</code> , then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within <code>statab</code> (Only pertinent if <code>statab</code> is not <code>NULL</code> ).
cascade_parts	If the table is partitioned and if <code>partname</code> is <code>NULL</code> , then setting this to <code>true</code> causes the deletion of statistics for this column for all underlying partitions as well.
statown	Schema containing <code>statab</code> (if different than <code>ownname</code> )
no_invalidate	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	When value of this argument is <code>TRUE</code> , deletes column statistics even if locked

**Table 141–10 (Cont.) DELETE\_COLUMN\_STATS Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
col_stat_type	Type of column statistics to be deleted. This argument takes the following values: <ul style="list-style-type: none"><li>■ HISTOGRAM - delete column histogram only</li><li>■ ALL - delete base column statistics and histogram</li></ul>

**Exceptions**

ORA-20000: Object does not exist or insufficient privileges

ORA-20005: Object statistics are locked

## DELETE\_DATABASE\_PREFS Procedure

This procedure is used to delete the statistics preferences of all the tables, excluding the tables owned by Oracle. These tables can be included by passing `TRUE` for the `add_sys` parameter.

### Syntax

```
DBMS_STATS.DELETE_DATABASE_PREFS (
  pname          IN   VARCHAR2,
  add_sys        IN   BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–11** *DELETE\_DATABASE\_PREFS Procedure Parameters*

Parameter	Description
<code>pname</code>	<p>Preference name. The default value for following parameters can be deleted:</p> <ul style="list-style-type: none"> <li>■ CASCADE</li> <li>■ DEGREE</li> <li>■ ESTIMATE_PERCENT</li> <li>■ METHOD_OPT</li> <li>■ NO_INVALIDATE</li> <li>■ GRANULARITY</li> <li>■ PUBLISH</li> <li>■ INCREMENTAL</li> <li>■ STALE_PERCENT</li> </ul> <p>· CASCADE - The value determines whether or not index statistics are collected as part of gathering table statistics</p> <p>· DEGREE - The value determines degree of parallelism used for gathering statistics</p> <p>· ESTIMATE_PERCENT - The value determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to have Oracle determine the appropriate sample size for good statistics. This is the default.</p>

**Table 141–11 (Cont.) DELETE\_DATABASE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>METHOD_OPT - The value controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns</li> <li>- column_name : name of a column</li> <li>- extension : can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO.</p>
.	<p>NO_INVALIDATE - The value controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default.</p>
.	<p>GRANULARITY - The value determines granularity of statistics to collect (only pertinent if the table is partitioned)</p> <p>'ALL' - gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - gathers global statistics</p> <p>'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - gathers partition-level statistics</p> <p>'SUBPARTITION' - gathers subpartition-level statistics</p>



**Table 141-11 (Cont.) DELETE\_DATABASE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>PUBLISH - This value determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
.	<p>INCREMENTAL - This value determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ the INCREMENTAL value for the partitioned table is set to TRUE;</li> <li>■ the PUBLISH value for the partitioned table is set to TRUE;</li> <li>■ the user specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table.</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
.	<p>STALE_PERCENT - Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%. Note that if you set stale_percent to zero the AUTO STATS gathering job will gather statistics for this table every time a row in the table is modified.</p>
add_sys	Value TRUE will include the Oracle-owned tables

## Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Invalid or Illegal input values

## Usage Notes

- To run this procedure, you need to have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.
- All pname arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

## Examples

```
DBMS_STATS.DELETE_DATABASE_PREFS ('CASCADE', FALSE);
DBMS_STATS.DELETE_DATABASE_PREFS ('ESTIMATE_PERCENT', TRUE);
```

## DELETE\_DATABASE\_STATS Procedure

This procedure deletes statistics for all the tables in a database.

### Syntax

```
DBMS_STATS.DELETE_DATABASE_STATS (
  statab          VARCHAR2 DEFAULT NULL,
  statid          VARCHAR2 DEFAULT NULL,
  statown         VARCHAR2 DEFAULT NULL,
  no_invalidate   BOOLEAN   DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
  force           BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 141–12** *DELETE\_DATABASE\_STATS Procedure Parameters*

Parameter	Description
statab	User statistics table identifier describing from where to delete the statistics. If <code>statab</code> is <code>NULL</code> , then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within <code>statab</code> (Only pertinent if <code>statab</code> is not <code>NULL</code> )
statown	Schema containing <code>statab</code> (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	When the value of this argument is <code>TRUE</code> , deletes statistics of tables in a database even if they are locked

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

## DELETE\_DICTIONARY\_STATS Procedure

This procedure deletes statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas).

### Syntax

```
DBMS_STATS.DELETE_DICTIONARY_STATS (
  statab          VARCHAR2 DEFAULT NULL,
  statid          VARCHAR2 DEFAULT NULL,
  statown         VARCHAR2 DEFAULT NULL,
  no_invalidate  BOOLEAN   DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
  force           BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 141–13** DELETE\_DICTIONARY\_STATS Procedure Parameters

Parameter	Description
statab	User statistics table identifier describing from where to delete the statistics. If statab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within statab (Only pertinent if statab is not NULL)
statown	Schema containing statab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>
force	When the value of this argument is TRUE, deletes statistics of tables in a database even if they are locked

### Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

## DELETE\_FIXED\_OBJECTS\_STATS Procedure

This procedure deletes statistics of all fixed tables.

### Syntax

```
DBMS_STATS.DELETE_FIXED_OBJECTS_STATS (
  statab      VARCHAR2 DEFAULT NULL,
  statid      VARCHAR2 DEFAULT NULL,
  statown     VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
  force       BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–14** *DELETE\_FIXED\_OBJECTS\_STATS Procedure Parameters*

Parameter	Description
statab	The user statistics table identifier describing from where to delete the current statistics. If <code>statab</code> is <code>NULL</code> , the statistics will be deleted directly in the dictionary.
statid	The (optional) identifier to associate with these statistics within <code>statab</code> . This only applies if <code>statab</code> is not <code>NULL</code> .
statown	Schema containing <code>statab</code> (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>
force	Ignores the statistics lock on objects and deletes the statistics if set to <code>TRUE</code>

### Usage Notes

You must have the `SYSDBA` or `ANALYZE ANY DICTIONARY` system privilege to execute this procedure.

### Exceptions

ORA-20000: Insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

## DELETE\_INDEX\_STATS Procedure

This procedure deletes index-related statistics.

### Syntax

```
DBMS_STATS.DELETE_INDEX_STATS (
  ownname          VARCHAR2,
  indname          VARCHAR2,
  partname         VARCHAR2 DEFAULT NULL,
  statab          VARCHAR2 DEFAULT NULL,
  statid          VARCHAR2 DEFAULT NULL,
  cascade_parts   BOOLEAN  DEFAULT TRUE,
  statown         VARCHAR2 DEFAULT NULL,
  no_invalidate   BOOLEAN  DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
  force           BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–15** *DELETE\_INDEX\_STATS Procedure Parameters*

Parameter	Description
ownname	Name of the schema
indname	Name of the index
partname	Name of the index partition for which to delete the statistics. If the index is partitioned and if partname is NULL, then index statistics are deleted at the global level.
statab	User statistics table identifier describing from where to delete the statistics. If statab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within statab (Only pertinent if statab is not NULL)
cascade_parts	If the index is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this index for all underlying partitions as well
statown	Schema containing statab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	When value of this argument is TRUE, deletes index statistics even if locked

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20005: Object statistics are locked

## DELETE\_PENDING\_STATS Procedure

This procedure is used to delete the pending statistics that have been collected but have not been published.

### Syntax

```
DBMS_STATS.DELETE_PENDING_STATS (  
    ownname    IN  VARCHAR2  DEFAULT USER,  
    tabname    IN  VARCHAR2);
```

### Parameters

**Table 141–16** *DELETE\_PENDING\_STATS Procedure Parameters*

Parameter	Description
ownname	Owner name
tabname	Table name

### Exceptions

ORA-20000: Insufficient privileges

### Usage Notes

- If the parameter `tabname` is `NULL` delete applies to all tables of the specified schema.
- The default owner/schema is the user who runs the procedure.
- To run this procedure, you need to have the same privilege for gathering statistics on the tables that will be touched by this procedure.

### Examples

```
DBMS_STATS.DELETE_PENDING_STATS('SH', 'SALES');
```

## DELETE\_SCHEMA\_PREFS Procedure

This procedure is used to delete the statistics preferences of all the tables owned by the specified owner name.

### Syntax

```
DBMS_STATS.DELETE_SCHEMA_PREFS (
  ownname  IN  VARCHAR2,
  pname    IN  VARCHAR2);
```

### Parameters

**Table 141–17** *DELETE\_SCHEMA\_PREFS Procedure Parameters*

Parameter	Description
ownname	Owner name
pname	Preference name. The default value for following parameters can be deleted: <ul style="list-style-type: none"> <li>▪ CASCADE</li> <li>▪ DEGREE</li> <li>▪ ESTIMATE_PERCENT</li> <li>▪ METHOD_OPT</li> <li>▪ NO_INVALIDATE</li> <li>▪ GRANULARITY</li> <li>▪ PUBLISH</li> <li>▪ INCREMENTAL</li> <li>▪ STALE_PERCENT</li> </ul>
.	CASCADE - Determines whether or not index statistics are collected as part of gathering table statistics
.	DEGREE - Determines degree of parallelism used for gathering statistics
.	ESTIMATE_PERCENT - Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.

**Table 141–17 (Cont.) DELETE\_SCHEMA\_PREFS Procedure Parameters**

Parameter	Description
.	<p>METHOD_OPT - Controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns</li> <li>- column_name : name of a column</li> <li>- extension: can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO.</p>
.	<p>NO_INVALIDATE - Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
.	<p>GRANULARITY - The value determines granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - gathers global statistics</p> <p>'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - gathers partition-level statistics</p> <p>'SUBPARTITION' - gathers subpartition-level statistics.</p>



**Table 141-17 (Cont.) DELETE\_SCHEMA\_PREFS Procedure Parameters**

Parameter	Description
PUBLISH	<p>PUBLISH - This value determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
INCREMENTAL	<p>INCREMENTAL - This value determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ the INCREMENTAL value for the partitioned table is set to TRUE</li> <li>■ the PUBLISH value for the partitioned table is set to TRUE</li> <li>■ the user specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
STALE_PERCENT	<p>STALE_PERCENT - Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%. Note that if you set stale_percent to zero the AUTO_STATS gathering job will gather statistics for this table every time a row in the table is modified.</p>

## Exceptions

ORA-20000: Insufficient privileges / Schema "<schema>" does not exist

ORA-20001: Invalid or Illegal input values

## Usage Notes

- To run this procedure, you need to connect as owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

## Examples

```
DBMS_STATS.DELETE_SCHEMA_PREFS('SH', 'CASCADE');
DBMS_STATS.DELETE_SCHEMA_PREFS('SH', 'ESTIMATE_PERCENT');
DBMS_STATS.DELETE_SCHEMA_PREFS('SH', 'DEGREE');
```

## DELETE\_SCHEMA\_STATS Procedure

This procedure deletes statistics for an entire schema.

### Syntax

```
DBMS_STATS.DELETE_SCHEMA_STATS (
  ownname          VARCHAR2,
  statab           VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
  force            BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–18** *DELETE\_SCHEMA\_STATS Procedure Parameters*

Parameter	Description
ownname	Name of the schema
statab	User statistics table identifier describing from where to delete the statistics. If <code>statab</code> is <code>NULL</code> , then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within <code>statab</code> (Only pertinent if <code>statab</code> is not <code>NULL</code> )
statown	Schema containing <code>statab</code> (if different than <code>ownname</code> )
no_invalidate	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	When value of this argument is <code>TRUE</code> , deletes statistics of tables in a schema even if locked

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

## DELETE\_SYSTEM\_STATS Procedure

This procedure deletes workload statistics (collected using the 'INTERVAL' or 'START' and 'STOP' options) and resets the default to noworkload statistics (collected using 'NOWORKLOAD' option) if `stattab` is not specified. If `stattab` is specified, the subprogram deletes all system statistics with the associated `statid` from the `stattab`.

### Syntax

```
DBMS_STATS.DELETE_SYSTEM_STATS (
  stattab      VARCHAR2 DEFAULT NULL,
  statid       VARCHAR2 DEFAULT NULL,
  statown      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–19** *DELETE\_SYSTEM\_STATS Procedure Parameters*

Parameter	Description
<code>stattab</code>	Identifier of the user statistics table where the statistics will be saved
<code>statid</code>	Optional identifier associated with the statistics saved in the <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table; may need to be upgraded

## DELETE\_TABLE\_PREFS Procedure

This procedure is used to delete the statistics preferences of the specified table in the specified schema.

### Syntax

```
DBMS_STATS.DELETE_TABLE_PREFS (
    ownname    IN  VARCHAR2,
    tablename  IN  VARCHAR2,
    pname      IN  VARCHAR2);
```

### Parameters

**Table 141–20** *DELETE\_TABLE\_PREFS Procedure Parameters*

Parameter	Description
ownname	Owner name
tablename	Table name
pname	Preference name. The default value for following preferences can be deleted: <ul style="list-style-type: none"> <li>▪ CASCADE</li> <li>▪ DEGREE</li> <li>▪ ESTIMATE_PERCENT</li> <li>▪ METHOD_OPT</li> <li>▪ NO_INVALIDATE</li> <li>▪ GRANULARITY</li> <li>▪ PUBLISH</li> <li>▪ INCREMENTAL</li> <li>▪ STALE_PERCENT</li> </ul>
.	CASCADE - Determines whether or not index statistics are collected as part of gathering table statistics
.	DEGREE - Determines degree of parallelism used for gathering statistics
.	ESTIMATE_PERCENT - Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.

**Table 141–20 (Cont.) DELETE\_TABLE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>METHOD_OPT - The value controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms.</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> <li>- column_name : name of a column</li> <li>- extension : can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO.</p>
.	<p>NO_INVALIDATE - Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default.</p>
.	<p>GRANULARITY - Determines granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics.</p>

**Table 141–20 (Cont.) DELETE\_TABLE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>PUBLISH - Determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
.	<p>INCREMENTAL - Determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ INCREMENTAL value for the partitioned table is set to TRUE</li> <li>■ PUBLISH value for the partitioned table is set to TRUE;</li> <li>■ User specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table.</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
.	<p>STALE_PERCENT - Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%.</p>
pvalue	<p>Preference value. If NULL is specified, it will set the Oracle default value.</p>

## Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Invalid or Illegal input values

## Usage Notes

- To run this procedure, you need to connect as owner of the table, be granted ANALYZE privilege on the table, or ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

## Examples

```
DBMS_STATS.DELETE_TABLE_PREFS('SH', 'SALES', 'CASCADE');
DBMS_STATS.DELETE_TABLE_PREFS('SH', 'SALES', 'DEGREE');
```

## DELETE\_TABLE\_STATS Procedure

This procedure deletes table-related statistics.

### Syntax

```
DBMS_STATS.DELETE_TABLE_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    statab          VARCHAR2 DEFAULT NULL,
    statid          VARCHAR2 DEFAULT NULL,
    cascade_parts    BOOLEAN  DEFAULT TRUE,
    cascade_columns  BOOLEAN  DEFAULT TRUE,
    cascade_indexes  BOOLEAN  DEFAULT TRUE,
    statown         VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
    force           BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–21** *DELETE\_TABLE\_STATS Procedure Parameters*

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if <code>partname</code> is <code>NULL</code> , then the statistics are retrieved from the global table level.
statab	User statistics table identifier describing from where to retrieve the statistics. If <code>statab</code> is <code>NULL</code> , then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within <code>statab</code> (Only pertinent if <code>statab</code> is not <code>NULL</code> )
cascade_parts	If the table is partitioned and if <code>partname</code> is <code>NULL</code> , then setting this to <code>TRUE</code> causes the deletion of statistics for this table for all underlying partitions as well
cascade_columns	Indicates that <code>DELETE_COLUMN_STATS</code> should be called for all underlying columns (passing the <code>cascade_parts</code> parameter)
cascade_indexes	Indicates that <code>DELETE_INDEX_STATS</code> should be called for all underlying indexes (passing the <code>cascade_parts</code> parameter)
statown	Schema containing <code>statab</code> (if different than <code>ownname</code> )
no_invalidate	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	When value of this argument is <code>TRUE</code> , deletes table statistics even if locked

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

ORA-20005: Object statistics are locked



## DIFF\_TABLE\_STATS\_IN\_HISTORY Function

This function can be used to compare statistics for a table from two timestamps in past and compare the statistics as of that timestamps.

### Syntax

```
DBMS_STATS.DIFF_TABLE_STATS_IN_HISTORY(
  ownname      IN VARCHAR2,
  tablename    IN VARCHAR2,
  time1        IN TIMESTAMP WITH TIME ZONE,
  time2        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  pctthreshold IN NUMBER DEFAULT 10)
RETURN DiffRepTab pipelined;
```

### Parameters

**Table 141–22** *DIFF\_TABLE\_STATS\_IN\_HISTORY Function Parameters*

Parameter	Description
ownname	Owner of the table. Specify NULL for current schema.
tablename	Table for which statistics are to be compared
time1	First timestamp 1
time2	Second timestamp 2
pctthreshold	The function reports difference in statistics only if it exceeds this limit. The default value is 10.

### Usage Notes

If the second timestamp is NULL, the function compares the current statistics in dictionary with the statistics as of the other timestamp.

## DIFF\_TABLE\_STATS\_IN\_PENDING Function

This function compares pending statistics and statistics as of a timestamp or statistics from dictionary.

### Syntax

```
DBMS_STATS.DIFF_TABLE_STATS_IN_PENDING(  
    ownname      IN VARCHAR2,  
    tabname      IN VARCHAR2,  
    timestamp    IN TIMESTAMP WITH TIME ZONE,  
    pctthreshold IN NUMBER DEFAULT 10)  
RETURN DiffRepTab pipelined;
```

### Parameters

**Table 141–23** DIFF\_TABLE\_STATS\_IN\_PENDING Function Parameters

Parameter	Description
ownname	Owner of the table. Specify NULL for current schema.
tabname	Table for which statistics are to be compared
timestamp	Time stamp to get statistics from the history
pctthreshold	The function reports difference in statistics only if it exceeds this limit. The default value is 10.

### Usage Notes

If the second timestamp is NULL, the function compares the current statistics in dictionary with the statistics as of the other timestamp.

## DIFF\_TABLE\_STATS\_IN\_STATTAB Function

This function can be used to compare statistics for a table from two different sources. The statistics can be drawn from

- two different user statistics tables
- a single user statistics table containing 2 sets of statistics that can be identified using `statids`
- a user statistics table and dictionary

The function also compares the statistics of the dependent objects (indexes, columns, partitions) as well. It displays statistics of the object(s) from both sources if the difference between those statistics exceeds a certain threshold (%). The threshold can be specified as an argument to the function. The statistics corresponding to the first source (`stattab1` or `time1`) will be used as basis for computing the difference percentage.

### Syntax

```
DBMS_STATS.DIFF_TABLE_STATS_IN_STATTAB(
    ownname          IN VARCHAR2,
    tabname          IN VARCHAR2,
    stattab1         IN VARCHAR2,
    stattab2         IN VARCHAR2 DEFAULT NULL,
    pctthreshold     IN NUMBER   DEFAULT 10,
    statid1          IN VARCHAR2 DEFAULT NULL,
    statid2          IN VARCHAR2 DEFAULT NULL,
    stattab1own      IN VARCHAR2 DEFAULT NULL,
    stattab2own      IN VARCHAR2 DEFAULT NULL)
RETURN DiffRepTab pipelined;
```

### Parameters

**Table 141–24** *DIFF\_TABLE\_STATS\_IN\_STATTAB Function Parameters*

Parameter	Description
<code>ownname</code>	Owner of the table. Specify NULL for current schema.
<code>tabname</code>	Table for which statistics are to be compared
<code>stattab1</code>	User statistics table 1
<code>stattab2</code>	User statistics table 2. If NULL, statistics in <code>stattab1</code> is compared with current statistics in dictionary. This is the default. Specify same table as <code>stattab1</code> to compare two sets within the statistics table (see <code>statid</code> below).
<code>pctthreshold</code>	The function reports difference in statistics only if it exceeds this limit. The default value is 10.
<code>statid1</code>	(optional) Identifies statistics set within <code>stattab1</code> .
<code>statid2</code>	(optional) Identifies statistics set within <code>stattab2</code>
<code>stattab1own</code>	Schema containing <code>stattab1</code> (if other than <code>ownname</code> )
<code>stattab2own</code>	Schema containing <code>stattab2</code> (if other than <code>ownname</code> )

## DROP\_EXTENDED\_STATS Procedure

This function drops the statistics entry that is created for the user specified extension. This cancels the effects of the [CREATE\\_EXTENDED\\_STATS Function](#).

### Syntax

```
DBMS_STATS.DROP_EXTENDED_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2,  
    extension  VARCHAR2);
```

### Parameters

**Table 141–25 DROP\_EXTENDED\_STATS Procedure Parameters**

Parameter	Description
ownname	Owner name of a table
tabname	Name of the table
extension	Can be either a column group or an expression. Suppose the specified table has two column c1, c2. An example column group can be "(c1, c2)" and an example expression can be "(c1 + c2)".

### Usage Notes

If no extended statistics set is created for the extension, this function throws an error.

### Exceptions

- ORA-20000: Insufficient privileges or extension does not exist
- ORA-20001: Error when processing extension

## DROP\_STAT\_TABLE Procedure

This procedure drops a user statistics table.

### Syntax

```
DBMS_STATS.DROP_STAT_TABLE (  
    ownname VARCHAR2,  
    stattab VARCHAR2);
```

### Parameters

**Table 141-26** *DROP\_STAT\_TABLE Procedure Parameters*

Parameter	Description
ownname	Name of the schema
stattab	User statistics table identifier

### Exceptions

ORA-20000: Table does not exists or insufficient privileges.

## EXPORT\_COLUMN\_STATS Procedure

This procedure retrieves statistics for a particular column and stores them in the user statistics table identified by *stattab*.

### Syntax

```
DBMS_STATS.EXPORT_COLUMN_STATS (
  ownname  VARCHAR2,
  tabname  VARCHAR2,
  colname  VARCHAR2,
  partname VARCHAR2 DEFAULT NULL,
  stattab  VARCHAR2,
  statid   VARCHAR2 DEFAULT NULL,
  statown  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–27 EXPORT\_COLUMN\_STATS Procedure Parameters**

Parameter	Description
<i>ownname</i>	Name of the schema
<i>tabname</i>	Name of the table to which this column belongs
<i>colname</i>	Name of the column or extension
<i>partname</i>	Name of the table partition. If the table is partitioned and if <i>partname</i> is <i>NULL</i> , then global and partition column statistics are exported.
<i>stattab</i>	User statistics table identifier describing where to store the statistics
<i>statid</i>	Identifier (optional) to associate with these statistics within <i>stattab</i>
<i>statown</i>	Schema containing <i>stattab</i> (if different than <i>ownname</i> )

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## EXPORT\_DATABASE\_PREFS Procedure

This procedure is used to export the statistics preferences of all the tables, excluding the tables owned by Oracle. These tables can be included by passing `TRUE` for the `add_sys` parameter.

### Syntax

```
DBMS_STATS.EXPORT_DATABASE_PREFS (
  stattab   IN  VARCHAR2,
  statid    IN  VARCHAR2 DEFAULT NULL,
  statown   IN  VARCHAR2 DEFAULT NULL,
  add_sys   IN  BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–28 EXPORT\_DATABASE\_PREFS Procedure Parameters**

Parameter	Description
<code>stattab</code>	Statistics table name to where statistics should be exported
<code>statid</code>	(Optional) Identifier to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if other than <code>ownname</code> )
<code>add_sys</code>	Value <code>TRUE</code> will include the Oracle-owned tables

### Exceptions

ORA-20000: Insufficient privileges

### Usage Notes

- To run this procedure, you need to have the `SYSDBA` role, or both `ANALYZE ANY DICTIONARY` and `ANALYZE ANY` system privileges.
- All arguments are of type `VARCHAR2` and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.EXPORT_DATABASE_PREFS('STATTAB', statown=>'SH');
```

## EXPORT\_DATABASE\_STATS Procedure

This procedure retrieves statistics for all objects in the database and stores them in the user statistics tables identified by `statown.stattab`.

### Syntax

```
DBMS_STATS.EXPORT_DATABASE_STATS (
  stattab          VARCHAR2,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  stat_category    VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–29 EXPORT\_DATABASE\_STATS Procedure Parameters**

Parameter	Description
<code>stattab</code>	User statistics table identifier describing where to store the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)
<code>stat_category</code>	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>▪ 'SYNOPSSES' - auxiliary statistics information gathered on a partitioned table when <code>incremental</code> is <code>TRUE</code></li> </ul>

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.



## EXPORT\_DICTIONARY\_STATS Procedure

This procedure retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) and stores them in the user statistics table identified by `stattab`.

### Syntax

```
DBMS_STATS.EXPORT_DICTIONARY_STATS (
  stattab          VARCHAR2,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  stat_category    VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–30 EXPORT\_DICTIONARY\_STATS Procedure Parameters**

Parameter	Description
<code>stattab</code>	User statistics table identifier describing where to store the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)
<code>stat_category</code>	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>▪ 'SYNOPSISSES' - auxiliary statistics information gathered on a partitioned table when incremental is TRUE</li> </ul>

### Usage Notes

- You must have the `SYSDBA` or `ANALYZE ANY DICTIONARY` and `ANALYZE ANY SYSTEM` privilege to execute this procedure.
- Oracle does not support export or import of statistics across databases of different character sets.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

## EXPORT\_FIXED\_OBJECTS\_STATS Procedure

This procedure retrieves statistics for fixed tables and stores them in the user statistics table identified by `stattab`.

### Syntax

```
DBMS_STATS.EXPORT_FIXED_OBJECTS_STATS (  
    stattab VARCHAR2,  
    statid  VARCHAR2 DEFAULT NULL,  
    statown VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–31 EXPORT\_FIXED\_OBJECTS\_STATS Procedure Parameters**

Parameter	Description
<code>stattab</code>	User statistics table identifier describing where to store the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table, may need to upgrade it

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## EXPORT\_INDEX\_STATS Procedure

This procedure retrieves statistics for a particular index and stores them in the user statistics table identified by `stattab`.

### Syntax

```
DBMS_STATS.EXPORT_INDEX_STATS (
  ownname  VARCHAR2,
  indname  VARCHAR2,
  partname VARCHAR2 DEFAULT NULL,
  stattab  VARCHAR2,
  statid   VARCHAR2 DEFAULT NULL,
  statown  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–32** EXPORT\_INDEX\_STATS Procedure Parameters

Parameter	Description
<code>ownname</code>	Name of the schema
<code>indname</code>	Name of the index
<code>partname</code>	Name of the index partition. If the index is partitioned and if <code>partname</code> is <code>NULL</code> , then global and partition index statistics are exported.
<code>stattab</code>	User statistics table identifier describing where to store the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## EXPORT\_PENDING\_STATS Procedure

This procedure is used to export the statistics gathered and stored as pending.

### Syntax

```
DBMS_STATS.EXPORT_PENDING_STATS (
  ownname      IN VARCHAR2 DEFAULT USER,
  tablename    IN VARCHAR2,
  stattab      IN VARCHAR2,
  statid       IN VARCHAR2 DEFAULT NULL,
  statown      IN VARCHAR2 DEFAULT USER);
```

### Parameters

**Table 141–33 EXPORT\_PENDING\_STATS Procedure Parameters**

Parameter	Description
ownname	Owner name
tablename	Table name
stattab	Statistics table name to where to export the statistics
statid	(Optional) Identifier to associate with these statistics within <i>stattab</i>
statown	Schema containing <i>stattab</i> (if other than <i>ownname</i> )

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

- If the parameter *tablename* is `NULL` then export applies to all tables of the specified schema.
- The default owner/schema is the user who runs the procedure.
- To run this procedure, you need to have the same privilege for gathering statistics on the tables that will be touched by this procedure.
- All arguments are of type `VARCHAR2` and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.EXPORT_PENDING_STATS(NULL, NULL, 'MY_STAT_TABLE');
```

## EXPORT\_SCHEMA\_PREFS Procedure

This procedure is used to export the statistics preferences of all the tables owned by the specified owner name.

### Syntax

```
DBMS_STATS.EXPORT_SCHEMA_PREFS (
  ownname   IN  VARCHAR2,
  statab    IN  VARCHAR2,
  statid    IN  VARCHAR2 DEFAULT NULL,
  statown   IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141-34 EXPORT\_SCHEMA\_PREFS Procedure Parameters**

Parameter	Description
ownname	Owner name
statab	Statistics table name to where to export the statistics
statid	(Optional) Identifier to associate with these statistics within statab
statown	Schema containing statab (if different than ownname)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

- To run this procedure, you need to connect as owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.EXPORT_SCHEMA_PREFS('SH', 'STAT');
```

## EXPORT\_SCHEMA\_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by `ownname` and stores them in the user statistics tables identified by `stattab`.

### Syntax

```
DBMS_STATS.EXPORT_SCHEMA_STATS (
  ownname          VARCHAR2,
  stattab          VARCHAR2,
  statid          VARCHAR2 DEFAULT NULL,
  statown         VARCHAR2 DEFAULT NULL,
  stat_category   VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–35 EXPORT\_SCHEMA\_STATS Procedure Parameters**

Parameter	Description
<code>ownname</code>	Name of the schema
<code>stattab</code>	User statistics table identifier describing where to store the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
<code>stat_category</code>	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>▪ 'SYNOPSISSES' - auxiliary statistics information gathered on a partitioned table when <code>incremental</code> is <code>TRUE</code></li> </ul>

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## EXPORT\_SYSTEM\_STATS Procedure

This procedure retrieves system statistics and stores them in the user statistics table, identified by `stattab`.

### Syntax

```
DBMS_STATS.EXPORT_SYSTEM_STATS (
  stattab      VARCHAR2,
  statid       VARCHAR2 DEFAULT NULL,
  statown      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–36 EXPORT\_SYSTEM\_STATS Procedure Parameters**

Parameter	Description
<code>stattab</code>	Identifier of the user statistics table that describes where the statistics will be stored
<code>statid</code>	Optional identifier associated with the statistics stored from the <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table; may need to be upgraded

ORA-20003: Unable to export system statistics

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## EXPORT\_TABLE\_PREFS Procedure

This procedure is used to export the statistics preferences of the specified table in the specified schema into the specified statistics table.

### Syntax

```
DBMS_STATS.EXPORT_TABLE_PREFS (
  ownname      IN  VARCHAR2,
  tablename    IN  VARCHAR2,
  stattab      IN  VARCHAR2,
  statid       IN  VARCHAR2 DEFAULT NULL,
  statown      IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–37 EXPORT\_TABLE\_PREFS Procedure Parameters**

Parameter	Description
ownname	Owner name
tablename	Table name
stattab	Statistics table name where to export the statistics
statid	Optional identifier to associate with these statistics within stattab
statown	Schema containing stattab (if other than ownname)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

- To run this procedure, you need to connect as owner of the table, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.EXPORT_TABLE_PREFS('SH', 'SALES', 'STAT');
```



## EXPORT\_TABLE\_STATS Procedure

This procedure retrieves statistics for a particular table and stores them in the user statistics table. Cascade results in all index statistics associated with the specified table being exported as well.

### Syntax

```
DBMS_STATS.EXPORT_TABLE_STATS (
  ownname          VARCHAR2,
  tabname          VARCHAR2,
  partname         VARCHAR2 DEFAULT NULL,
  statab          VARCHAR2,
  statid          VARCHAR2 DEFAULT NULL,
  cascade         BOOLEAN DEFAULT TRUE,
  statown         VARCHAR2 DEFAULT NULL,
  stat_category   VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–38 EXPORT\_TABLE\_STATS Procedure Parameters**

Parameter	Description
ownname	Name of the schema
tabname	Name of the table
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are exported.
statab	User statistics table identifier describing where to store the statistics
statid	Identifier (optional) to associate with these statistics within statab
cascade	If true, then column and index statistics for this table are also exported
statown	Schema containing statab (if different than ownname)
stat_category	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>▪ 'SYNOPSISSES' - auxiliary statistics information gathered on a partitioned table when incremental is TRUE</li> </ul>

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## FLUSH\_DATABASE\_MONITORING\_INFO Procedure

This procedure flushes in-memory monitoring information for all tables in the dictionary. Corresponding entries in the \*\_TAB\_MODIFICATIONS, \*\_TAB\_STATISTICS and \*\_IND\_STATISTICS views are updated immediately, without waiting for the Oracle database to flush them periodically. This procedure is useful when you need up-to-date information in those views. Because the GATHER\_\*\_STATS procedures internally flush monitoring information, it is not necessary to run this procedure before gathering the statistics.

### Syntax

```
DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO;
```

### Exceptions

ORA-20000: Insufficient privileges

### Usage Notes

The ANALYZE\_ANY system privilege is required to run this procedure.

## GATHER\_DATABASE\_STATS Procedures

This procedure gathers statistics for all objects in the database.

### Syntax

```
DBMS_STATS.GATHER_DATABASE_STATS (
  estimate_percent NUMBER      DEFAULT to_estimate_percent_type
                                (get_param('ESTIMATE_PERCENT')),
  block_sample      BOOLEAN    DEFAULT FALSE,
  method_opt       VARCHAR2   DEFAULT get_param('METHOD_OPT'),
  degree           NUMBER     DEFAULT to_degree_type(get_param('DEGREE')),
  granularity      VARCHAR2   DEFAULT GET_PARAM('GRANULARITY'),
  cascade         BOOLEAN    DEFAULT to_cascade_type(get_param('CASCADE')),
  stattab         VARCHAR2   DEFAULT NULL,
  statid          VARCHAR2   DEFAULT NULL,
  options         VARCHAR2   DEFAULT 'GATHER',
  objlist         OUT        ObjectTab,
  statown        VARCHAR2   DEFAULT NULL,
  gather_sys      BOOLEAN    DEFAULT TRUE,
  no_invalidate   BOOLEAN    DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
  obj_filter_list ObjectTab  DEFAULT NULL);
```

```
DBMS_STATS.GATHER_DATABASE_STATS (
  estimate_percent NUMBER      DEFAULT to_estimate_percent_type
                                (get_param('ESTIMATE_PERCENT')),
  block_sample      BOOLEAN    DEFAULT FALSE,
  method_opt       VARCHAR2   DEFAULT get_param('METHOD_OPT'),
  degree           NUMBER     DEFAULT to_degree_type(get_param('DEGREE')),
  granularity      VARCHAR2   DEFAULT GET_PARAM('GRANULARITY'),
  cascade         BOOLEAN    DEFAULT to_cascade_type(get_param('CASCADE')),
  stattab         VARCHAR2   DEFAULT NULL,
  statid          VARCHAR2   DEFAULT NULL,
  options         VARCHAR2   DEFAULT 'GATHER',
  statown        VARCHAR2   DEFAULT NULL,
  gather_sys      BOOLEAN    DEFAULT TRUE,
  no_invalidate   BOOLEAN    DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
  obj_filter_list ObjectTab  DEFAULT NULL);
```

### Parameters

**Table 141–39 GATHER\_DATABASE\_STATS Procedure Parameters**

Parameter	Description
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.

**Table 141–39 (Cont.) GATHER\_DATABASE\_STATS Procedure Parameters**

Parameter	Description
method_opt	<p>Accepts:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer: Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT: Collects histograms only on the columns that already have histograms.</li> <li>- AUTO: Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY: Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
degree	<p>Degree of parallelism. The default for degree is NULL. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</p>
granularity	<p>Granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics</p>

**Table 141–39 (Cont.) GATHER\_DATABASE\_STATS Procedure Parameters**

Parameter	Description
<code>cascade</code>	Gather statistics on the indexes as well. Using this option is equivalent to running the <a href="#">GATHER_INDEX_STATS Procedure</a> on each of the indexes in the database in addition to gathering table and column statistics. Use the constant <code>DBMS_STATS.AUTO_CASCADE</code> to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
<code>stattab</code>	User statistics table identifier describing where to save the current statistics.  The statistics table is assumed to reside in the same schema as the object being analyzed, so there must be one such table in each schema to use this option.
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code> .
<code>options</code>	Further specification of which objects to gather statistics for:  <code>GATHER</code> : Gathers statistics on all objects in the schema.  <code>GATHER AUTO</code> : Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When <code>GATHER AUTO</code> is specified, the only additional valid parameters are <code>stattab</code> , <code>statid</code> , <code>objlist</code> and <code>statown</code> ; all other parameter settings are ignored. Returns a list of processed objects.  <code>GATHER STALE</code> : Gathers statistics on stale objects as determined by looking at the <code>*_tab_modifications</code> views. Also, return a list of objects found to be stale.  <code>GATHER EMPTY</code> : Gathers statistics on objects which currently have no statistics. Return a list of objects found to have no statistics.  <code>LIST AUTO</code> : Returns a list of objects to be processed with <code>GATHER AUTO</code>  <code>LIST STALE</code> : Returns a list of stale objects as determined by looking at the <code>*_tab_modifications</code> views  <code>LIST EMPTY</code> : Returns a list of objects which currently have no statistics
<code>objlist</code>	List of objects found to be stale or empty
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)
<code>gather_sys</code>	Gathers statistics on the objects owned by the 'SYS' user
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> . to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

**Table 141–39 (Cont.) GATHER\_DATABASE\_STATS Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>obj_filter_list</code>	A list of object filters. When provided, <code>GATHER_DATABASE_STATS</code> will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values <code>s1</code> , <code>s2</code> , ... are specified for attributes <code>a1</code> , <code>a2</code> , ... in one object filter. An object <code>o</code> is said to satisfy this object filter if <code>(o.a1 like s1)</code> and <code>(o.a2 like s2)</code> and ... is true. See <a href="#">Applying an Object Filter List</a> .

## Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Bad input value

## GATHER\_DICTIONARY\_STATS Procedure

This procedure gathers statistics for dictionary schemas 'SYS', 'SYSTEM' and schemas of RDBMS components.

### Syntax

```
DBMS_STATS.GATHER_DICTIONARY_STATS (
  comp_id          VARCHAR2 DEFAULT NULL,
  estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                          (get_param('ESTIMATE_PERCENT')),
  block_sample     BOOLEAN   DEFAULT FALSE,
  method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
  degree           NUMBER    DEFAULT to_degree_type(get_param('DEGREE')),
  granularity       VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
  cascade          BOOLEAN   DEFAULT to_cascade_type(get_param('CASCADE')),
  stattab          VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  options          VARCHAR2 DEFAULT 'GATHER AUTO',
  objlist          OUT      ObjectTab,
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN   DEFAULT to_no_invalidate_type (
                          get_param('NO_INVALIDATE')),
  obj_filter_list  ObjectTab DEFAULT NULL);

DBMS_STATS.GATHER_DICTIONARY_STATS (
  comp_id          VARCHAR2 DEFAULT NULL,
  estimate_percent NUMBER   DEFAULT
                          to_estimate_percent_type(GET_PARAM('ESTIMATE_PERCENT')),
  block_sample     BOOLEAN   DEFAULT FALSE,
  method_opt       VARCHAR2 DEFAULT GET_PARAM('METHOD_OPT'),
  degree           NUMBER    DEFAULT to_degree_type(GET_PARAM('DEGREE')),
  granularity       VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
  cascade          BOOLEAN   DEFAULT to_cascade_type(GET_PARAM('CASCADE')),
  stattab          VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  options          VARCHAR2 DEFAULT 'GATHER AUTO',
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN   DEFAULT
                          to_no_invalidate_type(get_param('NO_INVALIDATE')),
  obj_filter_list  ObjectTab DEFAULT NULL);
```

### Parameters

**Table 141–40 GATHER\_DICTIONARY\_STATS Procedure Parameters**

Parameter	Description
comp_id	Component id of the schema to analyze (NULL will result in analyzing schemas of all RDBMS components). Please refer to comp_id column of DBA_REGISTRY view. The procedure always gather statistics on 'SYS' and 'SYSTEM' schemas regardless of this argument.

**Table 141–40 (Cont.) GATHER\_DICTIONARY\_STATS Procedure Parameters**

Parameter	Description
estimate_percent	Percentage of rows to estimate (NULL means compute). The valid range is [0.000001, 100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
block_sample	Determines whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk then the sample values may be somewhat correlated. Only pertinent when performing estimate statistics.
method_opt	<p>Accepts:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer: Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT: Collects histograms only on the columns that already have histograms.</li> <li>- AUTO: Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY: Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
degree	Degree of parallelism. The default for degree is NULL. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> . NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=>NULL, DEGREE=>n, or DEGREE=>DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.



**Table 141–40 (Cont.) GATHER\_DICTIONARY\_STATS Procedure Parameters**

Parameter	Description
granularity	<p>Granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - gathers subpartition-level statistics</p>
cascade	<p>Gathers statistics on indexes also. Index statistics gathering will not be parallelized. Using this option is equivalent to running the <a href="#">GATHER_INDEX_STATS Procedure</a> on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant <code>DBMS_STATS.AUTO_CASCADE</code> to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
stattab	<p>User statistics table identifier describing where to save the current statistics</p>
statid	<p>The (optional) identifier to associate with these statistics within <code>stattab</code></p>
options	<p>Further specification of objects for which to gather statistics:</p> <ul style="list-style-type: none"> <li>■ 'GATHER' - Gathers statistics on all objects in the schema</li> <li>■ 'GATHER AUTO' - Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics and determines how to gather those statistics. When 'GATHER AUTO' is specified, the only additional valid parameters are <code>comp_id</code>, <code>stattab</code>, <code>statid</code> and <code>statown</code>; all other parameter settings will be ignored. Also, returns a list of objects processed.</li> <li>■ 'GATHER STALE' - Gathers statistics on stale objects as determined by looking at the <code>*_tab_modifications</code> views. Also, returns a list of objects found to be stale.</li> <li>■ 'GATHER EMPTY' - Gathers statistics on objects which currently have no statistics. Also, returns a list of objects found to have no statistics.</li> <li>■ 'LIST AUTO' - Returns list of objects to be processed with 'GATHER AUTO'</li> <li>■ 'LIST STALE' - Returns list of stale objects as determined by looking at the <code>*_tab_modifications</code> views</li> <li>■ 'LIST EMPTY' - Returns list of objects which currently have no statistics</li> </ul>

**Table 141–40 (Cont.) GATHER\_DICTIONARY\_STATS Procedure Parameters**

Parameter	Description
objlist	The list of objects found to be stale or empty
statown	Schema containingstattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
obj_filter_list	A list of object filters. When provided, this will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case- insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values s1, s2, ... are specified for attributes a1, a2, ... in one object filter. An object o is said to satisfy this object filter if (o.a1 like s1) and (o.a2 like s2) and ... is true. See <a href="#">Applying an Object Filter List</a> .

## Usage Notes

You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.

## Exceptions

ORA-20000: Index does not exist or insufficient privileges

ORA-20001: Bad input value

ORA-20002: Bad user statistics table, may need to upgrade it

## GATHER\_FIXED\_OBJECTS\_STATS Procedure

This procedure gathers statistics for all fixed objects (dynamic performance tables).

### Syntax

```
DBMS_STATS.GATHER_FIXED_OBJECTS_STATS (
  statab      VARCHAR2 DEFAULT NULL,
  statid      VARCHAR2 DEFAULT NULL,
  statown     VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')));
```

### Parameters

**Table 141-41** GATHER\_FIXED\_OBJECTS\_STATS Procedure Parameters

Parameter	Description
statab	User statistics table identifier describing where to save the current statistics
statid	Identifier to associate with these statistics within statab (optional)
statown	Schema containing statab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

### Usage Notes

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.

### Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Bad input value

ORA-20002: Bad user statistics table, may need to upgrade it

## GATHER\_INDEX\_STATS Procedure

This procedure gathers index statistics. It attempts to parallelize as much of the work as possible. Restrictions are described in the individual parameters. This operation will not parallelize with certain types of indexes, including cluster indexes, domain indexes, and bitmap join indexes. The `granularity` and `no_invalidate` arguments are not relevant to these types of indexes.

### Syntax

```
DBMS_STATS.GATHER_INDEX_STATS (
  ownname          VARCHAR2,
  indname          VARCHAR2,
  partname         VARCHAR2 DEFAULT NULL,
  estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                     (GET_PARAM('ESTIMATE_PERCENT')),
  statab           VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
  granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
  no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type
                                     (GET_PARAM('NO_INVALIDATE')),
  force            BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–42 GATHER\_INDEX\_STATS Procedure Parameters**

Parameter	Description
<code>ownname</code>	Schema of index to analyze
<code>indname</code>	Name of index
<code>partname</code>	Name of partition
<code>estimate_percent</code>	Percentage of rows to estimate (NULL means compute). The valid range is [0.000001, 100]. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
<code>statab</code>	User statistics table identifier describing where to save the current statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>statab</code>
<code>statown</code>	Schema containing <code>statab</code> (if different than <code>ownname</code> )

**Table 141-42 (Cont.) GATHER\_INDEX\_STATS Procedure Parameters**

Parameter	Description
degree	Degree of parallelism. The default for degree is NULL. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> . NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=>NULL, DEGREE=>n, or DEGREE=>DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). 'ALL' - Gathers all (subpartition, partition, and global) statistics 'AUTO' - Determines the granularity based on the partitioning type. This is the default value. 'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'. 'GLOBAL' - Gathers global statistics 'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object. 'PARTITION' - Gathers partition-level statistics 'SUBPARTITION' - Gathers subpartition-level statistics.
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Gather statistics on object even if it is locked

## Exceptions

ORA-20000: Index does not exist or insufficient privileges

ORA-20001: Bad input value

## GATHER\_SCHEMA\_STATS Procedures

This procedure gathers statistics for all objects in a schema.

### Syntax

```

DBMS_STATS.GATHER_SCHEMA_STATS (
  ownname          VARCHAR2,
  estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                        (get_param('ESTIMATE_PERCENT')),
  block_sample     BOOLEAN  DEFAULT FALSE,
  method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
  degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
  granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
  cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
  stattab          VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  options          VARCHAR2 DEFAULT 'GATHER',
  objlist          OUT      ObjectTab,
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                        get_param('NO_INVALIDATE')),
  force            BOOLEAN  DEFAULT FALSE,
  obj_filter_list  ObjectTab DEFAULT NULL);

DBMS_STATS.GATHER_SCHEMA_STATS (
  ownname          VARCHAR2,
  estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                        (get_param('ESTIMATE_PERCENT')),
  block_sample     BOOLEAN  DEFAULT FALSE,
  method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
  degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
  granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
  cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
  stattab          VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  options          VARCHAR2 DEFAULT 'GATHER',
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                        get_param('NO_INVALIDATE')),
  force            BOOLEAN  DEFAULT FALSE,
  obj_filter_list  ObjectTab DEFAULT NULL);

```

### Parameters

**Table 141–43 GATHER\_SCHEMA\_STATS Procedure Parameters**

Parameter	Description
ownname	Schema to analyze (NULL means current schema)
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100]. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

**Table 141-43 (Cont.) GATHER\_SCHEMA\_STATS Procedure Parameters**

Parameter	Description
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	<p>Accepts:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer: Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT: Collects histograms only on the columns that already have histograms</li> <li>- AUTO: Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY: Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
degree	<p>Degree of parallelism. The default for degree is NULL. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</p>

**Table 141–43 (Cont.) GATHER\_SCHEMA\_STATS Procedure Parameters**

Parameter	Description
granularity	<p>Granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics.</p>
cascade	<p>Gather statistics on the indexes as well. Using this option is equivalent to running the <a href="#">GATHER_INDEX_STATS Procedure</a> on each of the indexes in the schema in addition to gathering table and column statistics. Use the constant <code>DEMS_STATS.AUTO_CASCADE</code> to have Oracle determine whether index statistics to be collected or not. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
stattab	<p>User statistics table identifier describing where to save the current statistics</p>
statid	<p>Identifier (optional) to associate with these statistics within stattab</p>
options	<p>Further specification of which objects to gather statistics for:</p> <p><b>GATHER:</b> Gathers statistics on all objects in the schema.</p> <p><b>GATHER AUTO:</b> Gathers all necessary statistics automatically. Oracle implicitly determines which objects need new statistics, and determines how to gather those statistics. When <b>GATHER AUTO</b> is specified, the only additional valid parameters are <code>ownname</code>, <code>stattab</code>, <code>statid</code>, <code>objlist</code> and <code>statown</code>; all other parameter settings are ignored. Returns a list of processed objects.</p> <p><b>GATHER STALE:</b> Gathers statistics on stale objects as determined by looking at the <code>*_tab_modifications</code> views. Also, return a list of objects found to be stale.</p> <p><b>GATHER EMPTY:</b> Gathers statistics on objects which currently have no statistics. also, return a list of objects found to have no statistics.</p> <p><b>LIST AUTO:</b> Returns a list of objects to be processed with <b>GATHER AUTO</b>.</p> <p><b>LIST STALE:</b> Returns list of stale objects as determined by looking at the <code>*_tab_modifications</code> views.</p> <p><b>LIST EMPTY:</b> Returns list of objects which currently have no statistics.</p>



**Table 141–43 (Cont.) GATHER\_SCHEMA\_STATS Procedure Parameters**

Parameter	Description
objlist	List of objects found to be stale or empty
statown	Schema containingstattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Gather statistics on objects even if they are locked
obj_filter_list	A list of object filters. When provided, GATHER_SCHEMA_STATS will gather statistics only on objects which satisfy at least one object filter in the list as needed. In a single object filter, we can specify the constraints on the object attributes. The attribute values specified in the object filter are case-insensitive unless double-quoted. Wildcard is allowed in the attribute values. Suppose non-NULL values s1, s2, ... are specified for attributes a1, a2, ... in one object filter. An object o is said to satisfy this object filter if (o.a1 like s1) and (o.a2 like s2) and ... is true. See <a href="#">Applying an Object Filter List</a> .

## Usage Notes

When you use a specific value for the sampling percentage, DBMS\_STATS honors it except for when:

- The result is less than 2500 rows (too small a sample) and
- The specified percentage is more than the certain percentage.

## Exceptions

ORA-20000: Schema does not exist or insufficient privileges

ORA-20001: Bad input value

## Examples

### Applying an Object Filter List

The following example specifies that any table with a "T" prefix in the SAMPLE schema and any table in the SYS schema, if stale, will have statistics gathered upon it.

```

DECLARE
    filter_lst DBMS_STATS.OBJECTTAB := DBMS_STATS.OBJECTTAB();
BEGIN
    filter_lst.extend(2);
    filter_lst(1).ownname := 'SAMPLE';
    filter_lst(1).objname := 'T%';
    filter_lst(2).ownname := 'SYS';
    DBMS_STATS.GATHER_SCHEMA_STATS(NULL, obj_filter_list => filter_lst,
                                   options => 'GATHER STALE');
END;
```

## GATHER\_SYSTEM\_STATS Procedure

This procedure gathers system statistics.

### Syntax

```
DBMS_STATS.GATHER_SYSTEM_STATS (
  gathering_mode  VARCHAR2 DEFAULT 'NOWORKLOAD' ,
  interval        INTEGER   DEFAULT NULL,
  stattab         VARCHAR2  DEFAULT NULL,
  statid          VARCHAR2  DEFAULT NULL,
  statown         VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 141–44 GATHER\_SYSTEM\_STATS Procedure Parameters**

Parameter	Description
<code>gathering_mode</code>	<p>Mode values are:</p> <p><b>NOWORKLOAD:</b> Will capture characteristics of the I/O system. Gathering may take a few minutes and depends on the size of the database. During this period Oracle will estimate the average read seek time and transfer speed for the I/O system. This mode is suitable for the all workloads. Oracle recommends to run <code>GATHER_SYSTEM_STATS ('noworkload')</code> after creation of the database and tablespaces. To fine tune system statistics for the workload use 'START' and 'STOP' or 'INTERVAL' options. If you gather both 'NOWORKLOAD' and workload specific (statistics collected using 'INTERVAL' or 'START' and 'STOP'), the workload statistics will be used by optimizer. Collected components: <code>cpuspeednw</code>, <code>ioseektim</code>, <code>iotfrspeed</code>.</p> <p><b>INTERVAL:</b> Captures system activity during a specified interval. This works in combination with the <code>interval</code> parameter. You should provide an interval value in minutes, after which system statistics are created or updated in the dictionary or <code>stattab</code>. You can use <code>GATHER_SYSTEM_STATS (gathering_mode=&gt; 'STOP')</code> to stop gathering earlier than scheduled. Collected components: <code>maxthr</code>, <code>slavethr</code>, <code>cpuspeed</code>, <code>sreadtim</code>, <code>mreadtim</code>, <code>mbrc</code>.</p> <p><b>START   STOP:</b> Captures system activity during specified start and stop times and refreshes the dictionary or <code>stattab</code> with statistics for the elapsed period. Interval value is ignored. Collected components: <code>maxthr</code>, <code>slavethr</code>, <code>cpuspeed</code>, <code>sreadtim</code>, <code>mreadtim</code>, <code>mbrc</code>.</p>
<code>interval</code>	Time, in minutes, to gather statistics. This parameter applies only when <code>gathering_mode= 'INTERVAL'</code>
<code>stattab</code>	Identifier of the user statistics table where the statistics will be saved
<code>statid</code>	Optional identifier associated with the statistics saved in the <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)

### Exceptions

- ORA-20000: Object does not exist or insufficient privileges
- ORA-20001: Invalid input value

ORA-20002: Bad user statistics table; may need to be upgraded

ORA-20003: Unable to gather system statistics

ORA-20004: Error in the INTERVAL mode: system parameter job\_queue\_processes must be >0

## GATHER\_TABLE\_STATS Procedure

This procedure gathers table and column (and index) statistics. It attempts to parallelize as much of the work as possible, but there are some restrictions as described in the individual parameters.

### Syntax

```
DBMS_STATS.GATHER_TABLE_STATS (
  ownname          VARCHAR2,
  tabname          VARCHAR2,
  partname         VARCHAR2 DEFAULT NULL,
  estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                     (get_param('ESTIMATE_PERCENT')),
  block_sample     BOOLEAN  DEFAULT FALSE,
  method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
  degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
  granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
  cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
 stattab          VARCHAR2 DEFAULT NULL,
  statid           VARCHAR2 DEFAULT NULL,
  statown         VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type (
                                     get_param('NO_INVALIDATE')),
  stattype         VARCHAR2 DEFAULT 'DATA',
  force           BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–45 GATHER\_TABLE\_STATS Procedure Parameters**

Parameter	Description
ownname	Schema of table to analyze
tabname	Name of table
partname	Name of partition
estimate_percent	Percentage of rows to estimate (NULL means compute) The valid range is [0.000001,100]. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to have Oracle determine the appropriate sample size for good statistics. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.

**Table 141–45 (Cont.) GATHER\_TABLE\_STATS Procedure Parameters**

Parameter	Description
method_opt	<p>Accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> <li>- column_name : Name of a column</li> <li>- extension : can be either a column group in the format of (column_name, Column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
degree	<p>Degree of parallelism. The default for degree is NULL. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>. NULL means use the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement. Use the constant DBMS_STATS.DEFAULT_DEGREE to specify the default value based on the initialization parameters. The AUTO_DEGREE value determines the degree of parallelism automatically. This is between 1 (serial execution) and DEFAULT_DEGREE (the system default value based on number of CPUs and initialization parameters) according to the size of the object. When using DEGREE=&gt;NULL, DEGREE=&gt;n, or DEGREE=&gt;DBMS_STATS.DEFAULT_DEGREE, the current implementation of DBMS_STATS may use serial execution if the size of the object does not warrant parallel execution.</p>

**Table 141–45 (Cont.) GATHER\_TABLE\_STATS Procedure Parameters**

Parameter	Description
granularity	<p>Granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'APPROX_GLOBAL AND PARTITION' - similar to 'GLOBAL AND PARTITION' but in this case the global statistics are aggregated from partition level statistics. This option will aggregate all statistics except the number of distinct values for columns and number of distinct keys of indexes. The existing histograms of the columns at the table level are also aggregated. The aggregation will use only partitions with statistics, so to get accurate global statistics, users should make sure to have statistics for all partitions. Global statistics are gathered if <code>partname</code> is NULL or if the aggregation cannot be performed (for example, if statistics for one of the partitions is missing).</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics.</p>
cascade	<p>Gathers statistics on the indexes for this table. Using this option is equivalent to running the <a href="#">GATHER_INDEX_STATS Procedure</a> on each of the table's indexes. Use the constant <code>DBMS_STATS.AUTO_CASCADE</code> to have Oracle determine whether index statistics are to be collected or not. This is the default. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
stattab	<p>User statistics table identifier describing where to save the current statistics</p>
statid	<p>Identifier (optional) to associate with these statistics within <code>stattab</code></p>
statown	<p>Schema containing <code>stattab</code> (if different than <code>ownname</code>)</p>
no_invalidate	<p>Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use <code>DBMS_STATS.AUTO_INVALIDATE</code>. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
stattype	<p>Statistics type. The only value allowed is DATA.</p>
force	<p>Gather statistics of table even if it is locked</p>

## Usage Notes

Index statistics collection can be parallelized except for cluster, domain and join indexes.

## Exceptions

ORA-20000: Table does not exist or insufficient privileges

ORA-20001: Bad input value

## Examples

An extension can be either a column group (see Example 1) or an expression (see Example 2).

### Example 1

```
DBMS_STATS.GATHER_TABLE_STATS(  
    'SH', 'SALES', method_opt => 'FOR COLUMNS (empno, deptno)');
```

### Example 2

```
DBMS_STATS.GATHER_TABLE_STATS(  
    'SH', 'SALES', method_opt => 'FOR COLUMNS (sal+comm)');
```

## GENERATE\_STATS Procedure

This procedure generates object statistics from previously collected statistics of related objects. The currently supported objects are b-tree and bitmap indexes.

### Syntax

```
DBMS_STATS.GENERATE_STATS (
  ownname    VARCHAR2,
  objname    VARCHAR2,
  organized  NUMBER DEFAULT 7,
  force      BOOLEAN default FALSE);
```

### Parameters

**Table 141–46** *GENERATE\_STATS Procedure Parameters*

Parameter	Description
ownname	Schema of object
objname	Name of object
organized	Amount of ordering associated between the index and its underlying table. A heavily organized index would have consecutive index keys referring to consecutive rows on disk for the table (the same block). A heavily disorganized index would have consecutive keys referencing different table blocks on disk.  This parameter is only used for b-tree indexes. The number can be in the range of 0-10, with 0 representing a completely organized index and 10 a completely disorganized one.
force	If TRUE, generates statistics for the target object even if it is locked

### Usage Notes

For fully populated schemas, the gather procedures should be used instead when more accurate statistics are desired.

### Exceptions

ORA-20000: Unsupported object type of object does not exist

ORA-20001: Invalid option or invalid statistics



## GET\_COLUMN\_STATS Procedures

These procedures gets all column-related information. In the form of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

### Syntax

```
DBMS_STATS.GET_COLUMN_STATS (
  ownname      VARCHAR2,
  tabname      VARCHAR2,
  colname      VARCHAR2,
  partname     VARCHAR2 DEFAULT NULL,
  statab       VARCHAR2 DEFAULT NULL,
  statid       VARCHAR2 DEFAULT NULL,
  distcnt     OUT NUMBER,
  density     OUT NUMBER,
  nullcnt     OUT NUMBER,
  srec        OUT StatRec,
  avgclen     OUT NUMBER,
  statown     VARCHAR2 DEFAULT NULL);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_COLUMN_STATS (
  ownname      VARCHAR2,
  tabname      VARCHAR2,
  colname      VARCHAR2,
  partname     VARCHAR2 DEFAULT NULL,
  statab       VARCHAR2 DEFAULT NULL,
  statid       VARCHAR2 DEFAULT NULL,
  ext_stats    OUT RAW,
  statypown   OUT VARCHAR2 DEFAULT NULL,
  statypname   OUT VARCHAR2 DEFAULT NULL,
  statown     VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–47** GET\_COLUMN\_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
colname	Name of the column or extension
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, statistics are retrieved from the global table level.
statab	User statistics table identifier describing from where to retrieve the statistics. If statab is NULL, statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within statab (Only pertinent if statab is not NULL)
ext_stats	The user-defined statistics
statypown	Schema of the statistics type

**Table 141–47 (Cont.) GET\_COLUMN\_STATS Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
stattypname	Name of the statistics type
distcnt	Number of distinct values
density	Column density
nullcnt	Number of NULLs
srec	Structure holding internal representation of column minimum, maximum, and histogram values
avgclen	Average length of the column (in bytes)
statown	Schema containing stattab (if different than ownname)

## Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

## GET\_INDEX\_STATS Procedures

These procedures get all index-related information. In the form of this procedure that deals with user-defined statistics, the statistics type returned is the type stored, in addition to the user-defined statistics.

### Syntax

```
DBMS_STATS.GET_INDEX_STATS (
    ownname          VARCHAR2,
    indname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    statab           VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    numrows          OUT NUMBER,
    numlblks         OUT NUMBER,
    numdist          OUT NUMBER,
    avglblk          OUT NUMBER,
    avgdblks         OUT NUMBER,
    clstfct          OUT NUMBER,
    indlevel         OUT NUMBER,
    statown          VARCHAR2 DEFAULT NULL,
    cachedblk       OUT NUMBER,
    cachehit         OUT NUMBER);
```

```
DBMS_STATS.GET_INDEX_STATS (
    ownname          VARCHAR2,
    indname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    statab           VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    numrows          OUT NUMBER,
    numlblks         OUT NUMBER,
    numdist          OUT NUMBER,
    avglblk          OUT NUMBER,
    avgdblks         OUT NUMBER,
    clstfct          OUT NUMBER,
    indlevel         OUT NUMBER,
    statown          VARCHAR2 DEFAULT NULL,
    guessq           OUT NUMBER,
    cachedblk       OUT NUMBER,
    cachehit         OUT NUMBER);
```

Use the following for user-defined statistics:

```
DBMS_STATS.GET_INDEX_STATS (
    ownname          VARCHAR2,
    indname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    statab           VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    ext_stats        OUT RAW,
    statypown        OUT VARCHAR2 DEFAULT NULL,
    statypname       OUT VARCHAR2 DEFAULT NULL,
    statown          VARCHAR2 DEFAULT NULL,
    cachedblk       OUT NUMBER,
    cachehit         OUT NUMBER);
```

## Parameters

**Table 141–48** *GET\_INDEX\_STATS Procedure Parameters*

Parameter	Description
<code>ownname</code>	Name of the schema
<code>indname</code>	Name of the index
<code>partname</code>	Name of the index partition for which to get the statistics. If the index is partitioned and if <code>partname</code> is <code>NULL</code> , then the statistics are retrieved for the global index level.
<code>stattab</code>	User statistics table identifier describing from where to retrieve the statistics. If <code>stattab</code> is <code>NULL</code> , then the statistics are retrieved directly from the dictionary.
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code> (Only pertinent if <code>stattab</code> is not <code>NULL</code> )
<code>ext_stats</code>	User-defined statistics
<code>stattypown</code>	Schema of the statistics type
<code>stattypname</code>	Name of the statistics type
<code>numrows</code>	Number of rows in the index (partition)
<code>numlblks</code>	Number of leaf blocks in the index (partition)
<code>numdist</code>	Number of distinct keys in the index (partition)
<code>avglblk</code>	Average integral number of leaf blocks in which each distinct key appears for this index (partition)
<code>avgdblks</code>	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition)
<code>clstfct</code>	Clustering factor for the index (partition)
<code>indlevel</code>	Height of the index (partition)
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
<code>guessq</code>	Guess quality for the index (partition)
<code>cachedblk</code>	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
<code>cachehit</code>	The average cache hit ratio for the segment (index/table/index partition/table partition)

## Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains `cachedblk` and `cachehit` at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls `DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS` procedure for auto mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.

- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

## Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

## GET\_PARAM Function

---

---

**Note:** This subprogram has been replaced by improved technology and is maintained only for purposes of backward compatibility. In this case, use the [GET\\_PREFS Function](#).

See also [Deprecated Subprograms](#) on page 141-12.

---

---

This function returns the default value of parameters of DBMS\_STATS procedures.

### Syntax

```
DBMS_STATS.GET_PARAM (  
    pname      IN  VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 141–49** *GET\_PARAM Function Parameters*

Parameter	Description
pname	Parameter name

### Exceptions

ORA-20001: Invalid input values

## GET\_PREFS Function

This function returns the default value of the specified preference.

### Syntax

```
DBMS_STATS.GET_PREFS (
    pname      IN   VARCHAR2,
    ownname    IN   VARCHAR2 DEFAULT NULL,
    tabname    IN   VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 141–50** GET\_PREFS Function Parameters

Parameter	Description
pname	<p>Preference name. The default value for following preferences can be retrieved:</p> <ul style="list-style-type: none"> <li>▪ AUTOSTATS_TARGET</li> <li>▪ CASCADE</li> <li>▪ DEGREE</li> <li>▪ ESTIMATE_PERCENT</li> <li>▪ METHOD_OPT</li> <li>▪ NO_INVALIDATE</li> <li>▪ GRANULARITY</li> <li>▪ PUBLISH</li> <li>▪ INCREMENTAL</li> <li>▪ STALE_PERCENT</li> </ul> <p>AUTOSTATS_TARGET - This preference is applicable only for auto statistics collection. The value of this parameter controls the objects considered for statistics collection. It takes the following values:</p> <ul style="list-style-type: none"> <li>▪ 'ALL' - Statistics collected for all objects in system</li> <li>▪ 'ORACLE' - Statistics collected for all Oracle owned objects</li> <li>▪ 'AUTO' - Oracle decides on which objects to collect statistics</li> </ul> <p>CASCADE - Determines whether or not index statistics are collected as part of gathering table statistics.</p> <p>DEGREE - Determines degree of parallelism used for gathering statistics.</p> <p>ESTIMATE_PERCENT - Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.</p>

**Table 141–50 (Cont.) GET\_PREFS Function Parameters**

Parameter	Description
.	<p>METHOD_OPT - Controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms.</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> <li>- column_name : name of a column</li> <li>- extension : can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO. The default value can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a>, <a href="#">SET_GLOBAL_PREFS Procedure</a>, <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>.</p>
.	<p>NO_INVALIDATE - The value controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default.</p>
.	<p>GRANULARITY - Determines granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics.</p>



**Table 141–50 (Cont.) GET\_PREFS Function Parameters**

Parameter	Description
.	<p>PUBLISH - Determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
.	<p>INCREMENTAL - Determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ INCREMENTAL value for the partitioned table is set to TRUE</li> <li>■ PUBLISH value for the partitioned table is set to TRUE;</li> <li>■ User specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
.	<p>STALE_PERCENT - This value determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%.</p>
ownname	Owner name
tabname	Table name

## Exceptions

ORA-20001: Invalid input values

## Usage Notes

If the ownname and tabname are provided and a preference has been entered for the table, the function returns the preference as specified for the table. In all other cases it returns the global preference if it has been specified, otherwise the default value is returned.

## GET\_STATS\_HISTORY\_AVAILABILITY Function

This function returns oldest timestamp where statistics history is available. Users cannot restore statistics to a timestamp older than this one.

### Syntax

```
DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY  
RETURN TIMESTAMP WITH TIMEZONE;
```

## GET\_STATS\_HISTORY\_RETENTION Function

This function returns the current retention value.

### Syntax

```
DBMS_STATS.GET_STATS_HISTORY_RETENTION  
RETURN NUMBER;
```

## GET\_SYSTEM\_STATS Procedure

This procedure gets system statistics from `stattab`, or from the dictionary if `stattab` is `NULL`.

### Syntax

```
DBMS_STATS.GET_SYSTEM_STATS (
  status      OUT  VARCHAR2,
  dstart      OUT  DATE,
  dstop       OUT  DATE,
  pname       VARCHAR2,
  pvalue      OUT  NUMBER,
  stattab     IN   VARCHAR2 DEFAULT NULL,
  statid      IN   VARCHAR2 DEFAULT NULL,
  statown     IN   VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–51** GET\_SYSTEM\_STATS Procedure Parameters

Parameter	Description
<code>status</code>	Output is one of the following: <ul style="list-style-type: none"> <li>▪ <code>COMPLETED</code> :</li> <li>▪ <code>AUTOGATHERING</code> :</li> <li>▪ <code>MANUALGATHERING</code> :</li> <li>▪ <code>BADSTATS</code> :</li> </ul>
<code>dstart</code>	Date when statistics gathering started. If <code>status = MANUALGATHERING</code> , the start date is returned.
<code>dstop</code>	Date when statistics gathering stopped. <ul style="list-style-type: none"> <li>▪ If <code>status = COMPLETE</code>, the finish date is returned.</li> <li>▪ If <code>status = AUTOGATHERING</code>, the future finish date is returned.</li> <li>▪ If <code>status = BADSTATS</code>, the must-finished-by date is returned.</li> </ul>

**Table 141-51 (Cont.) GET\_SYSTEM\_STATS Procedure Parameters**

Parameter	Description
<code>pname</code>	The parameter name to get, which can have one of the following values: <ul style="list-style-type: none"> <li>▪ <code>iotfrspeed</code> - I/O transfer speed in bytes for each millisecond</li> <li>▪ <code>ioseektim</code> - seek time + latency time + operating system overhead time, in milliseconds</li> <li>▪ <code>sreadtim</code> - average time to read single block (random read), in milliseconds</li> <li>▪ <code>mreadtim</code> - average time to read an mbrc block at once (sequential read), in milliseconds</li> <li>▪ <code>cpuspeed</code> - average number of CPU cycles for each second, in millions, captured for the workload (statistics collected using 'INTERVAL' or 'START' and 'STOP' options)</li> <li>▪ <code>cpuspeednw</code> - average number of CPU cycles for each second, in millions, captured for the noworkload (statistics collected using 'NOWORKLOAD' option.</li> <li>▪ <code>mbrc</code> - average multiblock read count for sequential read, in blocks</li> <li>▪ <code>maxthr</code> - maximum I/O system throughput, in bytes/second</li> <li>▪ <code>slavethr</code> - average slave I/O throughput, in bytes/second</li> </ul>
<code>pvalue</code>	Parameter value to get
<code>stattab</code>	Identifier of the user statistics table where the statistics will be obtained. If <code>stattab</code> is null, the statistics will be obtained from the dictionary.
<code>statid</code>	Optional identifier associated with the statistics saved in the <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20002: Bad user statistics table; may need to be upgraded

ORA-20003: Unable to gather system statistics

ORA-20004: Parameter does not exist

## GET\_TABLE\_STATS Procedure

This procedure gets all table-related information.

### Syntax

```
DBMS_STATS.GET_TABLE_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    stattab         VARCHAR2 DEFAULT NULL,
    statid          VARCHAR2 DEFAULT NULL,
    numRows         OUT NUMBER,
    numblks        OUT NUMBER,
    avgrlen        OUT NUMBER,
    statown         VARCHAR2 DEFAULT NULL,
    cachedblk      OUT NUMBER,
    cachehit       OUT NUMBER);
```

### Parameters

**Table 141–52 GET\_TABLE\_STATS Procedure Parameters**

Parameter	Description
ownname	Name of the schema
tabname	Name of the table to which this column belongs
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if <code>partname</code> is <code>NULL</code> , then the statistics are retrieved from the global table level.
stattab	User statistics table identifier describing from where to retrieve the statistics. If <code>stattab</code> is <code>NULL</code> , then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within <code>stattab</code> (Only pertinent if <code>stattab</code> is not <code>NULL</code> )
numrows	Number of rows in the table (partition)
numblks	Number of blocks the table (partition) occupies
avgrlen	Average row length for the table (partition)
statown	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
cachehit	The average cache hit ratio for the segment (index/table/index partition/table partition)

### Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains `cachedblk` and `cachehit` at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls `DBMS_STATS.GATHER_`

[TABLE/INDEX/SCHEMA/DATABASE]\_STATS procedure for auto mode or DBMS\_STATS.GATHER\_SYSTEM\_STATS for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.

- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## IMPORT\_COLUMN\_STATS Procedure

This procedure retrieves statistics for a particular column from the user statistics table identified by `stattab` and stores them in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_COLUMN_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    colname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    stattab          VARCHAR2,
    statid           VARCHAR2 DEFAULT NULL,
    statown          VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–53** *IMPORT\_COLUMN\_STATS Procedure Parameters*

Parameter	Description
<code>ownname</code>	Name of the schema
<code>tabname</code>	Name of the table to which this column belongs
<code>colname</code>	Name of the column or extension
<code>partname</code>	Name of the table partition. If the table is partitioned and <code>partname</code> is <code>NULL</code> , then global and partition column statistics are imported.
<code>stattab</code>	User statistics table identifier describing from where to retrieve the statistics
<code>statid</code>	Identifier to associate with these statistics within <code>stattab</code> (optional)
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
<code>force</code>	If set to <code>TRUE</code> , imports statistics even if statistics are locked

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

ORA-20005: Object statistics are locked



## Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## IMPORT\_DATABASE\_PREFS Procedure

This procedure is used to import the statistics preferences of all the tables, excluding the tables owned by Oracle. These tables can be included by passing `TRUE` for the `add_sys` parameter.

### Syntax

```
DBMS_STATS.IMPORT_DATABASE_PREFS (
  stattab      IN  VARCHAR2,
  statid       IN  VARCHAR2 DEFAULT NULL,
  statown      IN  VARCHAR2 DEFAULT NULL,
  add_sys      IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–54** *IMPORT\_DATABASE\_PREFS Procedure Parameters*

Parameter	Description
<code>stattab</code>	Statistics table name where to import the statistics
<code>statid</code>	Optional identifier to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
<code>add_sys</code>	Value <code>TRUE</code> will include the Oracle-owned tables

### Exceptions

ORA-20000: Insufficient privileges.

### Usage Notes

- To run this procedure, you need to have the `SYSDBA` role, or both `ANALYZE ANY DICTIONARY` and `ANALYZE ANY system` privileges.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.IMPORT_DATABASE_PREFS('STATTAB', statown=>'SH');
```

## IMPORT\_DATABASE\_STATS Procedure

This procedure retrieves statistics for all objects in the database from the user statistics table(s) and stores them in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_DATABASE_STATS (
  statab          VARCHAR2,
  statid          VARCHAR2 DEFAULT NULL,
  statown         VARCHAR2 DEFAULT NULL,
  no_invalidate  BOOLEAN DEFAULT to_no_invalidate_type(
                                     get_param('NO_INVALIDATE')),
  force          BOOLEAN DEFAULT FALSE,
  stat_category  VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–55** *IMPORT\_DATABASE\_STATS Procedure Parameters*

Parameter	Description
statab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within statab
statown	Schema containing statab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Overrides statistics locked at the object (table) level: <ul style="list-style-type: none"> <li>▪ TRUE - Ignores the statistics lock and imports the statistics</li> <li>▪ FALSE - The statistics will be imported only if they are not locked</li> </ul>
stat_category	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>▪ 'SYNOPSIS' - auxiliary statistics information gathered on a partitioned table when incremental is TRUE</li> </ul>

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## IMPORT\_DICTIONARY\_STATS Procedure

This procedure retrieves statistics for all dictionary schemas ('SYS', 'SYSTEM' and RDBMS component schemas) from the user statistics table and stores them in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_DICTIONARY_STATS (
  stattab          VARCHAR2,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type(
                    get_param('NO_INVALIDATE')),
  force            BOOLEAN DEFAULT FALSE,
  stat_category    VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–56** *IMPORT\_DICTIONARY\_STATS Procedure Parameters*

Parameter	Description
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	The (optional) identifier to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Overrides statistics lock at the object (table) level: <ul style="list-style-type: none"> <li>■ TRUE - Ignores the statistics lock and imports the statistics.</li> <li>■ FALSE - The statistics will be imported only if there is no lock.</li> </ul>
stat_category	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>■ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>■ 'SYNOPSIS' - auxiliary statistics information gathered on a partitioned table when incremental is TRUE</li> </ul>

### Usage Notes

- You must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege to execute this procedure.
- Oracle does not support export or import of statistics across databases of different character sets.

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

ORA-20002: Bad user statistics table, may need to upgrade it

## IMPORT\_FIXED\_OBJECTS\_STATS Procedure

This procedure retrieves statistics for fixed tables from the user statistics table(s) and stores them in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_FIXED_OBJECTS_STATS (
  stattab      VARCHAR2,
  statid       VARCHAR2 DEFAULT NULL,
  statown      VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                     get_param('NO_INVALIDATE')),
  force        BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–57** *IMPORT\_FIXED\_OBJECTS\_STATS Procedure Parameters*

Parameter	Description
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different from current schema)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Overrides statistics lock: <ul style="list-style-type: none"> <li>■ TRUE - Ignores the statistics lock and imports the statistics</li> <li>■ FALSE - The statistics will be imported only if there is no lock</li> </ul>

### Usage Notes

- You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to execute this procedure.
- Oracle does not support export or import of statistics across databases of different character sets.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

ORA-20002: Bad user statistics table, may need to upgrade it

## IMPORT\_INDEX\_STATS Procedure

This procedure retrieves statistics for a particular index from the user statistics table identified by `stattab` and stores them in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_INDEX_STATS (
  ownname      VARCHAR2,
  indname      VARCHAR2,
  partname     VARCHAR2 DEFAULT NULL,
  stattab      VARCHAR2,
  statid       VARCHAR2 DEFAULT NULL,
  statown      VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT to_no_invalidate_type(
                                     get_param('NO_INVALIDATE')),
  force        BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–58** *IMPORT\_INDEX\_STATS Procedure Parameters*

Parameter	Description
<code>ownname</code>	Name of the schema
<code>indname</code>	Name of the index
<code>partname</code>	Name of the index partition. If the index is partitioned and if <code>partname</code> is <code>NULL</code> , then global and partition index statistics are imported.
<code>stattab</code>	User statistics table identifier describing from where to retrieve the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
<code>force</code>	Imports statistics even if index statistics are locked

### Exceptions

ORA-20000: Object does not exist or insufficient privileges  
 ORA-20001: Invalid or inconsistent values in the user statistics table  
 ORA-20005: Object statistics are locked

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## IMPORT\_SCHEMA\_PREFS Procedure

This procedure is used to import the statistics preferences of all the tables owned by the specified owner name.

### Syntax

```
DBMS_STATS.IMPORT_SCHEMA_PREFS (
  ownname      IN  VARCHAR2,
  stattab      IN  VARCHAR2,
  statid       IN  VARCHAR2 DEFAULT NULL,
  statown      IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–59** *IMPORT\_SCHEMA\_PREFS Procedure Parameters*

Parameter	Description
ownname	Owner name
stattab	Statistics table name from where to import the statistics
statid	(Optional) Identifier to associate with these statistics within <code>stattab</code>
statown	Schema containing <code>stattab</code> (if other than <code>ownname</code> )

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

- To run this procedure, you need to connect as owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.IMPORT_SCHEMA_PREFS('SH', 'STAT');
```



## IMPORT\_SCHEMA\_STATS Procedure

This procedure retrieves statistics for all objects in the schema identified by ownname from the user statistics table and stores them in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_SCHEMA_STATS (
  ownname          VARCHAR2,
  stattab          VARCHAR2,
  statid           VARCHAR2 DEFAULT NULL,
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type(
                    get_param('NO_INVALIDATE')),
  force            BOOLEAN DEFAULT FALSE,
  stat_category    VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–60** *IMPORT\_SCHEMA\_STATS Procedure Parameters*

Parameter	Description
ownname	Name of the schema
stattab	User statistics table identifier describing from where to retrieve the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Overrides statistics locked at the object (table) level: <ul style="list-style-type: none"> <li>▪ TRUE - Ignores the statistics lock and imports the statistics.</li> <li>▪ FALSE - Statistics will be imported only if there is no lock.</li> </ul>
stat_category	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ 'OBJECT_STATS' - table statistics, column statistics and index statistics (Default)</li> <li>▪ 'SYNOPSIS' - auxiliary statistics information gathered on a partitioned table when incremental is TRUE</li> </ul>

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

## Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## IMPORT\_SYSTEM\_STATS Procedure

This procedure retrieves system statistics from the user statistics table, identified by `stattab`, and stores the statistics in the dictionary.

### Syntax

```
DBMS_STATS.IMPORT_SYSTEM_STATS (
  stattab      VARCHAR2,
  statid       VARCHAR2 DEFAULT NULL,
  statown      VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–61** *IMPORT\_SYSTEM\_STATS Procedure Parameters*

Parameter	Description
<code>stattab</code>	Identifier of the user statistics table where the statistics will be retrieved
<code>statid</code>	Optional identifier associated with the statistics retrieved from the <code>stattab</code>
<code>statown</code>	Schema containing <code>stattab</code> (if different from current schema)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

ORA-20002: Bad user statistics table; may need to be upgraded

ORA-20003: Unable to import system statistics

### Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## IMPORT\_TABLE\_PREFS Procedure

This procedure is used to set the statistics preferences of the specified table in the specified schema.

### Syntax

```
DBMS_STATS.IMPORT_TABLE_PREFS (
    ownname     IN  VARCHAR2,
    tablename   IN  VARCHAR2,
    stattab     IN  VARCHAR2,
    statid      IN  VARCHAR2 DEFAULT NULL,
    statown     IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–62** *IMPORT\_TABLE\_PREFS Procedure Parameters*

Parameter	Description
ownname	Owner name
tablename	Table name
stattab	Statistics table name from where to import the statistics
statid	(Optional) Identifier to associate with these statistics within stattab
statown	Schema containing stattab (if other than ownname)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

### Usage Notes

- To run this procedure, you need to connect as owner of the table, or have the `ANALYZE ANY` system privilege.
- All arguments are of type `VARCHAR2` and values are enclosed in quotes.
- Oracle does not support export or import of statistics across databases of different character sets.

### Examples

```
DBMS_STATS.IMPORT_TABLE_PREFS('SH', 'SALES', 'STAT');
```

## IMPORT\_TABLE\_STATS Procedure

This procedure retrieves statistics for a particular table from the user statistics table identified by `stattab` and stores them in the dictionary. Cascade results in all index statistics associated with the specified table being imported as well.

### Syntax

```
DBMS_STATS.IMPORT_TABLE_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    stattab          VARCHAR2,
    statid           VARCHAR2 DEFAULT NULL,
    cascade          BOOLEAN DEFAULT TRUE,
    statown         VARCHAR2 DEFAULT NULL,
    no_invalidate   BOOLEAN DEFAULT to_no_invalidate_type(
                                get_param('NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE,
    stat_category    VARCHAR2 DEFAULT DEFAULT_STAT_CATEGORY);
```

### Parameters

**Table 141–63** *IMPORT\_TABLE\_STATS Procedure Parameters*

Parameter	Description
<code>ownname</code>	Name of the schema
<code>tabname</code>	Name of the table
<code>partname</code>	Name of the table partition. If the table is partitioned and if <code>partname</code> is <code>NULL</code> , then global and partition table statistics are imported.
<code>stattab</code>	User statistics table identifier describing from where to retrieve the statistics
<code>statid</code>	Identifier (optional) to associate with these statistics within <code>stattab</code>
<code>cascade</code>	If true, column and index statistics for this table are also imported
<code>statown</code>	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a>
<code>force</code>	Imports statistics even if table statistics are locked
<code>stat_category</code>	Specifies what statistics to import, accepting multiple values separated by a comma. Values supported: <ul style="list-style-type: none"> <li>▪ <code>'OBJECT_STATS'</code> - table statistics, column statistics and index statistics (Default)</li> <li>▪ <code>'SYNOPSIS'</code> - auxiliary statistics information gathered on a partitioned table when <code>incremental</code> is <code>TRUE</code></li> </ul>

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values in the user statistics table

## Usage Notes

Oracle does not support export or import of statistics across databases of different character sets.

## LOCK\_PARTITION\_STATS Procedure

This procedure enables the user to lock statistics for a partition.

### Syntax

```
DBMS_STATS.LOCK_PARTITION_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2,  
    partname   VARCHAR2);
```

### Parameters

**Table 141-64** LOCK\_PARTITION\_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema to lock
tabname	Name of the table
partname	Partition name

## LOCK\_SCHEMA\_STATS Procedure

This procedure locks the statistics of all tables of a schema.

### Syntax

```
DBMS_STATS.LOCK_SCHEMA_STATS (  
    ownname    VARCHAR2);
```

### Parameters

**Table 141–65** LOCK\_SCHEMA\_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema to lock

### Usage Notes

See "Usage Notes" for [LOCK\\_TABLE\\_STATS Procedure](#).



## LOCK\_TABLE\_STATS Procedure

This procedure locks the statistics on the table.

### Syntax

```
DBMS_STATS.LOCK_TABLE_STATS (
  ownname    VARCHAR2,
  tabname    VARCHAR2);
```

### Parameters

**Table 141–66** LOCK\_TABLE\_STATS Procedure Parameters

Parameter	Description
ownname	The name of the schema
tabname	The name of the table

### Usage Notes

- When statistics on a table are locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The SET\_\*, DELETE\_\*, IMPORT\_\*, GATHER\_\* procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as GATHER\_SCHEMA\_STATS) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.
- This procedure either freezes the current set of the statistics or keeps the statistics empty (uncollected) to use Dynamic Sampling.
- The locked or unlocked state is not exported along with the table statistics when using EXPORT\_\*\_STATS procedures.

## MERGE\_COL\_USAGE Procedure

This procedure merges column usage information from a source database by means of a dblink into the local database. If column usage information already exists for a given table or column MERGE\_COL\_USAGE will combine both the local and the remote information.

### Syntax

```
DBMS_STATS.MERGE_COL_USAGE (  
    dblink    IN    VARCHAR2);
```

### Parameters

**Table 141–67 MERGE\_COL\_USAGE Procedure Parameters**

Parameter	Description
dblink	Name of dblink

### Usage Notes

User must be SYS to execute this procedure.

### Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Parameter dblink cannot be NULL

ORA-20002: Unable to create a TEMP table

## PREPARE\_COLUMN\_VALUES Procedures

These procedures convert user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET\_COLUMN\_STATS.

### Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
  srec      IN OUT StatRec,
  charvals  CHARARRAY);
```

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
  srec      IN OUT StatRec,
  datevals  DATEARRAY);
```

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
  srec      IN OUT StatRec,
  dblvals   DBLARRAY);
```

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
  srec      IN OUT StatRec,
  fltvals   FLTARRAY);
```

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
  srec      IN OUT StatRec,
  numvals   NUMARRAY);
```

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
  srec      IN OUT StatRec,
  rawvals   RAWARRAY);
```

### Pragmas

```
pragma restrict_references(prepare_column_values, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(prepare_column_values_nvarchar, WNDS, RNDS, WNPS,
RNPS);
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 141–68** PREPARE\_COLUMN\_VALUES Procedure Parameters

Parameter	Description
srec.epc	<p>Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid).</p> <p>The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.</p>

**Table 141–68 (Cont.) PREPARE\_COLUMN\_VALUES Procedure Parameters**

Parameter	Description
<code>srec.bkvals</code>	If you want a frequency distribution, this array contains the number of occurrences of each distinct value specified in <code>charvals</code> , <code>datevals</code> , <code>dblvals</code> , <code>fltvals</code> , <code>numvals</code> , or <code>rawvals</code> . Otherwise, it is merely an output parameter, and it must be set to <code>NULL</code> when this procedure is called.

Datatype-specific input parameters (use one) are shown in [Table 141–69](#).

**Table 141–69 Datatype-Specific Input Parameters**

Type	Description
<code>charvals</code>	The array of values when the column type is character-based. Up to the first 32 bytes of each string should be provided. Arrays must have between 2 and 256 entries, inclusive. If the datatype is fixed <code>CHAR</code> , the strings must be space-padded to 15 characters for correct normalization.
<code>datevals</code>	Array of values when the column type is date-based
<code>dblvals</code>	Array of values when the column type is double-based
<code>fltvals</code>	Array of values when the column type is float-based
<code>numvals</code>	Array of values when the column type is numeric-based
<code>rawvals</code>	Array of values when the column type is <code>RAW</code> . Up to the first 32 bytes of each strings should be provided.
<code>nvmin</code> , <code>nvmax</code>	Minimum and maximum values when the column type is national character set based. No histogram information can be provided for a column of this type. If the datatype is fixed <code>CHAR</code> , the strings must be space-padded to 15 characters for correct normalization.
<code>rwmin</code> , <code>rwmax</code>	Minimum and maximum values when the column type is <code>rowid</code> . No histogram information is provided for a column of this type.

## Output Parameters

**Table 141–70 PREPARE\_COLUMN\_VALUES Procedure Output Parameters**

Parameter	Description
<code>srec.minval</code>	Internal representation of the minimum suitable for use in a call to <code>SET_COLUMN_STATS</code>
<code>srec.maxval</code>	Internal representation of the maximum suitable for use in a call to <code>SET_COLUMN_STATS</code>
<code>srec.bkvals</code>	Array suitable for use in a call to <code>SET_COLUMN_STATS</code>
<code>srec.novals</code>	Array suitable for use in a call to <code>SET_COLUMN_STATS</code>

## Exceptions

ORA-20001: Invalid or inconsistent input values

## PREPARE\_COLUMN\_VALUES\_NVARCHAR2 Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET\_COLUMN\_STATS.

### Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES_NVARCHAR2 (
  srec      IN OUT StatRec,
  nvmin     NVARCHAR2,
  nvmax     NVARCHAR2);
```

### Pragmas

```
pragma restrict_references(prepare_column_values_nvarchar, WNDS, RNDS, WNPS,
RNPS);
```

### Parameters

**Table 141–71** PREPARE\_COLUMN\_VALUES\_NVARCHAR2 Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid).  The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.
srec.bkvals	If you want a frequency distribution, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype-specific input parameters (use one) are shown in [Table 141–69](#).

**Table 141–72** Datatype-Specific Input Parameters

Type	Description
nvmin, nvmax	The minimum and maximum values when the column type is national character set based. No histogram information can be provided for a column of this type. If the datatype is fixed CHAR, the strings must be space-padded to 15 characters for correct normalization.

## Output Parameters

**Table 141–73** *PREPARE\_COLUMN\_VALUES Procedure Output Parameters*

<b>Parameter</b>	<b>Description</b>
<code>srec.minval</code>	Internal representation of the minimum suitable for use in a call to <code>SET_COLUMN_STATS</code>
<code>srec.maxval</code>	Internal representation of the maximum suitable for use in a call to <code>SET_COLUMN_STATS</code>
<code>srec.bkvals</code>	Array suitable for use in a call to <code>SET_COLUMN_STATS</code> .
<code>srec.novals</code>	Array suitable for use in a call to <code>SET_COLUMN_STATS</code>

## Exceptions

ORA-20001: Invalid or inconsistent input values

## PREPARE\_COLUMN\_VALUES\_ROWID Procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage using SET\_COLUMN\_STATS.

### Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES_ROWID (
  srec IN OUT StatRec,
  rwmn          ROWID,
  rwmax          ROWID);
```

### Pragmas

```
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 141–74** PREPARE\_COLUMN\_VALUES\_ROWID Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid).  The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.
srec.bkvals	If you want a frequency distribution, this array contains the number of occurrences of each distinct value specified in charvals, datevals, dblvals, fltvals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype-specific input parameters (use one) are shown in [Table 141–69](#).

**Table 141–75** Datatype-Specific Input Parameters

Type	Description
rwmn, rwmax	Minimum and maximum values when the column type is rowid. No histogram information is provided for a column of this type.

### Output Parameters

**Table 141–76** PREPARE\_COLUMN\_VALUES Procedure Output Parameters

Parameter	Description
srec.minval	Internal representation of the minimum suitable for use in a call to SET_COLUMN_STATS.

**Table 141-76 (Cont.) PREPARE\_COLUMN\_VALUES Procedure Output Parameters**

<b>Parameter</b>	<b>Description</b>
<code>srec.maxval</code>	Internal representation of the maximum suitable for use in a call to <code>SET_COLUMN_STATS</code> .
<code>srec.bkvals</code>	Array suitable for use in a call to <code>SET_COLUMN_STATS</code> .
<code>srec.novals</code>	Array suitable for use in a call to <code>SET_COLUMN_STATS</code> .

## Exceptions

ORA-20001: Invalid or inconsistent input values



## PUBLISH\_PENDING\_STATS Procedure

This procedure is used to publish the statistics gathered and stored as pending.

### Syntax

```
DBMS_STATS.PUBLISH_PENDING_STATS (
  ownname          IN VARCHAR2 DEFAULT USER,
  tabname          IN VARCHAR2,
  no_invalidate    BOOLEAN DEFAULT
    to_no_invalidate_type(get_param('NO_INVALIDATE')),
  force            IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141-77 PUBLISH\_PENDING\_STATS Procedure Parameters**

Parameter	Description
ownname	Owner name
tabname	Table name
no_invalidate	Do not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	If TRUE, will override the lock

### Exceptions

ORA-20000: Insufficient privileges

### Usage Notes

- If the parameter `tabname` is NULL then publish applies to all tables of the specified schema.
- The default owner/schema is the user who runs the procedure.
- To run this procedure, you need to have the same privilege for gathering statistics on the tables that will be touched by this procedure.

### Examples

```
DBMS_STATS.PUBLISH_PENDING_STATS ('SH', null);
```

## PURGE\_STATS Procedure

This procedure purges old versions of statistics saved in the dictionary. To run this procedure, you must have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.

### Syntax

```
DBMS_STATS.PURGE_STATS(  
    before_timestamp          TIMESTAMP WITH TIME ZONE);
```

### Parameters

**Table 141–78** PURGE\_STATS Procedure Parameters

Parameter	Description
before_timestamp	Versions of statistics saved before this timestamp are purged. If NULL, it uses the purging policy used by automatic purge. The automatic purge deletes all history older than the older of (current time - statistics history retention) and (time of recent analyze in the system - 1). The statistics history retention value can be changed using ALTER_STATS_HISTORY_RETENTION Procedure. The default is 31 days.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

## RESET\_GLOBAL\_PREF\_DEFAULTS Procedure

This procedure sets global preference, such as `CASCADE`, `ESTIMATE_PERCENT` and `GRANULARITY`, to default values. This reverses the global preferences set by the [SET\\_GLOBAL\\_PREFS Procedure](#).

### Syntax

```
DBMS_STATS.RESET_GLOBAL_PREF_DEFAULTS;
```

## RESET\_PARAM\_DEFAULTS Procedure

---

---

**Note:** This subprogram has been replaced by improved technology and is maintained only for purposes of backward compatibility. In this case, use the [RESET\\_GLOBAL\\_PREF\\_DEFAULTS Procedure](#).

See also [Deprecated Subprograms](#) on page 141-12.

---

---

This procedure resets the default values of all parameters to Oracle recommended values.

### Syntax

```
DBMS_STATS.RESET_PARAM_DEFAULTS;
```

## RESTORE\_DATABASE\_STATS Procedure

This procedure restores statistics of all tables of the database as of a specified timestamp (`as_of_timestamp`).

### Syntax

```
DBMS_STATS.RESTORE_DATABASE_STATS (
  as_of_timestamp    TIMESTAMP WITH TIME ZONE,
  force              BOOLEAN DEFAULT FALSE,
  no_invalidate      BOOLEAN DEFAULT to_no_invalidate_type
                    (GET_PARAM('NO_INVALIDATE')));
```

### Parameters

**Table 141–79 RESTORE\_DATABASE\_STATS Procedure Parameters**

Parameter	Description
<code>as_of_timestamp</code>	The timestamp to which to restore statistics
<code>force</code>	Restores statistics even if their statistics are locked
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> . to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available

## RESTORE\_DICTIONARY\_STATS Procedure

This procedure restores statistics of all dictionary tables (tables of 'SYS', 'SYSTEM' and RDBMS component schemas) as of a specified timestamp (`as_of_timestamp`).

### Syntax

```
DBMS_STATS.RESTORE_DICTIONARY_STATS (
  as_of_timestamp      TIMESTAMP WITH TIME ZONE,
  force                BOOLEAN DEFAULT FALSE,
  no_invalidate       BOOLEAN DEFAULT to_no_invalidate_type
                      (GET_PARAM('NO_INVALIDATE')));
```

### Parameters

**Table 141–80 RESTORE\_DICTIONARY\_STATS Procedure Parameters**

Parameter	Description
<code>as_of_timestamp</code>	Timestamp to which to restore statistics
<code>force</code>	Restores statistics even if their statistics are locked
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

### Usage Notes

To run this procedure, you must have the `SYSDBA` or both `ANALYZE ANY DICTIONARY` and `ANALYZE ANY` system privilege.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available

## RESTORE\_FIXED\_OBJECTS\_STATS Procedure

This procedure restores statistics of all fixed tables as of a specified timestamp (`as_of_timestamp`).

### Syntax

```
DBMS_STATS.RESTORE_FIXED_OBJECTS_STATS(
  as_of_timestamp    TIMESTAMP WITH TIME ZONE,
  force              BOOLEAN DEFAULT FALSE,
  no_invalidate      BOOLEAN DEFAULT to_no_invalidate_type
                    (GET_PARAM('NO_INVALIDATE')));
```

### Parameters

**Table 141–81 RESTORE\_FIXED\_OBJECTS\_STATS Procedure Parameters**

Parameter	Description
<code>as_of_timestamp</code>	The timestamp to which to restore statistics
<code>force</code>	Restores statistics even if their statistics are locked
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

### Usage Notes

To run this procedure, you must have the `SYSDBA` or `ANALYZE ANY DICTIONARY` system privilege.

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available

## RESTORE\_SCHEMA\_STATS Procedure

This procedure restores statistics of all tables of a schema as of a specified timestamp (`as_of_timestamp`).

### Syntax

```
DBMS_STATS.RESTORE_SCHEMA_STATS (
  ownname          VARCHAR2,
  as_of_timestamp  TIMESTAMP WITH TIME ZONE,
  force            BOOLEAN DEFAULT FALSE,
  no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type
                  (GET_PARAM('NO_INVALIDATE')));
```

### Parameters

**Table 141–82** RESTORE\_SCHEMA\_STATS Procedure Parameters

Parameter	Description
<code>ownname</code>	Schema of the tables for which the statistics are to be restored
<code>as_of_timestamp</code>	The timestamp to which to restore statistics
<code>force</code>	Restores statistics even if their statistics are locked
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available



## RESTORE\_SYSTEM\_STATS Procedure

This procedure restores system statistics as of a specified timestamp (`as_of_timestamp`).

### Syntax

```
DBMS_STATS.RESTORE_SCHEMA_STATS (  
    as_of_timestamp          TIMESTAMP WITH TIME ZONE);
```

### Parameters

**Table 141–83** *RESTORE\_SYSTEM\_STATS Procedure Parameters*

Parameter	Description
<code>as_of_timestamp</code>	The timestamp to which to restore statistics

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available

## RESTORE\_TABLE\_STATS Procedure

This procedure restores statistics of a table as of a specified timestamp (`as_of_timestamp`). The procedure will restore statistics of associated indexes and columns as well. If the table statistics were locked at the specified timestamp the procedure will lock the statistics. The procedure will not restore user defined statistics.

### Syntax

```
DBMS_STATS.RESTORE_TABLE_STATS (
  ownname          VARCHAR2,
  tabname          VARCHAR2,
  as_of_timestamp  TIMESTAMP WITH TIME ZONE,
  restore_cluster_index  BOOLEAN DEFAULT FALSE,
  force            BOOLEAN DEFAULT FALSE,
  no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type
                                     (GET_PARAM('NO_INVALIDATE')));
```

### Parameters

**Table 141–84 RESTORE\_TABLE\_STATS Procedure Parameters**

Parameter	Description
<code>ownname</code>	The schema of the table for which the statistics are to be restored
<code>tabname</code>	The table name
<code>as_of_timestamp</code>	The timestamp to which to restore statistics
<code>restore_cluster_index</code>	If the table is part of a cluster, restore statistics of the cluster index if set to <code>TRUE</code>
<code>force</code>	Restores statistics even if the table statistics are locked. If the table statistics were not locked at the specified timestamp, it unlocks the statistics.
<code>no_invalidate</code>	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent values

ORA-20006: Unable to restore statistics, statistics history not available

## SEED\_COL\_USAGE Procedure

This procedure iterates over the SQL statements in the specified SQL tuning set, compiles them and seeds column usage information for the columns that appear in these statements.

### Syntax

```
DBMS_STATS.SEED_COL_USAGE (  
    sqlset_name    IN    VARCHAR2,  
    owner_name     IN    VARCHAR2,  
    time_limit     IN    POSITIVE DEFAULT NULL);
```

### Parameters

**Table 141–85 SEED\_COL\_USAGE Procedure Parameters**

Parameter	Description
sqlset_name	Name of the SQL tuning set
owner_name	Owner of the SQL tuning set
time_limit	Time limit (in seconds)

### Exceptions

ORA-20000: Insufficient privileges

## SET\_COLUMN\_STATS Procedures

This procedure sets column-related information. In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is NULL, the statistics type associated with the index or column is stored.

### Syntax

```
DBMS_STATS.SET_COLUMN_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    colname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    stattab          VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    distcnt          NUMBER DEFAULT NULL,
    density          NUMBER DEFAULT NULL,
    nullcnt          NUMBER DEFAULT NULL,
    srec             StatRec DEFAULT NULL,
    avgclen          NUMBER DEFAULT NULL,
    flags            NUMBER DEFAULT NULL,
    statown          VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type(
                                get_param('NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_COLUMN_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    colname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    stattab          VARCHAR2 DEFAULT NULL,
    statid           VARCHAR2 DEFAULT NULL,
    ext_stats        RAW,
    stattypown       VARCHAR2 DEFAULT NULL,
    stattypname      VARCHAR2 DEFAULT NULL,
    statown          VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN DEFAULT to_no_invalidate_type(
                                get_param('NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–86 SET\_COLUMN\_STATS Procedure Parameters**

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column or extension
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.

**Table 141–86 (Cont.) SET\_COLUMN\_STATS Procedure Parameters**

Parameter	Description
stattab	User statistics table identifier describing where to store the statistics. If <code>stattab</code> is <code>NULL</code> , then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within <code>stattab</code> (Only pertinent if <code>stattab</code> is not <code>NULL</code> )
ext_stats	User-defined statistics
stattypown	Schema of the statistics type
stattypname	Name of the statistics type
distcnt	Number of distinct values
density	Column density. If this value is <code>NULL</code> and if <code>distcnt</code> is not <code>NULL</code> , then density is derived from <code>distcnt</code> .
nullcnt	Number of <code>NULL</code> s
srec	<code>StatRec</code> structure filled in by a call to <code>PREPARE_COLUMN_VALUES</code> or <code>GET_COLUMN_STATS</code>
avgclen	Average length for the column (in bytes)
flags	For internal Oracle use (should be left as <code>NULL</code> )
statown	Schema containing <code>stattab</code> (if different than <code>ownname</code> )
no_invalidate	Does not invalidate the dependent cursors if set to <code>TRUE</code> . The procedure invalidates the dependent cursors immediately if set to <code>FALSE</code> . Use <code>DBMS_STATS.AUTO_INVALIDATE</code> . to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
force	Sets the values even if statistics of the column are locked

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or inconsistent input values

ORA-20005: Object statistics are locked

## SET\_DATABASE\_PREFS Procedure

This procedure is used to set the statistics preferences of all the tables, excluding the tables owned by Oracle. These tables can be included by passing `TRUE` for the `add_sys` parameter.

### Syntax

```
DBMS_STATS.SET_DATABASE_PREFS (
    pname          IN   VARCHAR2,
    pvalue        IN   VARCHAR2,
    add_sys       IN   BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 141–87 SET\_DATABASE\_PREFS Procedure Parameters**

Parameter	Description
<code>pname</code>	<p>Preference name. The default value for following parameters can be set:</p> <ul style="list-style-type: none"> <li>▪ CASCADE</li> <li>▪ DEGREE</li> <li>▪ ESTIMATE_PERCENT</li> <li>▪ METHOD_OPT</li> <li>▪ NO_INVALIDATE</li> <li>▪ GRANULARITY</li> <li>▪ PUBLISH</li> <li>▪ INCREMENTAL</li> <li>▪ STALE_PERCENT</li> </ul> <p>. CASCADE - Determines whether or not index statistics are collected as part of gathering table statistics.</p> <p>. DEGREE - Determines degree of parallelism used for gathering statistics.</p> <p>. ESTIMATE_PERCENT - Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant <code>DBMS_STATS.AUTO_SAMPLE_SIZE</code> to have Oracle determine the appropriate sample size for good statistics. This is the default.</p>

**Table 141–87 (Cont.) SET\_DATABASE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>METHOD_OPT - Controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> <li>- column_name : Name of a column</li> <li>- extension : Can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO.</p>
.	<p>NO_INVALIDATE - Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default.</p>
.	<p>GRANULARITY - Determines granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics.</p>

**Table 141–87 (Cont.) SET\_DATABASE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>PUBLISH - Determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
.	<p>INCREMENTAL - Determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ the INCREMENTAL value for the partitioned table is set to TRUE;</li> <li>■ the PUBLISH value for the partitioned table is set to TRUE;</li> <li>■ the user specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table.</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
.	<p>STALE_PERCENT - Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%.</p>
pvalue	Preference value. If NULL is specified, it will set the Oracle default value.s
add_sys	Value TRUE will include the Oracle-owned tables

## Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Invalid or illegal input values

## Usage Notes

- To run this procedure, you need to have the SYSDBA role or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.
- Both arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

## Examples

```
DBMS_STATS.SET_DATABASE_PREFS('CASCADE', 'DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_DATABASE_PREFS('ESTIMATE_PERCENT', '9');
DBMS_STATS.SET_DATABASE_PREFS('DEGREE', '99');
```



## SET\_GLOBAL\_PREFS Procedure

This procedure is used to set the global statistics preferences.

### Syntax

```
DBMS_STATS.SET_GLOBAL_PREFS (
  pname      IN   VARCHAR2,
  pvalue     IN   VARCHAR2);
```

### Parameters

**Table 141–88** SET\_GLOBAL\_PREFS Procedure Parameters

Parameter	Description
pname	<p>Preference name. The default value for the following preferences can be set:</p> <ul style="list-style-type: none"> <li>▪ AUTOSTATS_TARGET</li> <li>▪ CASCADE</li> <li>▪ DEGREE</li> <li>▪ ESTIMATE_PERCENT</li> <li>▪ METHOD_OPT</li> <li>▪ NO_INVALIDATE</li> <li>▪ GRANULARITY</li> <li>▪ PUBLISH</li> <li>▪ INCREMENTAL</li> <li>▪ STALE_PERCENT</li> </ul> <p>AUTOSTATS_TARGET - This preference is applicable only for auto statistics collection. The value of this parameter controls the objects considered for stats collection. It takes the following values:</p> <ul style="list-style-type: none"> <li>▪ 'ALL' - Statistics collected for all objects in system</li> <li>▪ 'ORACLE' - Statistics collected for all Oracle owned objects</li> <li>▪ 'AUTO' - Oracle decides on which objects to collect statistics</li> </ul> <p>CASCADE - Determines whether or not index statistics are collected as part of gathering table statistics</p> <p>DEGREE - Determines degree of parallelism used for gathering statistics</p> <p>ESTIMATE_PERCENT - Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.</p>

**Table 141–88 (Cont.) SET\_GLOBAL\_PREFS Procedure Parameters**

Parameter	Description
.	<p>METHOD_OPT - The value controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms.</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> <li>- column_name : Name of a column</li> <li>- extension : Can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO.</p>
.	<p>NO_INVALIDATE - Controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default.</p>
.	<p>GRANULARITY - Determines granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - gathers partition-level statistics</p> <p>'SUBPARTITION' - gathers subpartition-level statistics.</p>

**Table 141–88 (Cont.) SET\_GLOBAL\_PREFS Procedure Parameters**

Parameter	Description
.	<p>PUBLISH - This value determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
.	<p>INCREMENTAL - This value determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ INCREMENTAL value for the partitioned table is set to TRUE</li> <li>■ PUBLISH value for the partitioned table is set to TRUE;</li> <li>■ User specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
.	<p>STALE_PERCENT - Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%.</p>
pvalue	<p>Preference value. If NULL is specified, it will set the Oracle default value.s</p>

## Exceptions

ORA-20000: Insufficient privileges

ORA-20001: Invalid or illegal input values

## Usage Notes

- This setting is honored only if there is no preference specified for the table to be analyzed.
- To run this procedure, you need to have the SYSDBA or both ANALYZE ANY DICTIONARY and ANALYZE ANY system privilege.
- Both arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

## Examples

```
DBMS_STATS.SET_GLOBAL_PREFS('ESTIMATE_PERCENT', '9');
DBMS_STATS.SET_GLOBAL_PREFS('DEGREE', '99');
```

## SET\_INDEX\_STATS Procedures

These procedures set index-related information. In the version of this procedure that deals with user-defined statistics, the statistics type specified is the type to store in the dictionary, in addition to the actual user-defined statistics. If this statistics type is NULL, the statistics type associated with the index or column is stored.

### Syntax

```
DBMS_STATS.SET_INDEX_STATS (
  ownname      VARCHAR2,
  indname      VARCHAR2,
  partname     VARCHAR2  DEFAULT NULL,
  stattab      VARCHAR2  DEFAULT NULL,
  statid       VARCHAR2  DEFAULT NULL,
  numrows      NUMBER    DEFAULT NULL,
  numlblks     NUMBER    DEFAULT NULL,
  numdist      NUMBER    DEFAULT NULL,
  avglblk      NUMBER    DEFAULT NULL,
  avgdblks     NUMBER    DEFAULT NULL,
  clstfct      NUMBER    DEFAULT NULL,
  indlevel     NUMBER    DEFAULT NULL,
  flags        NUMBER    DEFAULT NULL,
  statown      VARCHAR2  DEFAULT NULL,
  no_invalidate BOOLEAN  DEFAULT to_no_invalidate_type(
                                get_param('NO_INVALIDATE')),
  guessq       NUMBER    DEFAULT NULL,
  cachedblk    NUMBER    DEFAULT NULL,
  cachehit     NUMBER    DEFUALT NULL,
  force        BOOLEAN  DEFAULT FALSE);
```

Use the following for user-defined statistics:

```
DBMS_STATS.SET_INDEX_STATS (
  ownname      VARCHAR2,
  indname      VARCHAR2,
  partname     VARCHAR2  DEFAULT NULL,
  stattab      VARCHAR2  DEFAULT NULL,
  statid       VARCHAR2  DEFAULT NULL,
  ext_stats    RAW,
  statypown    VARCHAR2  DEFAULT NULL,
  statypname   VARCHAR2  DEFAULT NULL,
  statown      VARCHAR2  DEFAULT NULL,
  no_invalidate BOOLEAN  DEFAULT to_no_invalidate_type(
                                get_param('NO_INVALIDATE')),
  cachedblk    NUMBER    DEFAULT NULL,
  cachehit     NUMBER    DEFUALT NULL,
  force        BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–89** SET\_INDEX\_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema
indname	Name of the index

**Table 141–89 (Cont.) SET\_INDEX\_STATS Procedure Parameters**

Parameter	Description
partname	Name of the index partition in which to store the statistics. If the index is partitioned and if partname is NULL, then the statistics are stored at the global index level.
stattab	User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
ext_stats	User-defined statistics
stattypown	Schema of the statistics type
stattypname	Name of the statistics type
numrows	Number of rows in the index (partition)
numlblks	Number of leaf blocks in the index (partition)
numdist	Number of distinct keys in the index (partition)
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition). If not provided, then this value is derived from numlblks and numdist.
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition). If not provided, then this value is derived from clstfct and numdist.
clstfct	See clustering_factor column of the all_indexes view for a description
indlevel	Height of the index (partition)
flags	For internal Oracle use (should be left as NULL)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
guessq	Guess quality. See the pct_direct_access column of the all_indexes view for a description.
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)
cachehit	The average cache hit ratio for the segment (index/table/index partition/table partition)
force	Sets the values even if statistics of the index are locked

## Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.

- Oracle maintains `cachedblk` and `cachehit` at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls `DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS` procedure for auto mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.
- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid input value

ORA-20005: Object statistics are locked

## SET\_PARAM Procedure

---

**Note:** This subprogram has been replaced by improved technology and is maintained only for purposes of backward compatibility. In this case, use the [SET\\_GLOBAL\\_PREFS Procedure](#).

See also [Deprecated Subprograms](#) on page 141-12.

---

This procedure sets default values for parameters of DBMS\_STATS procedures. You can use the GET\_PARAM Function to get the current default value of a parameter.

### Syntax

```
DBMS_STATS.SET_PARAM (
  pname      IN   VARCHAR2,
  pval       IN   VARCHAR2);
```

### Parameters

**Table 141–90 SET\_PARAM Procedure Parameters**

Parameter	Description
pname	<p>The parameter name The default value for following parameters can be set.</p> <ul style="list-style-type: none"> <li>■ CASCADE - The default value for CASCADE set by SET_PARAM is not used by export/import procedures.It is used only by gather procedures.</li> <li>■ DEGREE</li> <li>■ ESTIMATE_PERCENT</li> <li>■ METHOD_OPT</li> <li>■ NO_INVALIDATE</li> <li>■ GRANULARITY</li> <li>■ AUTOSTATS_TARGET - This parameter is applicable only for auto statistics collection. The value of this parameter controls the objects considered for statistics collection (see pval)</li> </ul>
pval	<p>The parameter value. If NULL is specified, it will set the default value determined by Oracle. When pname is AUTOSTATS_TARGET, the following are valid values:</p> <ul style="list-style-type: none"> <li>■ 'ALL' - Statistics are collected for all objects in the system</li> <li>■ 'ORACLE' - Statistics are collected for all Oracle owned objects</li> <li>■ 'AUTO' - Oracle decides for which objects to collect statistics</li> </ul>

### Usage Notes

- To run this procedure, you must have the SYSDBA or both the ANALYZE ANY DICTIONARY and ANALYZE ANY system privileges.
- Note that both arguments are of type VARCHAR2 and the values need to be enclosed in quotes even when they represent numbers.

- Note also the difference between NULL and 'NULL':
  - When NULL is unquoted, this sets the parameter to the value Oracle recommends.
  - In the case of the quoted 'NULL', this sets the value of the parameter to NULL.

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or illegal input value

## Examples

```
DBMS_STATS.SET_PARAM('CASCADE', 'DBMS_STATS.AUTO_CASCADE');  
DBMS_STATS.SET_PARAM('ESTIMATE_PERCENT', '5');  
DBMS_STATS.SET_PARAM('DEGREE', 'NULL');
```



## SET\_SCHEMA\_PREFS Procedure

This procedure is used to set the statistics preferences of all the tables owned by the specified owner name.

### Syntax

```
DBMS_STATS.SET_SCHEMA_PREFS (
    ownname    IN    VARCHAR2,
    pname      IN    VARCHAR2,
    pvalue     IN    VARCHAR2);
```

### Parameters

**Table 141–91 SET\_SCHEMA\_PREFS Procedure Parameters**

Parameter	Description
ownname	Owner name
pname	Preference name. The default value for following parameters can be set: <ul style="list-style-type: none"> <li>■ CASCADE</li> <li>■ DEGREE</li> <li>■ ESTIMATE_PERCENT</li> <li>■ METHOD_OPT</li> <li>■ NO_INVALIDATE</li> <li>■ GRANULARITY</li> <li>■ PUBLISH</li> <li>■ INCREMENTAL</li> <li>■ STALE_PERCENT</li> </ul>
pvalue	Preference value. If NULL is specified, it will set the Oracle default value.s

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or illegal input value

### Usage Notes

- To run this procedure, you need to connect as owner, or have the SYSDBA privilege, or have the ANALYZE ANY system privilege.
- Both arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

### Examples

```
DBMS_STATS.SET_SCHEMA_PREFS('SH', 'CASCADE', 'DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_SCHEMA_PREFS('SH', 'ESTIMATE_PERCENT', '9');
DBMS_STATS.SET_SCHEMA_PREFS('SH', 'DEGREE', '99');
```

## SET\_SYSTEM\_STATS Procedure

This procedure sets systems statistics.

### Syntax

```
DBMS_STATS.SET_SYSTEM_STATS (
  pname          VARCHAR2,
  pvalue         NUMBER,
  statab        IN   VARCHAR2 DEFAULT NULL,
  statid        IN   VARCHAR2 DEFAULT NULL,
  statown       IN   VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 141–92 SET\_SYSTEM\_STATS Procedure Parameters**

Parameter	Description
pname	The parameter name to get, which can have one of the following values: <ul style="list-style-type: none"> <li>■ <code>iotfrspeed</code>—I/O transfer speed in bytes for each millisecond</li> <li>■ <code>ioseektim</code> - Seek time + latency time + operating system overhead time, in milliseconds</li> <li>■ <code>sreadtim</code> - Average time to read single block (random read), in milliseconds</li> <li>■ <code>mreadtim</code> - Average time to read an mbrc block at once (sequential read), in milliseconds</li> <li>■ <code>cpuspeed</code> - Average number of CPU cycles for each second, in millions, captured for the workload (statistics collected using 'INTERVAL' or 'START' and 'STOP' options)</li> <li>■ <code>cpuspeednw</code> - Average number of CPU cycles for each second, in millions, captured for the noworkload (statistics collected using 'NOWORKLOAD' option).</li> <li>■ <code>mbrc</code> - Average multiblock read count for sequential read, in blocks</li> <li>■ <code>maxthr</code> - Maximum I/O system throughput, in bytes/second</li> <li>■ <code>slavethr</code> - Average slave I/O throughput, in bytes/second</li> </ul>
pvalue	Parameter value to get
statab	Identifier of the user statistics table where the statistics will be obtained. If statab is null, the statistics will be obtained from the dictionary.
statid	Optional identifier associated with the statistics saved in the statab
statown	Schema containing statab (if different from current schema)

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid input value

ORA-20002: Bad user statistics table; may need to be upgraded

ORA-20003: Unable to set system statistics

ORA-20004: Parameter does not exist

## SET\_TABLE\_PREFS Procedure

This procedure is used to set the statistics preferences of the specified table in the specified schema.

### Syntax

```
DBMS_STATS.SET_TABLE_PREFS (
    ownname    IN  VARCHAR2,
    tabname    IN  VARCHAR2,
    pname      IN  VARCHAR2,
    pvalue     IN  VARCHAR2);
```

### Parameters

**Table 141–93 SET\_TABLE\_PREFS Procedure Parameters**

Parameter	Description
ownname	Owner name
tabname	Table name
pname	Preference name. The default value for following preferences can be set: <ul style="list-style-type: none"> <li>▪ CASCADE</li> <li>▪ DEGREE</li> <li>▪ ESTIMATE_PERCENT</li> <li>▪ METHOD_OPT</li> <li>▪ NO_INVALIDATE</li> <li>▪ GRANULARITY</li> <li>▪ PUBLISH</li> <li>▪ INCREMENTAL</li> <li>▪ STALE_PERCENT</li> </ul>
.	CASCADE - Determines whether or not index statistics are collected as part of gathering table statistics .
.	DEGREE - Determines degree of parallelism used for gathering statistics.
.	ESTIMATE_PERCENT - Determines the percentage of rows to estimate. The valid range is [0.000001,100]. Use the constant DBMS_STATS.AUTO_SAMPLE_SIZE to have Oracle determine the appropriate sample size for good statistics. This is the default.

**Table 141–93 (Cont.) SET\_TABLE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>METHOD_OPT - Controls column statistics collection and histogram creation. It accepts either of the following options, or both in combination:</p> <ul style="list-style-type: none"> <li>■ FOR ALL [INDEXED   HIDDEN] COLUMNS [size_clause]</li> <li>■ FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]</li> </ul> <p>size_clause is defined as size_clause := SIZE {integer   REPEAT   AUTO   SKEWONLY}</p> <p>column is defined as column := column_name   extension name   extension</p> <ul style="list-style-type: none"> <li>- integer : Number of histogram buckets. Must be in the range [1,254].</li> <li>- REPEAT : Collects histograms only on the columns that already have histograms</li> <li>- AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.</li> <li>- SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.</li> <li>- column_name : Name of a column</li> <li>- extension : Can be either a column group in the format of (column_name, column_name [, ...]) or an expression</li> </ul> <p>The default is FOR ALL COLUMNS SIZE AUTO.</p>
.	<p>NO_INVALIDATE - The value controls the invalidation of dependent cursors of the tables for which statistics are being gathered. Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE to have Oracle decide when to invalidate dependent cursors. This is the default.</p>
.	<p>GRANULARITY - Determines granularity of statistics to collect (only pertinent if the table is partitioned).</p> <p>'ALL' - Gathers all (subpartition, partition, and global) statistics</p> <p>'AUTO' - Determines the granularity based on the partitioning type. This is the default value.</p> <p>'DEFAULT' - Gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.</p> <p>'GLOBAL' - Gathers global statistics</p> <p>'GLOBAL AND PARTITION' - Gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.</p> <p>'PARTITION' - Gathers partition-level statistics</p> <p>'SUBPARTITION' - Gathers subpartition-level statistics.</p>

**Table 141–93 (Cont.) SET\_TABLE\_PREFS Procedure Parameters**

Parameter	Description
.	<p>PUBLISH - Determines whether or not newly gathered statistics will be published once the gather job has completed. Prior to Oracle Database 11g, Release 1 (11.1), once a statistic gathering job completed the new statistics were automatically published into the dictionary tables. The user now has the ability to gather statistics but not publish them immediately. This allows the DBA to test the new statistics before publishing them.</p>
.	<p>INCREMENTAL - Determines whether or not the global statistics of a partitioned table will be maintained without doing a full table scan. With partitioned tables it is very common to load new data into a new partition. As new partitions are added and data loaded, the global table statistics need to be kept up to date. Oracle will update the global table statistics by scanning only the partitions that have been changed instead of the entire table if the following conditions hold:</p> <ul style="list-style-type: none"> <li>■ INCREMENTAL value for the partitioned table is set to TRUE;</li> <li>■ PUBLISH value for the partitioned table is set to TRUE;</li> <li>■ User specifies AUTO_SAMPLE_SIZE for ESTIMATE_PERCENT and AUTO for GRANULARITY when gathering statistics on the table.</li> </ul> <p>If the INCREMENTAL value for the partitioned table was set to FALSE (default value), a full table scan is used to maintain the global statistics which is a much more resource intensive and time-consuming operation for large tables.</p>
.	<p>STALE_PERCENT - Determines the percentage of rows in a table that have to change before the statistics on that table are deemed stale and should be regathered. The valid domain for stale_percent is non-negative numbers. The default value is 10%.</p>
pvalue	<p>Preference value. If NULL is specified, it will set the Oracle default value.</p>

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid or illegal input values

## Usage Notes

- To run this procedure, you need to connect as owner of the table or should have the ANALYZE ANY system privilege.
- All arguments are of type VARCHAR2 and values are enclosed in quotes, even when they represent numbers.

## Examples

```
DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'CASCADE', 'DBMS_STATS.AUTO_CASCADE');
DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'ESTIMATE_PERCENT', '9');
DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'DEGREE', '99');
```

## SET\_TABLE\_STATS Procedure

This procedure sets table-related information.

### Syntax

```
DBMS_STATS.SET_TABLE_STATS (
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    stattab         VARCHAR2 DEFAULT NULL,
    statid          VARCHAR2 DEFAULT NULL,
    numrows         NUMBER  DEFAULT NULL,
    numblks         NUMBER  DEFAULT NULL,
    avgrlen         NUMBER  DEFAULT NULL,
    flags           NUMBER  DEFAULT NULL,
    statown         VARCHAR2 DEFAULT NULL,
    no_invalidate   BOOLEAN  DEFAULT to_no_invalidate_type (
                                get_param('NO_INVALIDATE')),
    cachedblk      NUMBER  DEFAULT NULL,
    cachehit        NUMBER  DEFAULT NULL,
    force           BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 141–94 SET\_TABLE\_STATS Procedure Parameters**

Parameter	Description
ownname	Name of the schema
tabname	Name of the table
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.
stattab	User statistics table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL)
numrows	Number of rows in the table (partition)
numblks	Number of blocks the table (partition) occupies
avgrlen	Average row length for the table (partition)
flags	For internal Oracle use (should be left as NULL)
statown	Schema containing stattab (if different than ownname)
no_invalidate	Does not invalidate the dependent cursors if set to TRUE. The procedure invalidates the dependent cursors immediately if set to FALSE. Use DBMS_STATS.AUTO_INVALIDATE. to have Oracle decide when to invalidate dependent cursors. This is the default. The default can be changed using the <a href="#">SET_DATABASE_PREFS Procedure</a> , <a href="#">SET_GLOBAL_PREFS Procedure</a> , <a href="#">SET_SCHEMA_PREFS Procedure</a> and <a href="#">SET_TABLE_PREFS Procedure</a> .
cachedblk	The average number of blocks in the buffer cache for the segment (index/table/index partition/table partition)

**Table 141–94 (Cont.) SET\_TABLE\_STATS Procedure Parameters**

Parameter	Description
<code>cachehit</code>	The average cache hit ratio for the segment (index/table/index partition/table partition)
<code>force</code>	Sets the values even if statistics of the table are locked

## Usage Notes

- The Optimizer uses the cached data to estimate number of cached blocks for index or statistics table access. The total cost of the operation will be combined from the I/O cost of reading not cached blocks from disk, the CPU cost of getting cached blocks from the buffer cache, and the CPU cost of processing the data.
- Oracle maintains `cachedblk` and `cachehit` at all times but uses correspondent caching statistics for optimization as part of the table and index statistics only when the user calls `DBMS_STATS.GATHER_[TABLE/INDEX/SCHEMA/DATABASE]_STATS` procedure for auto mode or `DBMS_STATS.GATHER_SYSTEM_STATS` for manual mode. In order to prevent the user from utilizing inaccurate and unreliable data, the optimizer will compute a 'confidence factor' for each `cachehit` and a `cachedblk` for each object. If the 'confidence factor' for the value meets confidence criteria, this value will be used, otherwise the defaults will be used.
- The automatic maintenance algorithm for object caching statistics assumes that there is only one major workload for the system and adjusts statistics to this workload, ignoring other "minor" workloads. If this is not the case, you must use manual mode for maintaining object caching statistics.
- The object caching statistics maintenance algorithm for auto mode prevents you from using statistics in the following situations
  - When not enough data has been analyzed, such as when an object has been recently create
  - When the system does not have one major workload resulting in averages not corresponding to real values.

## Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Invalid input value

ORA-20005: Object statistics are locked



## SHOW\_EXTENDED\_STATS\_NAME Function

This function returns the name of the statistics entry that is created for the user-specified extension. It raises an error if no extension has been created.

### Syntax

```
DBMS_STATS.SHOW_EXTENDED_STATS_NAME (
  ownname   VARCHAR2,
  tabname   VARCHAR2,
  extension VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 141–95** *SHOW\_EXTENDED\_STATS\_NAME Function Parameters*

Parameter	Description
ownname	Owner name of a table
tabname	Name of the table
extension	Can be either a column group or an expression. Suppose the specified table has two column c1, c2. An example column group can be "(c1, c2)" and an example expression can be "(c1 + c2)".

### Exceptions

ORA-20000: Object does not exist or insufficient privileges

ORA-20001: Error when processing extension

## UNLOCK\_PARTITION\_STATS Procedure

This procedure enables the user to unlock statistics for a partition.

### Syntax

```
DBMS_STATS.UNLOCK_PARTITION_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2,  
    partname   VARCHAR2);
```

### Parameters

**Table 141–96 UNLOCK\_PARTITION\_STATS Procedure Parameters**

Parameter	Description
ownname	Name of the schema to unlock
tabname	Name of the table
partname	Partition name

## UNLOCK\_SCHEMA\_STATS Procedure

This procedure unlocks the statistics on all the tables in schema.

### Syntax

```
DBMS_STATS.UNLOCK_SCHEMA_STATS (
    ownname    VARCHAR2);
```

### Parameters

**Table 141–97 UNLOCK\_SCHEMA\_STATS Procedure Parameters**

Parameter	Description
ownname	The name of the schema

### Usage Notes

- When statistics on a table is locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The SET\_\*, DELETE\_\*, IMPORT\_\*, GATHER\_\* procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as GATHER\_SCHEMA\_STATS) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.

## UNLOCK\_TABLE\_STATS Procedure

This procedure unlocks the statistics on the table.

### Syntax

```
DBMS_STATS.UNLOCK_TABLE_STATS (  
    ownname    VARCHAR2,  
    tabname    VARCHAR2);
```

### Parameters

**Table 141–98 UNLOCK\_TABLE\_STATS Procedure Parameters**

Parameter	Description
ownname	The name of the schema
tabname	The name of the table

### Usage Notes

- When statistics on a table is locked, all the statistics depending on the table, including table statistics, column statistics, histograms and statistics on all dependent indexes, are considered to be locked.
- The SET\_\*, DELETE\_\*, IMPORT\_\*, GATHER\_\* procedures that modify statistics in the dictionary of an individual table, index or column will raise an error if statistics of the object is locked.
- Procedures that operates on multiple objects (such as GATHER\_SCHEMA\_STATS) will skip modifying the statistics of an object if it is locked. Many procedures have force argument to override the lock.

## UPGRADE\_STAT\_TABLE Procedure

This procedure upgrades a user statistics table from an older version.

### Syntax

```
DBMS_STATS.UPGRADE_STAT_TABLE (  
    ownname    VARCHAR2,  
    statab     VARCHAR2);
```

### Parameters

**Table 141-99 UPGRADE\_STAT\_TABLE Procedure Parameters**

Parameter	Description
ownname	Name of the schema
statab	Name of the table

### Exceptions

ORA-20000: Unable to upgrade table



---

---

## DBMS\_STORAGE\_MAP

With the `DBMS_STORAGE_MAP` package, you can communicate with the Oracle background process `FMON` to invoke mapping operations that populate mapping views. `FMON` communicates with operating and storage system vendor-supplied mapping libraries.

This chapter contains the following topics:

- [Using DBMS\\_STORAGE\\_MAP](#)
  - Overview
  - Operational Notes
- [Summary of DBMS\\_STORAGE\\_MAP Subprograms](#)

---

## Using DBMS\_STORAGE\_MAP

- [Overview](#)
- [Operational Notes](#)



## Overview

The following terminology and descriptions will help you understand the DBMS\_STORAGE\_MAP API:

- Mapping libraries

Mapping libraries help you map the components of I/O processing stack elements. Examples of I/O processing components include files, logical volumes, and storage array I/O targets. The mapping libraries are identified in `filemap.ora`.

- Mapping files

A mapping file is a mapping structure that describes a file. It provides a set of attributes, including file size, number of extents that the file is composed of, and file type.

- Mapping elements and sub-elements

A mapping element is the abstract mapping structure that describes a storage component within the I/O stack. Examples of elements include mirrors, stripes, partitions, raid5, concatenated elements, and disks—structures that are the mapping building blocks. A mapping sub-element describes the link between an element and the next elements in the I/O mapping stack

- Mapping file extents

A mapping file extent describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides. In the case of a raw device or volume, the file is composed of only one file extent component. A mapping file extent is different from Oracle extents.

**See Also:**

- *Oracle Database Administrator's Guide* for more information
- *Oracle Database Reference* for V\$MAP views, including V\$MAP\_FILE, V\$MAP\_ELEMENT, V\$MAP\_SUBELEMENT, V\$MAP\_FILE\_EXTENT

## Operational Notes

For `MAP_ELEMENT`, `MAP_FILE`, and `MAP_ALL`: Invoking these functions when mapping information already exists will refresh the mapping if configuration IDs are supported. If configuration IDs are not supported, then invoking these functions again will rebuild the mapping.

**See Also:** *Oracle Database Administrator's Guide* for a discussion of the configuration ID, an attribute of the element or file that is changed.

---

## Summary of DBMS\_STORAGE\_MAP Subprograms

**Table 142–1 DBMS\_STORAGE\_MAP Package Subprograms**

Subprogram	Description
<a href="#">DROP_ALL Function</a> on page 142-6	Drops all mapping information in the shared memory of the instance
<a href="#">DROP_ELEMENT Function</a> on page 142-7	Drops the mapping information for the element defined by <code>elemname</code>
<a href="#">DROP_FILE Function</a> on page 142-8	Drops the file mapping information defined by <code>filename</code>
<a href="#">LOCK_MAP Procedure</a> on page 142-9	Locks the mapping information in the shared memory of the instance
<a href="#">MAP_ALL Function</a> on page 142-10	Builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements
<a href="#">MAP_ELEMENT Function</a> on page 142-11	Builds mapping information for the element identified by <code>elemname</code>
<a href="#">MAP_FILE Function</a> on page 142-12	Builds mapping information for the file identified by <code>filename</code>
<a href="#">MAP_OBJECT Function</a> on page 142-13	Builds the mapping information for the Oracle object identified by the object name, owner, and type
<a href="#">RESTORE Function</a> on page 142-14	Loads the entire mapping information from the data dictionary into the shared memory of the instance
<a href="#">SAVE Function</a> on page 142-15	Saves information needed to regenerate the entire mapping into the data dictionary
<a href="#">UNLOCK_MAP Procedure</a> on page 142-16	Unlocks the mapping information in the shared memory of the instance.

## DROP\_ALL Function

This function drops all mapping information in the shared memory of the instance.

### Syntax

```
DBMS_STORAGE_MAP.DROP_ALL(  
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 142–2** *DROP\_ALL Function Parameters*

Parameter	Description
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

## DROP\_ELEMENT Function

This function drops the mapping information for the element defined by `elemname`.

### Syntax

```
DBMS_STORAGE_MAP.DROP_ELEMENT(  
    elemname          IN VARCHAR2,  
    cascade           IN BOOLEAN,  
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 142-3** *DROP\_ELEMENT Function Parameters*

Parameter	Description
<code>elemname</code>	The element for which mapping information is dropped.
<code>cascade</code>	If <code>TRUE</code> , then <code>DROP_ELEMENT</code> is invoked recursively on all elements of the DAG defined by <code>elemname</code> , if possible.
<code>dictionary_update</code>	If <code>TRUE</code> , mapping information in the data dictionary is updated to reflect the changes. The default value is <code>TRUE</code> ; <code>dictionary_update</code> is an overloaded argument.

## DROP\_FILE Function

This function drops the file mapping information defined by `filename`.

### Syntax

```
DBMS_STORAGE_MAP.DROP_FILE(  
    filename          IN VARCHAR2,  
    cascade           IN BOOLEAN,  
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 142–4** *DROP\_FILE Function Parameters*

Parameter	Description
<code>filename</code>	The file for which file mapping information is dropped.
<code>cascade</code>	If <code>TRUE</code> , then the mapping DAGs for the elements where the file resides are also dropped, if possible.
<code>dictionary_update</code>	If <code>TRUE</code> , mapping information in the data dictionary is updated to reflect the changes. The default value is <code>TRUE</code> ; <code>dictionary_update</code> is an overloaded argument.

## LOCK\_MAP Procedure

This procedure locks the mapping information in the shared memory of the instance. This is useful when you need a consistent snapshot of the V\$MAP tables. Without locking the mapping information, V\$MAP\_ELEMENT and V\$MAP\_SUBELEMENT, for example, may be inconsistent.

### Syntax

```
DBMS_STORAGE_MAP.LOCK_MAP;
```

## MAP\_ALL Function

This function builds the entire mapping information for all types of Oracle files (except archive logs), including all directed acyclic graph (DAG) elements. It obtains the latest mapping information because it explicitly synchronizes all mapping libraries.

### Syntax

```
DBMS_STORAGE_MAP.MAP_ALL(  
    max_num_fileext IN NUMBER DEFAULT 100,  
    dictionary_update IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 142–5** MAP\_ALL Function Parameters

Parameter	Description
max_num_fileext	Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is 100; max_num_fileextent is an overloaded argument.
dictionary_update	If TRUE, mapping information in the data dictionary is updated to reflect the changes. The default value is TRUE; dictionary_update is an overloaded argument.

### Usage Notes

You must explicitly call MAP\_ALL in a cold startup scenario.



## MAP\_ELEMENT Function

This function builds mapping information for the element identified by `elemname`. It may not obtain the latest mapping information if the element being mapped, or any one of the elements within its I/O stack (if `cascade` is `TRUE`), is owned by a library that must be explicitly synchronized.

### Syntax

```
DBMS_STORAGE_MAP.MAP_ELEMENT (
  elemname          IN VARCHAR2,
  cascade           IN BOOLEAN,
  dictionary_update IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 142–6** MAP\_ELEMENT Function Parameters

Parameter	Description
<code>elemname</code>	The element for which mapping information is built.
<code>cascade</code>	If <code>TRUE</code> , all elements within the <code>elemname</code> I/O stack DAG are mapped.
<code>dictionary_update</code>	If <code>TRUE</code> , mapping information in the data dictionary is updated to reflect the changes. The default value is <code>TRUE</code> ; <code>dictionary_update</code> is an overloaded argument.

## MAP\_FILE Function

This function builds mapping information for the file identified by `filename`. Use this function if the mapping of one particular file has changed. The Oracle database server does not have to rebuild the entire mapping.

### Syntax

```
DBMS_STORAGE_MAP.MAP_FILE(
  filename          IN VARCHAR2,
  filetype          IN VARCHAR2,
  cascade           IN BOOLEAN,
  max_num_fileextent IN NUMBER DEFAULT 100,
  dictionary_update IN BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 142–7** MAP\_FILE Function Parameters

Parameter	Description
<code>filename</code>	The file for which mapping information is built.
<code>filetype</code>	Defines the type of the file to be mapped. It can be "DATAFILE", "SPFILE", "TEMPFILE", "CONTROLFILE", "LOGFILE", or "ARCHIVEFILE".
<code>cascade</code>	Should be <code>TRUE</code> only if a storage reconfiguration occurred. For all other instances, such as file resizing (either through an <code>ALTER SYSTEM</code> command or DML operations on extended files), <code>cascade</code> can be set to <code>FALSE</code> because the mapping changes are limited to the file extents only.  If <code>TRUE</code> , mapping DAGs are also built for the elements where the file resides.
<code>max_num_fileextent</code>	Defines the maximum number of file extents to be mapped. This limits the amount of memory used when mapping file extents. The default value is 100; <code>max_num_fileextent</code> is an overloaded argument.
<code>dictionary_update</code>	If <code>TRUE</code> , mapping information in the data dictionary is updated to reflect the changes. The default value is <code>TRUE</code> ; <code>dictionary_update</code> is an overloaded argument.

### Usage Notes

This function may not obtain the latest mapping information if the file being mapped, or any one of the elements within its I/O stack (if `cascade` is `TRUE`), is owned by a library that must be explicitly synchronized.

## MAP\_OBJECT Function

This function builds the mapping information for the Oracle object identified by the object name, owner, and type.

### Syntax

```
DBMS_STORAGE_MAP.MAP_OBJECT(  
  objname IN VARCHAR2,  
  owner   IN VARCHAR2,  
  objtype IN VARCHAR2);
```

### Parameters

**Table 142–8** *MAP\_OBJECT Function Parameters*

Parameter	Description
objname	The name of the object.
owner	The owner of the object.
objtype	The type of the object.

## RESTORE Function

This function loads the entire mapping information from the data dictionary into the shared memory of the instance. You can invoke `RESTORE` only after a `SAVE` operation. You must explicitly call `RESTORE` in a warm startup scenario.

### Syntax

```
DBMS_STORAGE_MAP.RESTORE;
```

## SAVE Function

This function saves information needed to regenerate the entire mapping into the data dictionary.

### Syntax

```
DBMS_STORAGE_MAP.SAVE;
```

## UNLOCK\_MAP Procedure

This procedure unlocks the mapping information in the shared memory of the instance.

### Syntax

```
DBMS_STORAGE_MAP.UNLOCK_MAP;
```

The `DBMS_STREAMS` package, one of a set of Oracle Streams packages, provides subprograms to convert `ANYDATA` objects into logical change record (LCR) objects, to return information about Oracle Streams attributes and Oracle Streams clients, and to annotate redo entries generated by a session with a binary tag. This tag affects the behavior of a capture process, a propagation, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_STREAMS Subprograms](#)

---

## Using DBMS\_STREAMS

This section contains topics which relate to using the DBMS\_STREAMS package.

- [Overview](#)
- [Security Model](#)



## Overview

This package provides subprograms to convert ANYDATA objects into logical change record (LCR) objects, to return information about Oracle Streams attributes and Oracle Streams clients, and to annotate redo entries generated by a session with a binary tag. This tag affects the behavior of a capture process, a propagation, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

**See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Oracle Streams

## Security Model

PUBLIC is granted EXECUTE privilege on this package.

**See Also:** *Oracle Database Security Guide* for more information about user group PUBLIC

---

## Summary of DBMS\_STREAMS Subprograms

**Table 143–1 DBMS\_STREAMS Package Subprograms**

Subprogram	Description
<a href="#">COMPATIBLE_11_2 Function</a> on page 143-6	Returns the DBMS_STREAMS.COMPATIBLE_11_2 constant
<a href="#">COMPATIBLE_11_1 Function</a> on page 143-7	Returns the DBMS_STREAMS.COMPATIBLE_11_1 constant
<a href="#">COMPATIBLE_10_2 Function</a> on page 143-8	Returns the DBMS_STREAMS.COMPATIBLE_10_2 constant
<a href="#">COMPATIBLE_10_1 Function</a> on page 143-9	Returns the DBMS_STREAMS.COMPATIBLE_10_1 constant
<a href="#">COMPATIBLE_9_2 Function</a> on page 143-10	Returns the DBMS_STREAMS.COMPATIBLE_9_2 constant
<a href="#">CONVERT_ANYDATA_TO_LCR_DDL Function</a> on page 143-11	Converts a ANYDATA object to a SYS.LCR\$_DDL_RECORD object
<a href="#">CONVERT_ANYDATA_TO_LCR_ROW Function</a> on page 143-12	Converts a ANYDATA object to a SYS.LCR\$_ROW_RECORD object
<a href="#">CONVERT_LCR_TO_XML Function</a> on page 143-13	Converts a logical change record (LCR) encapsulated in a ANYDATA object into an XML object that conforms to the XML schema for LCRs
<a href="#">CONVERT_XML_TO_LCR Function</a> on page 143-14	Converts an XML object that conforms to the XML schema for LCRs into a logical change record (LCR) encapsulated in a ANYDATA object
<a href="#">GET_INFORMATION Function</a> on page 143-15	Returns information about various Oracle Streams attributes
<a href="#">GET_STREAMS_NAME Function</a> on page 143-16	Returns the name of the invoker
<a href="#">GET_STREAMS_TYPE Function</a> on page 143-17	Returns the type of the invoker
<a href="#">GET_TAG Function</a> on page 143-18	Gets the binary tag for all redo entries generated by the current session
<a href="#">MAX_COMPATIBLE Function</a> on page 143-19	Returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database
<a href="#">SET_TAG Procedure</a> on page 143-20	Sets the binary tag for all redo entries subsequently generated by the current session

---

**Note:** The subprograms in this package do not commit.

---

## COMPATIBLE\_11\_2 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_11_2` constant.

### Syntax

```
DBMS_STREAMS.COMPATIBLE_11_2  
RETURN INTEGER;
```

### Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 11.2.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

#### See Also:

- [GET\\_COMPATIBLE Member Function](#) on page 248-38
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

## COMPATIBLE\_11\_1 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_11_1` constant.

### Syntax

```
DBMS_STREAMS.COMPATIBLE_11_1  
RETURN INTEGER;
```

### Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 11.1.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

#### See Also:

- [GET\\_COMPATIBLE Member Function](#) on page 248-38
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

## COMPATIBLE\_10\_2 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_10_2` constant.

### Syntax

```
DBMS_STREAMS.COMPATIBLE_10_2  
RETURN INTEGER;
```

### Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 10.2.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

#### See Also:

- [GET\\_COMPATIBLE Member Function](#) on page 248-38
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

## COMPATIBLE\_10\_1 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_10_1` constant.

### Syntax

```
DBMS_STREAMS.COMPATIBLE_10_1  
RETURN INTEGER;
```

### Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 10.1.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

#### See Also:

- [GET\\_COMPATIBLE Member Function](#) on page 248-38
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

## COMPATIBLE\_9\_2 Function

This function returns the `DBMS_STREAMS.COMPATIBLE_9_2` constant.

### Syntax

```
DBMS_STREAMS.COMPATIBLE_9_2  
RETURN INTEGER;
```

### Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The constant value returned by this function corresponds to 9.2.0 compatibility in a database. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

#### See Also:

- [GET\\_COMPATIBLE Member Function](#) on page 248-38
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter



## CONVERT\_ANYDATA\_TO\_LCR\_DDL Function

This function converts a ANYDATA object into a SYS.LCR\$\_DDL\_RECORD object.

### Syntax

```
DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_DDL(
  source IN ANYDATA)
RETURN SYS.LCR$_DDL_RECORD;
```

### Parameters

**Table 143–2** *CONVERT\_ANYDATA\_TO\_LCR\_DDL Function Parameters*

Parameter	Description
source	The ANYDATA object to be converted. If this object is not a DDL logical change record (DDL LCR), then the function raises an exception.

### Usage Notes

You can use this function in a transformation created by the CREATE\_TRANSFORMATION procedure in the DBMS\_TRANSFORM package. Use the transformation you create when you add a subscriber for propagation of DDL LCRs from a ANYDATA queue to a SYS.LCR\$\_DDL\_RECORD typed queue.

**See Also:** *Oracle Streams Advanced Queuing User's Guide*

## CONVERT\_ANYDATA\_TO\_LCR\_ROW Function

This function converts a ANYDATA object into a SYS.LCR\$\_ROW\_RECORD object.

### Syntax

```
DBMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(
    source IN ANYDATA)
RETURN SYS.LCR$_ROW_RECORD;
```

### Parameters

**Table 143–3 CONVERT\_ANYDATA\_TO\_LCR\_ROW Function Parameters**

Parameter	Description
source	The ANYDATA object to be converted. If this object is not a row logical change record (row LCR), then the function raises an exception.

### Usage Notes

You can use this function in a transformation created by the CREATE\_TRANSFORMATION procedure in the DBMS\_TRANSFORM package. Use the transformation you create when you add a subscriber for propagation of row LCRs from a ANYDATA queue to a SYS.LCR\$\_ROW\_RECORD typed queue.

**See Also:** *Oracle Streams Advanced Queuing User's Guide*

## CONVERT\_LCR\_TO\_XML Function

This function converts a logical change record (LCR) encapsulated in a `ANYDATA` object into an XML object that conforms to the XML schema for LCRs. The LCR can be a row LCR or a DDL LCR.

**See Also:** *Oracle Streams Concepts and Administration* for more information about the XML schema for LCRs

### Syntax

```
DBMS_STREAMS.CONVERT_LCR_TO_XML(  
    anylcr IN ANYDATA)  
RETURN SYS.XMLTYPE;
```

### Parameters

**Table 143–4** *CONVERT\_LCR\_TO\_XML Function Parameters*

Parameter	Description
<code>anylcr</code>	The <code>ANYDATA</code> encapsulated LCR to be converted. If this object is not a <code>ANYDATA</code> encapsulated LCR, then the function raises an exception.

## CONVERT\_XML\_TO\_LCR Function

This function converts an XML object that conforms to the XML schema for logical change records (LCRs) into an LCR encapsulated in a `ANYDATA` object. The LCR can be a row or DDL LCR.

**See Also:** *Oracle Streams Concepts and Administration* for more information about the XML schema for LCRs

### Syntax

```
DBMS_STREAMS.CONVERT_XML_TO_LCR(
    xml_dat IN SYS.XMLTYPE)
RETURN ANYDATA;
```

### Parameters

**Table 143–5** *CONVERT\_XML\_TO\_LCR Function Parameters*

Parameter	Description
<code>xml_dat</code>	The XML LCR object to be converted. If this object does not conform to XML schema for LCRs, then the function raises an exception.

## GET\_INFORMATION Function

This function returns information about various Oracle Streams attributes.

### Syntax

```
DBMS_STREAMS.GET_INFORMATION(  
    name IN VARCHAR2)  
RETURN ANYDATA;
```

### Parameters

**Table 143–6** *GET\_INFORMATION Function Parameters*

Parameter	Description
name	<p>The type of information you want to retrieve. Currently, the following names are available:</p> <ul style="list-style-type: none"> <li>▪ <b>SENDER:</b> Returns the name of the sender for the current logical change record (LCR) from its AQ message properties. This function is called inside a procedure DML handler, a DDL handler, an error handler, or a message handler. Returns <code>NULL</code> if called outside of an apply handler. The return value is to be interpreted as a <code>VARCHAR2</code>.</li> <li>▪ <b>CONSTRAINT_NAME:</b> Returns the name of the constraint that was violated for an LCR that raised an error. This function is called inside a procedure DML handler or error handler for an apply process. Returns <code>NULL</code> if called outside of a procedure DML handler or error handler. The return value is to be interpreted as a <code>VARCHAR2</code>.</li> </ul>

## GET\_STREAMS\_NAME Function

This function gets the Oracle Streams name of the invoker if the invoker is one of the following Oracle Streams types:

- CAPTURE
- APPLY
- ERROR\_EXECUTION

If the invoker is not one of these types, then this function returns a NULL.

### Syntax

```
DBMS_STREAMS.GET_STREAMS_NAME  
RETURN VARCHAR2;
```

### Usage Notes

You can use this function in rule conditions, rule-based transformations, apply handlers, and error handlers. For example, if you use one error handler for multiple apply processes, then you can use the `GET_STREAMS_NAME` function to determine the name of the apply process that raised the error.

## GET\_STREAMS\_TYPE Function

This function gets the Oracle Streams type of the invoker and returns one of the following types:

- CAPTURE
- APPLY
- ERROR\_EXECUTION

If the invoker is not one of these types, then this function returns a NULL.

### Syntax

```
DBMS_STREAMS.GET_STREAMS_TYPE  
RETURN VARCHAR2;
```

### Usage Notes

This function can be used in rule conditions, rule-based transformations, apply handlers, and error handlers. For example, you can use the `GET_STREAMS_TYPE` function to instruct a procedure DML handler to operate differently if it is processing messages from the error queue (`ERROR_EXECUTION` type) instead of the apply process's queue (`APPLY` type).

## GET\_TAG Function

This function gets the binary tag for all redo entries generated by the current session.

---

---

**Note:**

- To execute this function, a user must be granted either EXECUTE\_CATALOG\_ROLE or EXECUTE privilege on the DBMS\_STREAMS\_ADM package.
  - Instead of using the DBMS\_STREAMS.GET\_TAG function, Oracle recommends that you use the DBMS\_STREAMS\_ADM.GET\_TAG function. See [GET\\_TAG Function](#) on page 144-86.
- 
- 

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

### Syntax

```
DBMS_STREAMS.GET_TAG  
RETURN RAW;
```

### Examples

The following example illustrates how to display the current logical change record (LCR) tag as output:

```
SET SERVEROUTPUT ON  
DECLARE  
    raw_tag RAW(2000);  
BEGIN  
    raw_tag := DBMS_STREAMS.GET_TAG();  
    DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));  
END;  
/
```

You can also display the value by querying the DUAL view:

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```



## MAX\_COMPATIBLE Function

This function returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database.

### Syntax

```
DBMS_STREAMS.MAX_COMPATIBLE  
RETURN INTEGER;
```

### Usage Notes

You can use this function with the `GET_COMPATIBLE` member function for logical change records (LCRs) to specify behavior based on compatibility.

The `MAX_COMPATIBLE` function always returns the maximum compatibility for the release of Oracle Database on which it is run. Therefore, when you use this function in rule conditions, the rule conditions do not need to be changed when you upgrade to a later release of Oracle Database.

#### See Also:

- [GET\\_COMPATIBLE Member Function](#) on page 248-38
- *Oracle Streams Concepts and Administration* for information about creating rules that discard changes that are not supported by Oracle Streams
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

## SET\_TAG Procedure

This procedure sets the binary tag for all redo entries subsequently generated by the current session. Each redo entry generated by DML or DDL statements in the current session has this tag. This procedure affects only the current session.

---



---

### Note:

- To execute this procedure, a user must be granted either EXECUTE\_CATALOG\_ROLE or EXECUTE privilege on the DBMS\_STREAMS\_ADM package.
  - Instead of using the DBMS\_STREAMS.SET\_TAG procedure, Oracle recommends that you use the DBMS\_STREAMS\_ADM.SET\_TAG procedure. See [SET\\_TAG Procedure](#) on page 144-171.
- 
- 

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

## Syntax

```
DBMS_STREAMS.SET_TAG (
    tag IN RAW DEFAULT NULL);
```

## Parameters

**Table 143–7 SET\_TAG Procedure Parameters**

Parameter	Description
tag	The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits.  By default, the tag for a session is NULL.  The size limit for a tag value is 2000 bytes.

## Usage Notes

To set the tag to the hexadecimal value of '17' in the current session, run the following procedure:

```
EXEC DBMS_STREAMS.SET_TAG(tag => HEXTORAW('17'));
```

The following are considerations for the SET\_TAG procedure:

- This procedure is not transactional. That is, the effects of SET\_TAG cannot be rolled back.
- If the SET\_TAG procedure is run to set a non-NULL session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the SET\_TAG procedure in a session. A data dictionary build happens when the DBMS\_CAPTURE\_ADM.BUILD procedure is run. The BUILD procedure can be run automatically when a capture process is created.

**See Also:** [BUILD Procedure](#) on page 32-18

---

---

## DBMS\_STREAMS\_ADM

The `DBMS_STREAMS_ADM` package, one of a set of Oracle Streams packages, provides subprograms for configuring Oracle Streams environments. This package also includes subprograms for adding and removing simple rules for capture, propagation, apply, and dequeue at the table, schema, and database level. This package also includes subprograms for configuring and managing XStream outbound servers and inbound servers.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS\\_ADM](#)
  - Overview
  - Deprecated Subprograms
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_STREAMS\\_ADM Subprograms](#)

## Using DBMS\_STREAMS\_ADM

This section contains topics that relate to using the DBMS\_STREAMS\_ADM package.

- [Overview](#)
- [Deprecated Subprograms](#)
- [Security Model](#)
- [Operational Notes](#)

## Overview

The DBMS\_STREAMS\_ADM package, one of a set of Oracle Streams packages, provides subprograms for configuring an Oracle Streams environment. This package also includes subprograms for adding and removing simple rules for capture, propagation, apply, and dequeue at the table, schema, and database level. These rules support logical change records (LCRs), which include row LCRs and data definition language (DDL) LCRs. This package also contains subprograms for creating message rules for specific message types. This package also contains subprograms for creating queues, and for managing Oracle Streams metadata, such as data dictionary information.

If you require more sophisticated rules, then refer to [Chapter 126, "DBMS\\_RULE"](#) package.

### See Also:

- *Oracle Streams Concepts and Administration*, *Oracle Streams Replication Administrator's Guide*, and *Oracle Database 2 Day + Data Replication and Integration Guide* for more information about this package and Oracle Streams
- [Chapter 126, "DBMS\\_RULE"](#)

## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated subprograms. Support for deprecated features is for backward compatibility only.

---

---

The following subprograms are deprecated with Oracle Database 10g Release 2 and later:

- `MAINTAIN_SIMPLE_TABLESPACE`

This procedure is replaced by the `MAINTAIN_SIMPLE_TTS` procedure.

**See Also:** [MAINTAIN\\_SIMPLE\\_TTS Procedure](#) on page 144-111

- `MAINTAIN_TABLESPACES`

This procedure is replaced by the `MAINTAIN_TTS` procedure.

**See Also:** [MAINTAIN\\_TTS Procedure](#) on page 144-126

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

A user is associated with each Oracle Streams client. The following sections describe these users:

- [Oracle Streams Administrator](#)
- [Capture User](#)
- [Propagation User](#)
- [Apply User for an Oracle Streams Apply Process](#)
- [Apply User for an XStream Inbound Server](#)
- [Messaging Client User](#)

---



---

**Note:** The user must be granted additional privileges to perform some administrative tasks using the subprograms in this package, such as creating a synchronous capture. If additional privileges are required for a subprogram, then the privileges are documented in the section that describes the subprogram.

---



---

### Oracle Streams Administrator

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

### Capture User

The following procedures can create a capture process:

- [ADD\\_GLOBAL\\_RULES Procedure](#)
- [ADD\\_SCHEMA\\_RULES Procedure](#)
- [ADD\\_SUBSET\\_RULES Procedure](#)
- [ADD\\_TABLE\\_RULES Procedure](#)

The following procedures can create a synchronous capture:

- [ADD\\_SUBSET\\_RULES Procedure](#)
- [ADD\\_TABLE\\_RULES Procedure](#)

If one of these procedures creates a capture process or a synchronous capture, then it configures the current user as the capture user. The capture user is the user in whose security domain a capture process or synchronous capture captures changes that

satisfy its rule set(s) and runs custom rule-based transformations configured for these rules. This user must have the necessary privileges to capture changes. The procedure grants the capture user `ENQUEUE` privilege on the queue used by the capture process or synchronous capture and configures the user as a secure queue user of the queue.

**See Also:** [CREATE\\_CAPTURE Procedure](#) on page 32-19 and [CREATE\\_SYNC\\_CAPTURE Procedure](#) on page 32-29 for information about the privileges required to capture changes (refer to the `capture_user` parameter)

## Propagation User

The following procedures can create a propagation:

- [ADD\\_GLOBAL\\_PROPAGATION\\_RULES Procedure](#)
- [ADD\\_MESSAGE\\_PROPAGATION\\_RULE Procedure](#)
- [ADD\\_SCHEMA\\_PROPAGATION\\_RULES Procedure](#)
- [ADD\\_SUBSET\\_PROPAGATION\\_RULES Procedure](#)
- [ADD\\_TABLE\\_PROPAGATION\\_RULES Procedure](#)

When a propagation is created, a propagation job also might be created. If a propagation job is created when one of these procedures is run, then the user who runs the procedure owns the propagation job. Each propagation job is an Oracle Scheduler job. You can adjust the schedule of a propagation job using Oracle Scheduler.

---

---

**Note:**

- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users might or might not be the same.
  - For a propagation to work properly, the owner of the source queue must have the necessary privileges to propagate messages.
- 
- 

**See Also:**

- [CREATE\\_PROPAGATION Procedure](#) on page 107-8 for more information about the required privileges
- "Propagation Rules for LCRs" on page 144-11 for information about when a propagation job is created

## Apply User for an Oracle Streams Apply Process

The following procedures can create an apply process:

- [ADD\\_GLOBAL\\_RULES Procedure](#)
- [ADD\\_MESSAGE\\_RULE Procedure](#)
- [ADD\\_SCHEMA\\_RULES Procedure](#)
- [ADD\\_SUBSET\\_RULES Procedure](#)
- [ADD\\_TABLE\\_RULES Procedure](#)



If one of these procedures creates an apply process, then it configures the current user as the apply user. For an apply process, the apply user is the user in whose security domain an apply process dequeues messages that satisfy its rule sets.

An apply user applies messages directly to database objects, runs custom rule-based transformations configured for apply process rules, and runs apply handlers configured for the apply process. This user must have the necessary privileges to apply changes. The procedure grants the apply user `DEQUEUE` privilege on the queue used by the apply process and configures the user as a secure queue user of the queue.

**See Also:** [CREATE\\_APPLY Procedure](#) on page 21-18 for information about the privileges required to apply changes (refer to the `apply_user` parameter)

## Apply User for an XStream Inbound Server

The following procedures can create an XStream inbound server:

- [ADD\\_GLOBAL\\_RULES Procedure](#)
- [ADD\\_SCHEMA\\_RULES Procedure](#)
- [ADD\\_SUBSET\\_RULES Procedure](#)
- [ADD\\_TABLE\\_RULES Procedure](#)

---

---

**Note:** These procedures cannot create an outbound server.

---

---

If the `streams_name` parameter is set to `NULL` and no relevant apply process, inbound server, or outbound server exists, then the procedure creates an apply process automatically with a system-generated name.

The apply process remains an apply process if it receives captured logical change records (LCRs) from a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives captured LCRs from a capture process. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.

If one of these procedures creates an inbound server, then it configures the current user as the apply user. The apply user is the user in whose security domain an XStream client application attaches to an Oracle database.

An apply user applies changes directly to database objects, runs custom rule-based transformations configured for inbound server rules, and runs apply handlers configured for the inbound server. This user must have the necessary privileges to apply changes. The procedure grants the apply user `DEQUEUE` privilege on the queue used by the inbound server and configures the user as a secure queue user.

Each inbound server must have a unique name. The name cannot be used by an apply process, outbound server, or messaging client in the same database, and the name cannot be used by another inbound server in the same database.

If a relevant apply process, inbound server, or outbound server exists, then the procedure does not create an inbound server. Instead, the procedure uses the relevant apply process, inbound server, or outbound server. If the `streams_name` parameter specifies an existing apply process, inbound server, or outbound server, then the specified client is used.

When `streams_name` parameter is `NULL` and the `streams_type` parameter is set to `apply`, the relevant apply process, inbound server, or outbound server is identified in one of the following ways:

- If one existing apply process or outbound server has the source database specified in the `source_database` parameter and uses the queue specified in the `queue_name` parameter, then the procedure uses this apply process or outbound server.
- If the `source_database` parameter is set to `NULL` and one existing apply process, inbound server, or outbound server is using the queue specified in the `queue_name` parameter, then the procedure uses this apply process, inbound server, or outbound server.

If the `streams_name` parameter is set to `NULL` and multiple relevant apply processes, inbound servers, or outbound servers exist, then the procedure raises an error.

---

---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

---

## Messaging Client User

The following procedures can create a messaging client:

- [ADD\\_GLOBAL\\_RULES Procedure](#)
- [ADD\\_MESSAGE\\_RULE Procedure](#)
- [ADD\\_SCHEMA\\_RULES Procedure](#)
- [ADD\\_SUBSET\\_RULES Procedure](#)
- [ADD\\_TABLE\\_RULES Procedure](#)

If one of these procedures creates a messaging client, then the user who runs this procedure is granted the privileges to dequeue from the queue using the messaging client. The procedure configures this user as a secure queue user of the queue, and only this user can use the messaging client.

## Operational Notes

Several procedures in this package create rules for Oracle Streams clients and XStream clients, and several procedures configure an Oracle Streams environment. The following sections provide information about using these procedures:

- [Procedures That Create Rules for Oracle Streams Clients and XStream Clients](#)
- [Procedures That Configure an Oracle Streams Environment](#)

### Procedures That Create Rules for Oracle Streams Clients and XStream Clients

Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients. XStream clients include XStream outbound servers and inbound servers. Some of the procedures in the `DBMS_STREAMS_ADM` package add rules to the rule sets of Oracle Streams clients and XStream clients. The rules can pertain to changes in the redo log, to data manipulation language (DML) changes made to a table, to logical change records (LCRs), or to user messages.

An LCR represents either a row change that results from a DML operation or a data definition language (DDL) change. An LCR that represents a row change is a row LCR, and an LCR that represents a DDL change is a DDL LCR. LCRs can either represent changes that were captured by a capture process or a synchronous capture, or they can represent changes created by a user or application. A user message is a custom message that is based on a user-defined type and created by users or applications.

A capture process, propagation, apply process, messaging client, outbound server, or inbound server can have both positive and negative rule sets. A synchronous capture can have only a positive rule set.

For all of the procedures except the ones that create subset rules, and for all clients except for synchronous captures, you use the `inclusion_rule` parameter to specify the type of rule set (either positive or negative) for the created rules. If the client does not have a rule set of the specified type, then a rule set is created automatically, and the rules are added to the rule set. Other rules in an existing rule set for the client are not affected. Additional rules can be added to a rule set using either the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package. If an client has both a positive and a negative rule set, then the negative rule set is always evaluated first.

The following sections describe each type of rule in detail:

- [Capture Process Rules for Changes in the Redo Log](#)
- [Synchronous Capture Rules for DML Changes to Tables](#)
- [Propagation Rules for LCRs](#)
- [Propagation Rules for User Messages](#)
- [Apply Process Rules for LCRs](#)
- [Apply Process Rules for User Messages](#)
- [Messaging Client Rules for LCRs](#)
- [Messaging Client Rules for User Messages](#)
- [XStream Outbound Server Rules for LCRs](#)
- [XStream Inbound Server Rules for LCRs](#)

---

---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

---

**See Also:** *Oracle Streams Concepts and Administration* for more information about how rules are used in Oracle Streams

### Capture Process Rules for Changes in the Redo Log

The following procedures add rules to a rule set of a capture process when you specify capture for the `streams_type` parameter:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all changes made to a source database. See [ADD\\_GLOBAL\\_RULES Procedure](#) on page 144-34.
- The `ADD_SCHEMA_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for changes made to a specified schema. See [ADD\\_SCHEMA\\_RULES Procedure](#) on page 144-50.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for DML changes made to a subset of rows in a specified table. See [ADD\\_SUBSET\\_RULES Procedure](#) on page 144-59.
- The `ADD_TABLE_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for changes made to a specified table. See [ADD\\_TABLE\\_RULES Procedure](#) on page 144-70.

If one of these procedures adds rules to the positive rule set for a capture process, then the capture process captures row changes resulting from DML changes, or DDL changes, or both from a source database and enqueues these changes into the specified queue. If one of these procedures adds rules to the negative rule set for a capture process, then the capture process discards row changes, or DDL changes, or both from a source database.

A capture process can capture changes locally at a source database or remotely at a downstream database. Therefore, for capture process rules, you should execute the procedure either at the source database or at a downstream database.

If the capture process is a local capture process, or if the capture process is a downstream capture process that uses a database link to the source database, then these procedures automatically prepare the appropriate database objects for instantiation:

- `ADD_GLOBAL_RULES` invokes the `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package at the source database.
- `ADD_SCHEMA_RULES` invokes the `PREPARE_SCHEMA_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package at the source database.
- `ADD_SUBSET_RULES` and `ADD_TABLE_RULES` invoke the `PREPARE_TABLE_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package at the source database.

These procedures also enable supplemental logging for the primary key, unique key, foreign key, and bitmap index columns in the tables prepared for instantiation. The primary key columns are unconditionally logged. The unique key, foreign key, and bitmap index columns are conditionally logged.

If the capture process is a downstream capture process that does not use a database link to the source database, then you must prepare the appropriate objects for

instantiation and specify the necessary supplemental logging manually at the source database.

If one of these procedures is executed at a downstream database, then you specify the source database using the `source_database` parameter, and the specified capture process must exist. The procedure cannot create a capture process if it is run at a downstream database. You can create a capture process at a downstream database using the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

**See Also:** [Chapter , "Summary of DBMS\\_CAPTURE\\_ADM Subprograms"](#) on page 32-5 for more information about the `CREATE_CAPTURE` procedure and the procedures that prepare database objects for instantiation

### Synchronous Capture Rules for DML Changes to Tables

The following procedures add rules to the rule set of a synchronous capture when you specify `sync_capture` for the `streams_type` parameter:

- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to TRUE for DML changes made to a subset of rows in a specified table. See [ADD\\_SUBSET\\_RULES Procedure](#) on page 144-59.
- The `ADD_TABLE_RULES` procedure adds a rule whose rule condition evaluates to TRUE for DML changes made to a specified table. See [ADD\\_TABLE\\_RULES Procedure](#) on page 144-70.

If one of these procedures adds rules to the positive rule set for a synchronous capture, then the synchronous capture captures row changes resulting from DML changes to the table at the source database and enqueues these changes into the specified queue. A synchronous capture cannot have a negative rule set.

A synchronous capture captures changes locally at the database where it is configured. This database is the source database for changes captured by the synchronous capture. Therefore, for synchronous capture rules, you should execute the procedure at the source database.

These procedures automatically prepare the appropriate tables for instantiation by invoking the `PREPARE_SYNC_INSTANTIATION` function in the `DBMS_CAPTURE_ADM` package at the source database.

---



---

#### Note:

- A synchronous capture ignores rules in its rule set that were created by a procedure other than `ADD_SUBSET_RULES` or `ADD_TABLE_RULES`.
  - When the `ADD_TABLE_RULES` or the `ADD_SUBSET_RULES` procedure adds rules to a synchronous capture rule set, the procedure must obtain an exclusive lock on the specified table. If there are outstanding transactions on the specified table, then the procedure waits until it can obtain a lock.
- 
- 

### Propagation Rules for LCRs

The following procedures add propagation rules for LCRs to a rule set of a propagation:

- The `ADD_GLOBAL_PROPAGATION_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all LCRs in a source queue. See [ADD\\_GLOBAL\\_PROPAGATION\\_RULES Procedure](#) on page 144-30.
- The `ADD_SCHEMA_PROPAGATION_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in a source queue containing changes made to a specified schema. See [ADD\\_SCHEMA\\_PROPAGATION\\_RULES Procedure](#) on page 144-46.
- The `ADD_SUBSET_PROPAGATION_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for row LCRs in a source queue containing the results of DML changes made to a subset of rows in a specified table. See ["ADD\\_SUBSET\\_PROPAGATION\\_RULES Procedure"](#) on page 144-55.
- The `ADD_TABLE_PROPAGATION_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in a source queue containing changes made to a specified table. See ["ADD\\_TABLE\\_PROPAGATION\\_RULES Procedure"](#) on page 144-65.

If one of these procedures adds rules to the positive rule set for the propagation, then the rules specify that the propagation propagates LCRs in a source queue to a destination queue. If one of these procedures adds rules to the negative rule set for the propagation, then the rules specify that the propagation discards LCRs in a source queue. When you create rules with one of these procedures, and you specify a value for the `source_database` parameter, then the rules include conditions for the specified source database.

#### **Propagation Rules for User Messages**

The `ADD_MESSAGE_PROPAGATION_RULE` procedure adds a message rule to a rule set of a propagation. If this procedure adds a rule to the positive rule set for the propagation, then the rule specifies that the propagation propagates the user messages of a specific message type that evaluate to `TRUE` for the rule condition from a source queue to a destination queue. If this procedure adds a rule to the negative rule set for the propagation, then the rule specifies that the propagation discards the user messages in a source queue of a specific message type that evaluate to `TRUE` for the rule condition. This procedure generates a rule name for the rule.

**See Also:** ["ADD\\_MESSAGE\\_PROPAGATION\\_RULE Procedure"](#) on page 144-39

#### **Apply Process Rules for LCRs**

The following procedures add rules to a rule set of an apply process when you specify `apply` for the `streams_type` parameter and an apply process for the `streams_name` parameter:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all LCRs in the apply process's queue. See ["ADD\\_GLOBAL\\_RULES Procedure"](#) on page 144-34.
- The `ADD_SCHEMA_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in the apply process's queue containing changes made to a specified schema. See ["ADD\\_SCHEMA\\_RULES Procedure"](#) on page 144-50.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for row LCRs in the apply process's queue containing the results of DML changes made to a subset of rows in a specified table. See ["ADD\\_SUBSET\\_RULES Procedure"](#) on page 144-59.

- The `ADD_TABLE_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in the apply process's queue containing changes made to a specified table. See "[ADD\\_TABLE\\_RULES Procedure](#)" on page 144-70.

If one of these procedures adds rules to the positive rule set for the apply process, then the rules specify that the apply process applies LCRs in its queue. If one of these procedures adds rules to the negative rule set for the apply process, then the rules specify that the apply process discards LCRs in its queue. For apply process rules, you should execute these procedures at the destination database.

Changes applied by an apply process created by one of these procedures generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to alter the tag value after the apply process is created, if necessary.

An apply process can apply captured LCRs from only one source database. If one of these procedures creates an apply process, then specify the source database for the apply process using the `source_database` parameter. If the `source_database` parameter is `NULL`, and one of these procedures creates an apply process, then the source database name of the first LCR received by the apply process is used for the source database.

The rules in the apply process rule sets determine which messages are dequeued by the apply process. When you create rules with one of these procedures, and you specify a value for the `source_database` parameter, then the rules include conditions for the specified source database. If the apply process dequeues an LCR with a source database that is different from the source database for the apply process, then an error is raised. In addition, when adding rules to an existing apply process, the database specified in the `source_database` parameter cannot be different from the source database for the apply process. You can determine the source database for an apply process by querying the `DBA_APPLY_PROGRESS` data dictionary view.

An apply process created by one of these procedures can apply messages only at the local database and can apply only captured messages. To create an apply process that applies messages at a remote database or an apply process that applies user messages, use the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package.

You can also use the `DBMS_APPLY_ADM.CREATE_APPLY` procedure to specify nondefault values for the `apply_captured`, `apply_user`, `apply_database_link`, and `apply_tag` parameters when you run that procedure. You can use one of the procedures in the `DBMS_STREAMS_ADM` package to add rules to a rule set used by the apply process after you create it.

#### See Also:

- "[ALTER\\_APPLY Procedure](#)" on page 21-10
- "[CREATE\\_APPLY Procedure](#)" on page 21-18

#### Apply Process Rules for User Messages

The `ADD_MESSAGE_RULE` procedure adds a message rule to a rule set of an apply process when you specify `apply` for the `streams_type` parameter. For an apply process rule, you should execute this procedure at the destination database.

If this procedure adds a rule to the positive rule set for an apply process, then the apply process dequeues user messages of a specific message type that satisfy the apply process rule and sends these messages to its message handler. If no message handler is specified for the apply process, then use the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to set the message handler. If this procedure adds a rule to the

negative rule set for an apply process, then the apply process discards user messages of a specific message type that satisfy the apply process rule.

**See Also:**

- [ADD\\_MESSAGE\\_RULE Procedure](#) on page 144-43
- [ALTER\\_APPLY Procedure](#) on page 21-10

**Messaging Client Rules for LCRs**

The following procedures add rules to a rule set of a messaging client when you specify `dequeue` for the `streams_type` parameter:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for all LCRs in the messaging client's queue. See "[ADD\\_GLOBAL\\_RULES Procedure](#)" on page 144-34.
- The `ADD_SCHEMA_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in the messaging client's queue containing changes made to a specified schema. See "[ADD\\_SCHEMA\\_RULES Procedure](#)" on page 144-50.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for row LCRs in the messaging client's queue containing the results of DML changes made to a subset of rows in a specified table. See "[ADD\\_SUBSET\\_RULES Procedure](#)" on page 144-59.
- The `ADD_TABLE_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs in the messaging client's queue containing changes made to a specified table. See "[ADD\\_TABLE\\_RULES Procedure](#)" on page 144-70.

If one of these procedures adds rules to the positive rule set for a messaging client, then the messaging client can dequeue persistent row LCRs, or DDL LCRs, or both that originated at the source database that matches the `source_database` parameter. If one of these procedures adds rules to the negative rule set for a messaging client, then the messaging client discards persistent row LCRs, or DDL LCRs, or both that originated at the source database that matches the `source_database` parameter. You should execute these procedures at the database where you want to dequeue the messages with the messaging client.

**Messaging Client Rules for User Messages**

The `ADD_MESSAGE_RULE` procedure adds a message rule to a rule set of a messaging client when you specify `dequeue` for the `streams_type` parameter. You should execute this procedure at the database that will dequeue messages.

If this procedure adds a rule to the positive rule set for a messaging client, then the messaging client dequeues user messages of a specific message type that satisfy the message rule. If this procedure adds a rule to the negative rule set for a messaging client, then the messaging client discards user messages of a specific message type that satisfy the message rule.

**See Also:** "[ADD\\_MESSAGE\\_RULE Procedure](#)" on page 144-43

**XStream Outbound Server Rules for LCRs**

When you specify `apply` for the `streams_type` parameter and an XStream outbound server for the `streams_name` parameter, the following procedures add rules to a rule set of the specified outbound server:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for all LCRs.



- The `ADD_SCHEMA_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for LCRs that contain changes made to a specified schema.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for row LCRs that contain the results of DML changes made to a subset of rows in a specified table.
- The `ADD_TABLE_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for LCRs that contain changes made to a specified table.

These rules are evaluated against LCRs in the outbound server's queue.

If one of the preceding procedures adds rules to the positive rule set for the outbound server, then the rules specify that the outbound server sends LCRs in its queue to the XStream client application. If one of these procedures adds rules to the negative rule set for the outbound server, then the rules specify that the outbound server discards LCRs in its queue. For outbound server rules, execute these procedures at the database to which the XStream client application attaches.

An outbound server can process captured LCRs from only one source database. The source database is the database where the changes originated. If one of these procedures adds rules to the rule set of an outbound server, then specify the source database for the outbound server using the `source_database` parameter.

The rules in the outbound server's rule sets determine which LCRs are dequeued by the outbound server. When you create rules with one of these procedures, and you specify a value for the `source_database` parameter, then the rules include conditions for the specified source database. If the outbound server dequeues an LCR with a source database that is different from the source database for the outbound server, then an error is raised. In addition, when adding rules to an existing outbound server, the database specified in the `source_database` parameter cannot be different from the source database for the outbound server. You can determine the source database for an outbound server by querying the `DBA_XSTREAM_OUTBOUND` data dictionary view.

---



---

**Note:** These procedures cannot create an XStream outbound server. You can use one of the procedures in the `DBMS_STREAMS_ADM` package to add rules to a rule set used by the outbound server after you create it.

---



---

**See Also:** *Oracle Database XStream Guide* for information about creating an outbound server

### XStream Inbound Server Rules for LCRs

When you specify `apply` for the `streams_type` parameter and an XStream inbound server for the `streams_name` parameter, the following procedures add rules to a rule set of the specified inbound server:

- The `ADD_GLOBAL_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for all LCRs sent to the inbound server.
- The `ADD_SCHEMA_RULES` procedure adds rules whose rule conditions evaluate to `TRUE` for LCRs sent to the inbound server that contain changes made to a specified schema.
- The `ADD_SUBSET_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for row LCRs sent to the inbound server that contain the results of data definition language (DML) changes made to a subset of rows in a specified table.

- The `ADD_TABLE_RULES` procedure adds rules whose rule condition evaluates to `TRUE` for LCRs sent to the inbound server that contain changes made to a specified table.

If one of the preceding procedures adds rules to the positive rule set for the inbound server, then the rules specify that the inbound server applies LCRs sent to it by the XStream client application. If one of these procedures adds rules to the negative rule set for the inbound server, then the rules specify that the inbound server discards LCRs sent to it by the XStream client application. For inbound server rules, execute these procedures at the database to which the XStream client application attaches. If an inbound server has no rule sets, then it applies all of the LCRs sent to it by the XStream client application.

Changes applied by an inbound server created by one of these procedures generate tags in the redo log at the destination database with a value of '00' (double zero). You can use the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to alter the tag value after the inbound server is created, if necessary.

The rules in the XStream inbound server rule sets determine which LCRs are either applied or discarded after the LCRs are received from the XStream client application. An inbound server can only process LCRs sent from an XStream client application.

When one of these procedures creates rules for an inbound server, the procedure ignores the `source_database` parameter.

---

---

**Note:** If the name specified in the `streams_name` parameter does not exist, then these procedures always create an apply process. The apply process remains an apply process if it receives captured LCRs from a capture process. The apply process can become an inbound server if an XStream client application attaches to it before it receives LCRs from a capture process. After the initial contact, an apply process cannot be changed into an inbound server, and an inbound server cannot be changed into an apply process.

---

---

**See Also:** *Oracle Database XStream Guide* for information about creating an inbound server

## Procedures That Configure an Oracle Streams Environment

The following procedures in this package configure an environment that is maintained by Oracle Streams:

- [MAINTAIN\\_CHANGE\\_TABLE Procedure](#) configures an Oracle Streams environment that records in a change table the data manipulation language (DML) changes made to a source table. Optionally, this procedure can also configure one-way replication of the table from the source database to the destination database.
- [MAINTAIN\\_GLOBAL Procedure](#) configures an Oracle Streams environment that replicates changes at the database level between two databases.
- [MAINTAIN\\_SCHEMAS Procedure](#) configures an Oracle Streams environment that replicates changes to specified schemas between two databases.
- [MAINTAIN\\_SIMPLE\\_TTS Procedure](#) clones a simple tablespace from a source database at a destination database and configures an Oracle Streams environment that replicates changes to specified tablespace between these two databases.

- [MAINTAIN\\_TABLES Procedure](#) configures an Oracle Streams environment that replicates changes to specified tables between two databases.
- [MAINTAIN\\_TTS Procedure](#) clones a set of tablespaces from a source database at a destination database and configures an Oracle Streams environment that replicates changes to specified tablespaces between these two databases.
- [PRE\\_INSTANTIATION\\_SETUP Procedure](#) and [POST\\_INSTANTIATION\\_SETUP Procedure](#)

The `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures must be used together to complete the Oracle Streams replication configuration. Typically, the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures are used to perform database maintenance operations with little or no down time. See *Oracle Streams Concepts and Administration* for more information.

The following sections contain information about using these procedures:

- [Automatic Platform Conversion](#)
- [Actions Performed by These Procedures](#)
- [Configuration Progress and Recoverability](#)
- [Requirements for Running These Procedures](#)
- [Common Parameters for the Configuration Procedures](#)

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about using these procedures

### Automatic Platform Conversion

If the source and destination databases run on different platforms, then these procedures, or the scripts generated by these procedures, convert transferred datafiles to the appropriate platform automatically.

### Actions Performed by These Procedures

To view all of the actions performed by one of these procedures in detail, use the procedure to generate a script, and view the script in a text editor.

### Configuration Progress and Recoverability

When one of these procedures is run with the `perform_actions` parameter set to `TRUE`, metadata about its configuration actions is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`. If the procedure stops because it encounters an error, then you can use the `RECOVER_OPERATION` procedure to complete the configuration after you correct the conditions that caused the error.

---

**Note:** When one of these procedures is run with the `perform_actions` parameter set to `FALSE`, these views are not populated. Also, the views are not populated when a script generated by one of these procedures is run.

---

**See Also:** "[RECOVER\\_OPERATION Procedure](#)" on page 144-147

### Requirements for Running These Procedures

Meet the following requirements when you use one of these procedures:

- Run the procedure at the capture database. The capture database is the database that will contain the capture process that captures changes made to the source database. If the capture database is the same as the source database, then a local capture process is configured. If the capture database is different from the source database, then a downstream capture process is configured. See *Oracle Streams Replication Administrator's Guide* for more information about the capture database.
- The user who runs the procedure must be able to use a database link from the source database to the destination database. This database link should have the same name as the global name of the destination database.
- If the procedure configures downstream capture, then the corresponding user at the capture database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
- If the procedure configures downstream capture, and the capture database is different from the destination database, then the corresponding user at the capture database must be able to use a database link to access the destination database. This database link should have the same name as the global name of the destination database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the `destination_database` parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- Grant the user who runs the procedure the DBA role. This user must have the necessary privileges to complete the following actions:
  - Create ANYDATA queues, capture processes, propagations, and apply processes.
  - Specify supplemental logging
  - Run subprograms in the `DBMS_STREAMS_ADM` and `DBMS_AQADM` packages.
  - Access the database specified in the `destination_database` parameter through a database link. This database link should have the same name as the global name of the destination database.

Typically, the DBA role can be revoked from the user, if necessary, after the configuration is complete.

- The procedure, or the scripts generated by these procedure, must be run at an Oracle Database 10g Release 2 or later database.
- If the `perform_actions` parameter is set to `TRUE` in the procedure to configure the Oracle Streams environment directly, then all of the databases configured by the procedure must be Oracle Database 10g Release 2 or later databases.
- If the `perform_actions` parameter is set to `FALSE` in the procedure, and the environment is configured with a generated script, then the databases configured by the script must be Oracle Database 10g Release 1 or later databases. If the script configures an Oracle Database 10g Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10g Release 2 or later, such as queue-to-queue propagation.

- Each specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes the procedure must have `READ` and `WRITE` privilege on each one.
- For procedures that include the `bi_directional` parameter, if the `bi_directional` parameter is set to `TRUE`, or if the source database is not the capture database, then the `source_database` parameter must specify a database that contains the database objects to be shared. The database specified in the `destination_database` parameter might or might not contain these database objects. If the destination database does not contain the shared database objects, then the procedure instantiates the database objects at the destination database (excluding the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures).
- For procedures that include the `bi_directional` parameter, if the `bi_directional` parameter is set to `TRUE` or if a network instantiation will be performed, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.

To ensure that the user who runs these procedures has the necessary privileges, you should configure an Oracle Streams administrator at each database, and each database link should be created in the Oracle Streams administrator's schema.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

### Common Parameters for the Configuration Procedures

Table 144–1 describes the common parameters for the procedures in this package that configure an Oracle Streams environment. Some of the procedures do not include all of the parameters in Table 144–1.

**Table 144–1 Common Parameters for Configuration Procedures**

Parameter	Description
<code>perform_actions</code>	<p>If <code>TRUE</code>, then the procedure performs the necessary actions to configure the environment directly.</p> <p>If <code>FALSE</code>, then the procedure does not perform the necessary actions to configure the environment directly.</p> <p>Specify <code>FALSE</code> when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify <code>FALSE</code> and either of the following parameters is <code>NULL</code>:</p> <ul style="list-style-type: none"> <li>■ <code>script_name</code></li> <li>■ <code>script_directory_object</code></li> </ul>

**Table 144–1 (Cont.) Common Parameters for Configuration Procedures**

Parameter	Description
script_name	<p>If non-NULL and the <code>perform_actions</code> parameter is <code>FALSE</code>, then specify the name of the script generated by this procedure. The script contains all of the statements used to configure the environment. If a file with the specified script name exists in the specified directory for the <code>script_directory_object</code> parameter, then the procedure appends the statements to the existing file.</p> <p>If non-NULL and the <code>perform_actions</code> parameter is <code>TRUE</code>, then the procedure generates the specified script and performs the actions to configure the environment directly.</p> <p>If NULL and the <code>perform_actions</code> parameter is <code>TRUE</code>, then the procedure performs the actions to configure the environment directly and does not generate a script.</p> <p>If NULL and the <code>perform_actions</code> parameter is <code>FALSE</code>, then the procedure raises an error.</p>
script_directory_object	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the <code>script_name</code> parameter is NULL, then the procedure ignores this parameter and does not generate a script.</p> <p>If NULL and the <code>script_name</code> parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>
capture_name	<p>The name of each capture process configured to capture changes. Do not specify an owner. If the <code>bi_directional</code> parameter is set to <code>TRUE</code>, then each capture process created by this procedure has the specified name.</p> <p>If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set.</p> <p>If NULL, then the system generates a name for each capture process it creates.</p> <p><b>Note:</b> The capture process name cannot be altered after the capture process is created.</p>
capture_queue_table	<p>The name of the queue table for each queue used by a capture process, specified as [<i>schema_name</i>.] <i>queue_table_name</i>. For example, <code>strmadmin.streams_queue_table</code>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the system generates a name for the queue table of each queue used by a capture process, and the current user is the owner of each queue table.</p>

**Table 144–1 (Cont.) Common Parameters for Configuration Procedures**

Parameter	Description
capture_queue_name	<p>The name of each queue used by a capture process, specified as [<i>schema_name</i>.] <i>queue_name</i>. For example, <i>strmadmin.streams_queue</i>.</p> <p>If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.</p> <p>If NULL, then the system generates a name for each queue used by a capture process.</p>
capture_queue_user	<p>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.</p> <p>If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</p>
propagation_name	<p>The name of each propagation configured to propagate changes. Do not specify an owner.</p> <p>If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the positive propagation rule set.</p> <p>If NULL, then the system generates a name for each propagation it creates.</p> <p><b>Note:</b> The propagation name cannot be altered after the propagation is created.</p>
apply_name	<p>The name of each apply process configured to apply changes. Do not specify an owner.</p> <p>If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the positive apply process rule set.</p> <p>The specified name must not match the name of an existing messaging client at the destination database.</p> <p>If NULL, then the system generates a name for each apply process it creates. When set to NULL, no apply process that applies changes from the source database can exist on the destination database. If an apply process that applies changes from the source database exists at the destination database, then specify a non-NULL value for this parameter.</p> <p><b>Note:</b> The apply process name cannot be altered after the apply process is created.</p>
apply_queue_table	<p>The name of the queue table for each queue used by an apply process, specified as [<i>schema_name</i>.] <i>queue_table_name</i>. For example, <i>strmadmin.streams_queue_table</i>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the system generates a name for the queue table of each queue used by an apply process, and the current user is the owner of each queue table.</p>

**Table 144–1 (Cont.) Common Parameters for Configuration Procedures**

Parameter	Description
apply_queue_name	<p>The name of each queue used by an apply process, specified as [<i>schema_name</i>.] <i>queue_name</i>. For example, <i>strmadmin.streams_queue</i>.</p> <p>If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.</p> <p>If NULL, then the system generates a name for each queue used by an apply process.</p>
apply_queue_user	<p>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.</p> <p>If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</p>
bi_directional	<p>Specify TRUE to configure bi-directional replication between the database specified in <i>source_database</i> and the database specified in <i>destination_database</i>. Both databases are configured as source and destination databases, a capture and apply process is configured to capture changes to both databases, and propagations are configured to propagate these changes. If TRUE, then a database link from the destination database to the source database with the same global name as the source database must exist.</p> <p>Specify FALSE to configure one way replication from the database specified in <i>source_database</i> and the database specified in <i>destination_database</i>. A capture process is configured at the current database and an apply process is configured at the destination database. A propagation is configured if necessary.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for information about when propagations are configured</p>
include_ddl	<p>Specify TRUE to configure an Oracle Streams replication environment that maintains both DML and DDL changes.</p> <p>Specify FALSE to configure an Oracle Streams replication environment that maintains DML changes only. When this parameter is set to FALSE, DDL changes, such as ALTER TABLE, are not replicated.</p>



---

## Summary of DBMS\_STREAMS\_ADM Subprograms

**Table 144–2 DBMS\_STREAMS\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ADD_COLUMN Procedure</a> on page 144-27	Either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule
<a href="#">ADD_GLOBAL_PROPAGATION_RULES Procedure</a> on page 144-30	Either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist
<a href="#">ADD_GLOBAL_RULES Procedure</a> on page 144-34	Adds global rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist
<a href="#">ADD_MESSAGE_PROPAGATION_RULE Procedure</a> on page 144-39	Either adds a message rule to the positive rule set for a propagation, or adds a message rule to the negative rule set for a propagation, and creates the specified propagation if it does not exist
<a href="#">ADD_MESSAGE_RULE Procedure</a> on page 144-43	Adds a message rule to either the positive or negative rule set of an apply process or messaging client, and creates the specified apply process or messaging client if it does not exist
<a href="#">ADD_SCHEMA_PROPAGATION_RULES Procedure</a> on page 144-46	Either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist
<a href="#">ADD_SCHEMA_RULES Procedure</a> on page 144-50	Adds schema rules to either the positive or negative rule set of a capture process, apply process, or messaging client, and creates the specified capture process, apply process, or messaging client if it does not exist
<a href="#">ADD_SUBSET_PROPAGATION_RULES Procedure</a> on page 144-55	Adds subset rules to the positive rule set for a propagation, and creates the specified propagation if it does not exist
<a href="#">ADD_SUBSET_RULES Procedure</a> on page 144-59	Adds subset rules to the positive rule set of a capture process, synchronous capture, apply process, or messaging client, and creates the specified capture process, synchronous capture, apply process, or messaging client if it does not exist
<a href="#">ADD_TABLE_PROPAGATION_RULES Procedure</a> on page 144-65	Either adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist
<a href="#">ADD_TABLE_RULES Procedure</a> on page 144-70	Adds table rules to the rule set of a capture process, synchronous capture, apply process, or messaging client, and creates the specified capture process, synchronous capture, apply process, or messaging client if it does not exist

**Table 144–2 (Cont.) DBMS\_STREAMS\_ADM Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CLEANUP_INSTANTIATION_SETUP Procedure</a> on page 144-76	Removes an Oracle Streams replication configuration that was set up by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in this package
<a href="#">DELETE_COLUMN Procedure</a> on page 144-80	Either adds or removes a declarative rule-based transformation which deletes a column from a row LCR that satisfies the specified rule
<a href="#">GET_MESSAGE_TRACKING Function</a> on page 144-83	Returns the tracking label for the current session
<a href="#">GET_SCN_MAPPING Procedure</a> on page 144-84	Gets information about the system change number (SCN) values to use for Oracle Streams capture and apply processes in an Oracle Streams replication environment
<a href="#">GET_TAG Function</a> on page 144-86	Gets the binary tag for all redo entries generated by the current session
<a href="#">KEEP_COLUMNS Procedure</a> on page 144-87	Either adds or removes a declarative rule-based transformation which keeps a list of columns in a row LCR that satisfies the specified rule
<a href="#">MAINTAIN_CHANGE_TABLE Procedure</a> on page 144-90	Configures an Oracle Streams environment that records in a change table the data manipulation language (DML) changes made to a source table. Optionally, this procedure can also configure one-way replication of the table from the source database to the destination database
<a href="#">MAINTAIN_GLOBAL Procedure</a> on page 144-97	Configures an Oracle Streams environment that replicates changes at the database level between two databases
<a href="#">MAINTAIN_SCHEMAS Procedure</a> on page 144-101	Configures an Oracle Streams environment that replicates changes to specified schemas between two databases
<a href="#">MAINTAIN_SIMPLE_TABLESPACE Procedure</a> on page 144-106	Clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases. This procedure is deprecated.
<a href="#">MAINTAIN_SIMPLE_TTS Procedure</a> on page 144-111	Clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases
<a href="#">MAINTAIN_TABLES Procedure</a> on page 144-114	Configures an Oracle Streams environment that replicates changes to specified tables between two databases
<a href="#">MAINTAIN_TABLESPACES Procedure</a> on page 144-119	Clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases. This procedure is deprecated.
<a href="#">MAINTAIN_TTS Procedure</a> on page 144-126	Clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases
<a href="#">MERGE_STREAMS Procedure</a> on page 144-130	Merges a stream flowing from one capture process with a stream flowing from another capture process

**Table 144–2 (Cont.) DBMS\_STREAMS\_ADM Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">MERGE_STREAMS_JOB Procedure</a> on page 144-133	Determines whether the original capture process and the cloned capture are within the specified merge threshold and, if they are, runs the MERGE_STREAMS procedure to merge the two streams
<a href="#">POST_INSTANTIATION_SETUP Procedure</a> on page 144-136	Performs the actions required after instantiation to configure an Oracle Streams replication environment
<a href="#">PRE_INSTANTIATION_SETUP Procedure</a> on page 144-141	Performs the actions required before instantiation to configure an Oracle Streams replication environment
<a href="#">PURGE_SOURCE_CATALOG Procedure</a> on page 144-146	Removes all Oracle Streams data dictionary information at the local database for the specified object
<a href="#">RECOVER_OPERATION Procedure</a> on page 144-147	Provides options for an Oracle Streams replication configuration operation that stopped because it encountered an error. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.
<a href="#">REMOVE_QUEUE Procedure</a> on page 144-149	Removes the specified ANYDATA queue
<a href="#">REMOVE_RULE Procedure</a> on page 144-150	Removes the specified rule or all rules from the rule set associated with the specified capture process, synchronous capture, propagation, apply process, or messaging client.
<a href="#">REMOVE_STREAMS_CONFIGURATION Procedure</a> on page 144-152	Removes the Oracle Streams configuration at the local database
<a href="#">RENAME_COLUMN Procedure</a> on page 144-154	Either adds or removes a declarative rule-based transformation which renames a column in a row LCR that satisfies the specified rule
<a href="#">RENAME_SCHEMA Procedure</a> on page 144-157	Either adds or removes a declarative rule-based transformation which renames a schema in a row LCR that satisfies the specified rule
<a href="#">RENAME_TABLE Procedure</a> on page 144-159	Either adds or removes a declarative rule-based transformation which renames a table in a row LCR that satisfies the specified rule
<a href="#">SET_MESSAGE_NOTIFICATION Procedure</a> on page 144-161	Sets a notification for messages that can be dequeued by a specified Oracle Streams messaging client from a specified queue
<a href="#">SET_MESSAGE_TRACKING Procedure</a> on page 144-166	Sets the tracking label for logical change records (LCRs) produced by the current session
<a href="#">SET_RULE_TRANSFORM_FUNCTION Procedure</a> on page 144-167	Sets or removes the transformation function name for a rule-based transformation
<a href="#">SET_TAG Procedure</a> on page 144-171	Sets the binary tag for all redo entries subsequently generated by the current session
<a href="#">SET_UP_QUEUE Procedure</a> on page 144-172	Creates a queue table and a queue for use with the capture, propagate, and apply functionality of Oracle Streams

**Table 144–2 (Cont.) DBMS\_STREAMS\_ADM Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SPLIT_STREAMS Procedure</a> on page 144-175	Splits one stream flowing from a capture process off from all of the other streams flowing from the capture process

---

---

**Note:** All subprograms commit unless specified otherwise.

---

---

## ADD\_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which adds a column to a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

This procedure is overloaded. The `column_value` and `column_function` parameters are mutually exclusive.

---



---

### Note:

- `ADD_COLUMN` transformations cannot add columns of the following data types: `BLOB`, `CLOB`, `NCLOB`, `BFILE`, `LONG`, `LONG RAW`, `ROWID`, user-defined types (including object types, `REFs`, `varrays`, nested tables), and Oracle-supplied types (including any types, `XML` types, spatial types, and media types).
  - Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.
- 
- 

**See Also:** *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

## Syntax

```
DBMS_STREAMS_ADM.ADD_COLUMN(
  rule_name      IN  VARCHAR2,
  table_name     IN  VARCHAR2,
  column_name    IN  VARCHAR2,
  column_value   IN  ANYDATA,
  value_type     IN  VARCHAR2      DEFAULT 'NEW',
  step_number    IN  NUMBER        DEFAULT 0,
  operation      IN  VARCHAR2      DEFAULT 'ADD');
```

```
DBMS_STREAMS_ADM.ADD_COLUMN(
  rule_name      IN  VARCHAR2,
  table_name     IN  VARCHAR2,
  column_name    IN  VARCHAR2,
  column_function IN  VARCHAR2,
  value_type     IN  VARCHAR2      DEFAULT 'NEW',
  step_number    IN  NUMBER        DEFAULT 0,
  operation      IN  VARCHAR2      DEFAULT 'ADD');
```

## Parameters

**Table 144–3 ADD\_COLUMN Procedure Parameters**

Parameter	Description
rule_name	The name of the rule, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . If NULL, then the procedure raises an error.  For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.
table_name	The name of the table to which the column is added in the row LCR, specified as [ <i>schema_name.</i> ] <i>object_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.
column_name	The name of the column added to each row LCR that satisfies the rule.
column_value	The value of the added column. Specify the appropriate ANYDATA function for the column data type and the column value. For example, if the data type of the column being added is NUMBER and the value is NULL, then specify the ANYDATA.ConvertNumber(NULL) function.  This parameter cannot be specified if the column_function parameter is specified.
column_function	Either the 'SYSDATE' or the 'SYSTIMESTAMP' SQL function.  The 'SYSDATE' SQL function places the current date and time set for the operating system on which the database resides. The data type of the returned value is DATE, and the format returned depends on the value of the NLS_DATE_FORMAT initialization parameter.  The 'SYSTIMESTAMP' SQL function returns the system date, including fractional seconds and time zone, of the system on which the database resides. The return type is TIMESTAMP WITH TIME ZONE.  The function executes when the rule evaluates to TRUE.  This parameter cannot be specified if the column_value parameter is specified.
value_type	Specify 'NEW' to add the column to the new values in the row LCR. Specify 'OLD' to add the column to the old values in the row LCR.
step_number	The order of execution of the transformation.  <b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.

## Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the add column declarative rule-based transformations for the specified rule are removed that match the specified table\_name, column\_name, and step\_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the ADD\_COLUMN procedures when one or more of these parameters is NULL:

table_name	column_name	step_number	Result
NULL	NULL	NULL	Remove all add column transformations for the specified rule.

<b>table_name</b>	<b>column_name</b>	<b>step_number</b>	<b>Result</b>
NULL	NULL	non-NULL	Remove all add column transformations with the specified <code>step_number</code> for the specified rule.
NULL	non-NULL	non-NULL	Remove all add column transformations with the specified <code>column_name</code> and <code>step_number</code> for the specified rule.
non-NULL	NULL	non-NULL	Remove all add column transformations with the specified <code>table_name</code> and <code>step_number</code> for the specified rule.
NULL	non-NULL	NULL	Remove all add column transformations with the specified <code>column_name</code> for the specified rule.
non-NULL	non-NULL	NULL	Remove all add column transformations with the specified <code>table_name</code> and <code>column_name</code> for the specified rule.
non-NULL	NULL	NULL	Remove all add column transformations with the specified <code>table_name</code> for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all add column transformations with the specified <code>table_name</code> , <code>column_name</code> , and <code>step_number</code> for the specified rule.

## ADD\_GLOBAL\_PROPAGATION\_RULES Procedure

This procedure either adds global rules to the positive rule set for a propagation, or adds global rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

### Syntax

```
DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES (
    streams_name          IN   VARCHAR2  DEFAULT NULL,
    source_queue_name     IN   VARCHAR2,
    destination_queue_name IN  VARCHAR2,
    include_dml           IN   BOOLEAN   DEFAULT TRUE,
    include_ddl           IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr    IN   BOOLEAN   DEFAULT FALSE,
    source_database       IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name         OUT  VARCHAR2,
    ddl_rule_name         OUT  VARCHAR2,
    inclusion_rule        IN   BOOLEAN   DEFAULT TRUE,
    and_condition         IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue        IN   BOOLEAN   DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES (
    streams_name          IN   VARCHAR2  DEFAULT NULL,
    source_queue_name     IN   VARCHAR2,
    destination_queue_name IN  VARCHAR2,
    include_dml           IN   BOOLEAN   DEFAULT TRUE,
    include_ddl           IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr    IN   BOOLEAN   DEFAULT FALSE,
    source_database       IN   VARCHAR2  DEFAULT NULL,
    inclusion_rule        IN   BOOLEAN   DEFAULT TRUE,
    and_condition         IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue        IN   BOOLEAN   DEFAULT NULL);
```

### Parameters

**Table 144–4 ADD\_GLOBAL\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
streams_name	<p>The name of the propagation. Do not specify an owner.</p> <p>If the specified propagation does not exist, then the procedure creates it automatically.</p> <p>If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation.</p> <p>If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</p>



**Table 144–4 (Cont.) ADD\_GLOBAL\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
source_queue_name	<p>The name of the source queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the source queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a source queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p>
destination_queue_name	<p>The name of the destination queue, including a database link, specified as [<i>schema_name</i>.] <i>queue_name</i>[@<i>dblink_name</i>], if the destination queue is in a remote database. The queue must be ANYDATA type.</p> <p>For example, to specify a destination queue named <i>streams_queue</i> in the <i>strmadmin</i> schema and use a database link named <i>db2.net</i>, enter <i>strmadmin.streams_queue@db2.net</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p> <p>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</p> <p><b>Note:</b> Connection qualifiers are not allowed.</p>
include_dml	<p>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</p>
include_ddl	<p>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</p>
include_tagged_lcr	<p>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR.</p> <p>If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag.</p> <p>Usually, specify TRUE for this parameter if the <i>inclusion_rule</i> parameter is set to FALSE.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>

**Table 144–4 (Cont.) ADD\_GLOBAL\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database. The source database is where the changes originated. If <code>NULL</code>, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</p> <p>Oracle recommends that you specify a source database for propagation rules.</p>
dml_rule_name	<p>If <code>include_dml</code> is <code>TRUE</code>, then this parameter contains the DML rule name.</p> <p>If <code>include_dml</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
ddl_rule_name	<p>If <code>include_ddl</code> is <code>TRUE</code>, then this parameter contains the DDL rule name.</p> <p>If <code>include_ddl</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
inclusion_rule	<p>If <code>inclusion_rule</code> is <code>TRUE</code>, then the procedure adds the rules to the positive rule set for the propagation.</p> <p>If <code>inclusion_rule</code> is <code>FALSE</code>, then the procedure adds the rules to the negative rule set for the propagation.</p> <p>In either case, the system creates the rule set if it does not exist.</p>
and_condition	<p>If non-<code>NULL</code>, appends the specified condition to the system-generated rule condition using an <code>AND</code> clause in the following way:</p> <pre>(system_condition) AND (and_condition)</pre> <p>The <code>:lcr</code> in the specified condition must be <code>:dml</code> or <code>:ddl</code>. For example, to specify that the global rules generated by the procedure evaluate to <code>TRUE</code> only if the Oracle Streams tag is the hexadecimal equivalent of <code>'02'</code>, specify the following condition:</p> <pre>:lcr.get_tag() = HEXTORAW('02')</pre> <p>The <code>:lcr</code> in the specified condition is converted to <code>:dml</code> or <code>:ddl</code>, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure the procedure only generates the appropriate rule.</p> <p>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify <code>TRUE</code> for the <code>include_dml</code> parameter and <code>FALSE</code> for the <code>include_ddl</code> parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify <code>FALSE</code> for the <code>include_dml</code> parameter and <code>TRUE</code> for the <code>include_ddl</code> parameter.</p> <p><b>See Also:</b> <a href="#">Chapter 248, "Logical Change Record TYPES"</a></p>

**Table 144-4 (Cont.) ADD\_GLOBAL\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
queue_to_queue	<p>If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.</p> <p>If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.</p> <p>The procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:</p> <ul style="list-style-type: none"> <li>■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.</li> <li>■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.</li> <li>■ If NULL, then the procedure does not change the queue to queue property of the propagation.</li> </ul> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about queue-to-queue propagations</p>

## Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A propagation uses the rules for filtering.

### See Also:

- ["Operational Notes"](#) on page 144-9 and ["Propagation Rules for LCRs"](#) on page 144-11 for more information about the rules created by this procedure
- ["Propagation User"](#) on page 144-6

## Examples

The following is an example of a global rule condition created for DML changes:

```
(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

## ADD\_GLOBAL\_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `capture`, this procedure adds capture process rules for capturing changes to an entire database. See ["Capture Process Rules for Changes in the Redo Log"](#) on page 144-10 for more information about these rules.
- When the `streams_type` parameter is set to `apply` and the `streams_name` parameter specifies the name of an apply process, this procedure adds apply process rules for applying all logical change records (LCRs) in a queue. The rules can specify that the LCRs must be from a particular source database. See ["Apply Process Rules for LCRs"](#) on page 144-12 for more information about these rules.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds messaging client rules for dequeuing all persistent LCRs from a queue. The rules can specify that the LCRs must be from a particular source database. See ["Messaging Client Rules for LCRs"](#) on page 144-14 for more information about these rules.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

---



---

**Caution:** If you add global rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects that are not supported by Oracle Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

---



---



---



---

**Note:** Currently, messaging clients cannot dequeue buffered messages.

---



---

### Syntax

```
DBMS_STREAMS_ADM.ADD_GLOBAL_RULES (
    streams_type      IN   VARCHAR2,
    streams_name      IN   VARCHAR2 DEFAULT NULL,
    queue_name        IN   VARCHAR2 DEFAULT 'streams_queue',
    include_dml       IN   BOOLEAN  DEFAULT TRUE,
    include_ddl       IN   BOOLEAN  DEFAULT FALSE,
    include_tagged_lcr IN   BOOLEAN  DEFAULT FALSE,
    source_database   IN   VARCHAR2 DEFAULT NULL,
    dml_rule_name     OUT  VARCHAR2,
    ddl_rule_name     OUT  VARCHAR2,
    inclusion_rule    IN   BOOLEAN  DEFAULT TRUE,
    and_condition     IN   VARCHAR2 DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_GLOBAL_RULES (
    streams_type      IN   VARCHAR2,
```

```

streams_name      IN  VARCHAR2  DEFAULT NULL,
queue_name        IN  VARCHAR2  DEFAULT 'streams_queue',
include_dml       IN  BOOLEAN   DEFAULT TRUE,
include_ddl       IN  BOOLEAN   DEFAULT FALSE,
include_tagged_lcr IN  BOOLEAN   DEFAULT FALSE,
source_database   IN  VARCHAR2  DEFAULT NULL,
inclusion_rule     IN  BOOLEAN   DEFAULT TRUE,
and_condition     IN  VARCHAR2  DEFAULT NULL);

```

## Parameters

**Table 144–5 ADD\_GLOBAL\_RULES Procedure Parameters**

Parameter	Description
streams_type	<p>The type of Oracle Streams client:</p> <ul style="list-style-type: none"> <li>Specify <code>capture</code> for a capture process.</li> <li>Specify <code>apply</code> for an apply process.</li> <li>Specify <code>dequeue</code> for a messaging client.</li> </ul>
streams_name	<p>The name of the capture process, apply process, or messaging client. Do not specify an owner.</p> <p>If NULL, if <code>streams_type</code> is <code>capture</code> or <code>dequeue</code>, and if one relevant capture process or messaging client for the queue exists, then the relevant Oracle Streams client is used. If no relevant Oracle Streams client exists for the queue, then an Oracle Streams client is created automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified <code>streams_type</code> for the queue exist, then the procedure raises an error.</p> <p>If NULL, if <code>streams_type</code> is <code>apply</code>, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:</p> <ul style="list-style-type: none"> <li>If one existing apply process has the source database specified in <code>source_database</code> and uses the queue specified in <code>queue_name</code>, then the procedure uses this apply process.</li> <li>If <code>source_database</code> is NULL and one existing apply process is using the queue specified in <code>queue_name</code>, then the procedure uses this apply process.</li> </ul> <p>If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.</p> <p>If NULL and multiple relevant apply processes exist, then the procedure raises an error.</p> <p>Each apply process and messaging client must have a unique name.</p>
queue_name	<p>The name of the local queue, specified as <code>[schema_name.]queue_name</code>. The current database must contain the queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.</p> <p>For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.</p>

**Table 144–5 (Cont.) ADD\_GLOBAL\_RULES Procedure Parameters**

Parameter	Description
<code>include_dml</code>	If <code>TRUE</code> , then the procedure creates a rule for DML changes. If <code>FALSE</code> , then the procedure does not create a DML rule. <code>NULL</code> is not permitted.
<code>include_ddl</code>	If <code>TRUE</code> , then the procedure creates a rule for DDL changes. If <code>FALSE</code> , then the procedure does not create a DDL rule. <code>NULL</code> is not permitted.
<code>include_tagged_lcr</code>	<p>If <code>TRUE</code>, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to <code>TRUE</code> regardless of whether a redo entry or LCR has a non-<code>NULL</code> tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-<code>NULL</code> tag. If the rules are added to a positive rule set, then setting this parameter to <code>TRUE</code> is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag.</p> <p>If <code>FALSE</code>, then the procedure adds a condition to each generated rule that causes the rule to evaluate to <code>TRUE</code> only if a redo entry or LCR has a <code>NULL</code> Oracle Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a <code>NULL</code> tag. If the rules are added to a positive rule set, then setting this parameter to <code>FALSE</code> might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a <code>NULL</code> tag.</p> <p>Usually, specify <code>TRUE</code> for this parameter if the <code>inclusion_rule</code> parameter is set to <code>FALSE</code>.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>
<code>source_database</code>	<p>The global name of the source database. If <code>NULL</code>, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>For capture process rules, specify <code>NULL</code> or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</p> <p>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database.</p> <p>For messaging client rules, specify <code>NULL</code> if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</p>

**Table 144-5 (Cont.) ADD\_GLOBAL\_RULES Procedure Parameters**

Parameter	Description
dml_rule_name	<p>If <code>include_dml</code> is <code>TRUE</code>, then this parameter contains the DML rule name.</p> <p>If <code>include_dml</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
ddl_rule_name	<p>If <code>include_ddl</code> is <code>TRUE</code>, then this parameter contains the DDL rule name.</p> <p>If <code>include_ddl</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
inclusion_rule	<p>If <code>inclusion_rule</code> is <code>TRUE</code>, then the procedure adds the rules to the positive rule set for the Oracle Streams client.</p> <p>If <code>inclusion_rule</code> is <code>FALSE</code>, then the procedure adds the rules to the negative rule set for the Oracle Streams client.</p> <p>In either case, the system creates the rule set if it does not exist.</p>
and_condition	<p>If non-<code>NULL</code>, appends the specified condition to the system-generated rule condition using an <code>AND</code> clause in the following way:</p> <pre>(system_condition) AND (and_condition)</pre> <p>The variable in the specified condition must be <code>:lcr</code>. For example, to specify that the global rules generated by the procedure evaluate to <code>TRUE</code> only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</p> <pre>:lcr.get_tag() = HEXTORAW('02')</pre> <p>The <code>:lcr</code> in the specified condition is converted to <code>:dml</code> or <code>:ddl</code>, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</p> <p>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify <code>TRUE</code> for the <code>include_dml</code> parameter and <code>FALSE</code> for the <code>include_ddl</code> parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify <code>FALSE</code> for the <code>include_dml</code> parameter and <code>TRUE</code> for the <code>include_ddl</code> parameter.</p> <p><b>See Also:</b> <a href="#">Chapter 248, "Logical Change Record TYPES"</a></p>

## Usage Notes

This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the database name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the database name plus the sequence number is too long, then the database name is truncated. A capture process, apply process, or messaging client uses the rules for filtering.

### See Also:

- ["Operational Notes"](#) on page 144-9
- ["Security Model"](#) on page 144-5

## Examples

The following is an example of a global rule condition created for DML changes:

```
(:dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```



## ADD\_MESSAGE\_PROPAGATION\_RULE Procedure

This procedure adds a message rule to the positive rule set for a propagation, or adds a message rule to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains the OUT parameter `rule_name`, and the other does not.

### Syntax

```
DBMS_STREAMS_ADM.ADD_MESSAGE_PROPAGATION_RULE (
  message_type          IN   VARCHAR2,
  rule_condition        IN   VARCHAR2,
  streams_name         IN   VARCHAR2  DEFAULT NULL,
  source_queue_name    IN   VARCHAR2,
  destination_queue_name IN  VARCHAR2,
  inclusion_rule       IN   BOOLEAN   DEFAULT TRUE,
  rule_name            OUT  VARCHAR2,
  queue_to_queue       IN   BOOLEAN   DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_MESSAGE_PROPAGATION_RULE (
  message_type          IN   VARCHAR2,
  rule_condition        IN   VARCHAR2,
  streams_name         IN   VARCHAR2  DEFAULT NULL,
  source_queue_name    IN   VARCHAR2,
  destination_queue_name IN  VARCHAR2,
  inclusion_rule       IN   BOOLEAN   DEFAULT TRUE,
  queue_to_queue       IN   BOOLEAN   DEFAULT NULL);
```

### Parameters

**Table 144–6 ADD\_MESSAGE\_PROPAGATION\_RULE Procedure Parameters**

Parameter	Description
<code>message_type</code>	<p>The type of the message. The type can be an Oracle built-in type, such as <code>VARCHAR2</code> or <code>NUMBER</code>, or it can be a user-defined type.</p> <p>If the type is not an Oracle built-in type, then it is specified as <code>[schema_name.] type_name</code>. If the schema is not specified, then the current user is the default.</p> <p>For example, to specify <code>VARCHAR2</code>, enter <code>VARCHAR2 (n)</code>, where <code>n</code> is the size specification. To specify a type named <code>usr_msg</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.usr_msg</code> for this parameter.</p> <p>The following data types require a size specification: <code>VARCHAR2</code>, <code>NVARCHAR2</code>, and <code>RAW</code>.</p> <p><b>See Also:</b> <i>Oracle Database SQL Language Reference</i> for more information about data types</p>
<code>rule_condition</code>	<p>The rule condition for this message type. The rule variable name specified in the rule condition must be the following:</p> <pre>:msg</pre>

**See Also:** "Examples" on page 144-41

**Table 144–6 (Cont.) ADD\_MESSAGE\_PROPAGATION\_RULE Procedure Parameters**

Parameter	Description
streams_name	<p>The name of the propagation. Do not specify an owner.</p> <p>If the specified propagation does not exist, then the procedure creates it automatically.</p> <p>If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation.</p> <p>If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</p>
source_queue_name	<p>The name of the source queue, specified as [<i>schema_name.</i>] <i>queue_name</i>. The current database must contain the source queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a source queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p>
destination_queue_name	<p>The name of the destination queue, including a database link, specified as [<i>schema_name.</i>] <i>queue_name</i>[<i>@dblink_name</i>], if the destination queue is in a remote database. The queue must be ANYDATA type.</p> <p>For example, to specify a destination queue named <i>streams_queue</i> in the <i>strmadmin</i> schema and use a database link named <i>db2.net</i>, enter <i>strmadmin.streams_queue@db2.net</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p> <p>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</p> <p><b>Note:</b> Connection qualifiers are not allowed.</p>
inclusion_rule	<p>If <i>inclusion_rule</i> is TRUE, then the procedure adds the rule to the positive rule set for the propagation.</p> <p>If <i>inclusion_rule</i> is FALSE, then the procedure adds the rule to the negative rule set for the propagation.</p> <p>In either case, the system creates the rule set if it does not exist.</p>
rule_name	Contains the rule name

**Table 144–6 (Cont.) ADD\_MESSAGE\_PROPAGATION\_RULE Procedure Parameters**

Parameter	Description
queue_to_queue	<p>If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.</p> <p>If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.</p> <p>This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:</p> <ul style="list-style-type: none"> <li>▪ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.</li> <li>▪ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.</li> <li>▪ If NULL, then the procedure does not change the queue to queue property of the propagation.</li> </ul> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about queue-to-queue propagations</p>

## Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

When you use this procedure to create a rule set for a message rule, the new rule set does not have an evaluation context. If no evaluation context exists for the specified message type, then this procedure creates a new evaluation context and associates it with the new rule. The evaluation context also has a system-generated name. If you create new rules that use an existing message type, then the new rules use the existing evaluation context for the message type.

### See Also:

- ["Operational Notes"](#) on page 144-9 and ["Propagation Rules for User Messages"](#) on page 144-12 for more information about the rules created by this procedure
- ["Propagation User"](#) on page 144-6

## Examples

Suppose the message type is `VARCHAR2 (128)`. Given this type, the following rule condition can be specified:

```
':msg = 'HQ'''
```

This rule condition evaluates to TRUE if a user message of type `VARCHAR2 (128)` has HQ for its value.

Suppose the message type is `usr_msg`, and that this type has the following attributes: `source_dbname`, `owner`, `name`, and `message`. Given this type, the following rule condition can be specified:

```
':msg.source_dbname = 'DBS1.NET' AND ' || ':msg.owner = 'HR' AND ' ||  
' :msg.name = 'EMPLOYEES''
```

This rule condition evaluates to TRUE if a user message of type `usr_msg` has `DBS1.NET` for its `source_dbname` attribute, `HR` for its `owner` attribute, and `EMPLOYEES` for its `name` attribute.

---

---

**Note:** The quotation marks in the preceding examples are all single quotation marks.

---

---

## ADD\_MESSAGE\_RULE Procedure

This procedure adds a message rule to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `apply`, this procedure adds an apply process rule for dequeuing user messages of a specific message type from a queue. See ["Apply Process Rules for User Messages"](#) on page 144-13 for more information about such rules.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds a messaging client rule for dequeuing user messages of a specific message type from a queue. See ["Messaging Client Rules for User Messages"](#) on page 144-14 for more information about such rules.

This procedure also creates the specified Oracle Streams client if it does not exist.

This procedure is overloaded. One version of this procedure contains the `OUT` parameter `rule_name`, and the other does not.

---



---

**Note:** Currently, messaging clients cannot dequeue buffered messages.

---



---

### Syntax

```
DBMS_STREAMS_ADM.ADD_MESSAGE_RULE (
  message_type      IN   VARCHAR2,
  rule_condition    IN   VARCHAR2,
  streams_type      IN   VARCHAR2,
  streams_name      IN   VARCHAR2 DEFAULT NULL,
  queue_name        IN   VARCHAR2 DEFAULT 'streams_queue',
  inclusion_rule    IN   BOOLEAN  DEFAULT TRUE,
  rule_name         OUT  VARCHAR2);
```

```
DBMS_STREAMS_ADM.ADD_MESSAGE_RULE (
  message_type      IN   VARCHAR2,
  rule_condition    IN   VARCHAR2,
  streams_type      IN   VARCHAR2,
  streams_name      IN   VARCHAR2 DEFAULT NULL,
  queue_name        IN   VARCHAR2 DEFAULT 'streams_queue',
  inclusion_rule    IN   BOOLEAN  DEFAULT TRUE);
```

## Parameters

**Table 144–7 ADD\_MESSAGE\_RULE Procedure Parameters**

Parameter	Description
message_type	<p>The type of the message. The type can be an Oracle built-in type, such as VARCHAR2 or NUMBER, or it can be a user-defined type.</p> <p>If the type is not an Oracle built-in type, then it is specified as [<i>schema_name</i>.] <i>type_name</i>. If the schema is not specified, then the current user is the default.</p> <p>For example, to specify VARCHAR2, enter VARCHAR2 (<i>n</i>), where <i>n</i> is the size specification. To specify a type named <i>usr_msg</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.usr_msg</i> for this parameter.</p> <p>The following data types require a size specification: VARCHAR2, NVARCHAR2, and RAW.</p> <p><b>See Also:</b> <i>Oracle Database SQL Language Reference</i> for more information about data types</p>
rule_condition	<p>The rule condition for this message type. The rule variable name specified in the rule condition must be the following:</p> <p>:msg</p> <p><b>See Also:</b> "Examples" on page 144-45</p>
streams_type	The type of message consumer, either <i>apply</i> for apply process or <i>dequeue</i> for messaging client
streams_name	<p>The name of the Oracle Streams apply process or messaging client.</p> <p>If the specified <i>streams_type</i> is <i>apply</i>, then specify the name of the apply process. Do not specify an owner. If the specified apply process does not exist, then the procedure creates it automatically with a system-generated name.</p> <p>If the specified <i>streams_type</i> is <i>dequeue</i>, then specify the messaging client. For example, if the user <i>strmadmin</i> is the messaging client, then specify <i>strmadmin</i>.</p> <p>If NULL and a relevant apply process or messaging client for the queue exists, then the procedure uses the relevant apply process or messaging client. If NULL and multiple relevant apply processes or messaging clients for the queue exist, then the procedure raises an error.</p> <p>If NULL and no Oracle Streams client of the specified <i>streams_type</i> exists for the queue, then the procedure creates an apply process or messaging client automatically with a system-generated name.</p> <p>An apply process and a messaging client cannot have the same name.</p>
queue_name	<p>The name of the local queue from which messages will be dequeued, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter. If the schema is not specified, then the current user is the default.</p>
inclusion_rule	<p>If <i>inclusion_rule</i> is TRUE, then the procedure adds the rule to the positive rule set for the apply process or messaging client.</p> <p>If <i>inclusion_rule</i> is FALSE, then the procedure adds the rule to the negative rule set for the apply process or messaging client.</p> <p>In either case, the system creates the rule set if it does not exist.</p>
rule_name	Contains the rule name

## Usage Notes

If an apply process rule is added, then this procedure creates the apply process if it does not exist. An apply process created by this procedure can apply only user messages, and dequeued messages are sent to the message handler for the apply process. If a messaging client rule is added, then this procedure creates a messaging client if it does not exist.

When you use this procedure to create a rule set for a message rule, the new rule set does not have an evaluation context. If no evaluation context exists for the specified message type, then this procedure creates a new evaluation context and associates it with the new rule. The evaluation context also has a system-generated name. If you create new rules that use an existing message type, then the new rules use the existing evaluation context for the message type.

### See Also:

- ["Operational Notes"](#) on page 144-9
- ["Security Model"](#) on page 144-5
- [ALTER\\_APPLY Procedure](#) on page 21-10 for more information about setting a message handler for an apply process

## Examples

Suppose the message type is `VARCHAR2 (128)`. Given this type, the following rule condition can be specified:

```
':msg = 'HQ'''
```

This rule condition evaluates to `TRUE` if a user message of type `VARCHAR2 (128)` has `HQ` for its value.

Suppose the message type is `usr_msg`, and that this type has the following attributes: `source_dbname`, `owner`, `name`, and `message`. Given this type, the following rule condition can be specified:

```
':msg.source_dbname = 'DBS1.NET'' AND ' || ':msg.owner = 'HR'' AND ' || ':msg.name = 'EMPLOYEES'''
```

This rule condition evaluates to `TRUE` if a user message of type `usr_msg` has `DBS1.NET` for its `source_dbname` attribute, `HR` for its `owner` attribute, and `EMPLOYEES` for its `name` attribute.

---



---

**Note:** The quotation marks in the preceding examples are all single quotation marks.

---



---

## ADD\_SCHEMA\_PROPAGATION\_RULES Procedure

This procedure either adds schema rules to the positive rule set for a propagation, or adds schema rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

### Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES (
  schema_name          IN   VARCHAR2,
  streams_name         IN   VARCHAR2  DEFAULT NULL,
  source_queue_name    IN   VARCHAR2,
  destination_queue_name IN  VARCHAR2,
  include_dml          IN   BOOLEAN  DEFAULT TRUE,
  include_ddl          IN   BOOLEAN  DEFAULT FALSE,
  include_tagged_lcr   IN   BOOLEAN  DEFAULT FALSE,
  source_database      IN   VARCHAR2  DEFAULT NULL,
  dml_rule_name        OUT  VARCHAR2,
  ddl_rule_name        OUT  VARCHAR2,
  inclusion_rule       IN   BOOLEAN  DEFAULT TRUE,
  and_condition        IN   VARCHAR2  DEFAULT NULL,
  queue_to_queue       IN   BOOLEAN  DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES (
  schema_name          IN   VARCHAR2,
  streams_name         IN   VARCHAR2  DEFAULT NULL,
  source_queue_name    IN   VARCHAR2,
  destination_queue_name IN  VARCHAR2,
  include_dml          IN   BOOLEAN  DEFAULT TRUE,
  include_ddl          IN   BOOLEAN  DEFAULT FALSE,
  include_tagged_lcr   IN   BOOLEAN  DEFAULT FALSE,
  source_database      IN   VARCHAR2  DEFAULT NULL,
  inclusion_rule       IN   BOOLEAN  DEFAULT TRUE,
  and_condition        IN   VARCHAR2  DEFAULT NULL,
  queue_to_queue       IN   BOOLEAN  DEFAULT NULL);
```

### Parameters

**Table 144–8 ADD\_SCHEMA\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
schema_name	The name of the schema. For example, hr.
streams_name	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.



**Table 144–8 (Cont.) ADD\_SCHEMA\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
source_queue_name	<p>The name of the source queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the source queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a source queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p>
destination_queue_name	<p>The name of the destination queue, including a database link, specified as [<i>schema_name</i>.] <i>queue_name</i>[<i>@dblink_name</i>], if the destination queue is in a remote database. The queue must be ANYDATA type.</p> <p>For example, to specify a destination queue named <i>streams_queue</i> in the <i>strmadmin</i> schema and use a database link named <i>db2.net</i>, enter <i>strmadmin.streams_queue@db2.net</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p> <p>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</p> <p><b>Note:</b> Connection qualifiers are not allowed.</p>
include_dml	<p>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</p>
include_ddl	<p>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</p>
include_tagged_lcr	<p>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a logical change record (LCR) has a non-NULL tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR.</p> <p>If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if an LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a NULL tag.</p> <p>Usually, specify TRUE for this parameter if the <i>inclusion_rule</i> parameter is set to FALSE.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>

**Table 144–8 (Cont.) ADD\_SCHEMA\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database. The source database is where the change originated. If <code>NULL</code>, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</p> <p>Oracle recommends that you specify a source database for propagation rules.</p>
dml_rule_name	<p>If <code>include_dml</code> is <code>TRUE</code>, then this parameter contains the DML rule name.</p> <p>If <code>include_dml</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
ddl_rule_name	<p>If <code>include_ddl</code> is <code>TRUE</code>, then this parameter contains the DDL rule name.</p> <p>If <code>include_ddl</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
inclusion_rule	<p>If <code>inclusion_rule</code> is <code>TRUE</code>, then the procedure adds the rules to the positive rule set for the propagation.</p> <p>If <code>inclusion_rule</code> is <code>FALSE</code>, then the procedure adds the rules to the negative rule set for the propagation.</p> <p>In either case, the system creates the rule set if it does not exist.</p>
and_condition	<p>If non-<code>NULL</code>, appends the specified condition to the system-generated rule condition using an <code>AND</code> clause in the following way:</p> <pre>(system_condition) AND (and_condition)</pre> <p>The <code>:lcr</code> in the specified condition must be <code>:dml</code> or <code>:ddl</code>. For example, to specify that the schema rules generated by the procedure evaluate to <code>TRUE</code> only if the Oracle Streams tag is the hexadecimal equivalent of <code>'02'</code>, specify the following condition:</p> <pre>:lcr.get_tag() = HEXTORAW('02')</pre> <p>The <code>:lcr</code> in the specified condition is converted to <code>:dml</code> or <code>:ddl</code>, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</p> <p>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify <code>TRUE</code> for the <code>include_dml</code> parameter and <code>FALSE</code> for the <code>include_ddl</code> parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify <code>FALSE</code> for the <code>include_dml</code> parameter and <code>TRUE</code> for the <code>include_ddl</code> parameter.</p> <p><b>See Also:</b> <a href="#">Chapter 248, "Logical Change Record TYPES"</a></p>

**Table 144–8 (Cont.) ADD\_SCHEMA\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
queue_to_queue	<p>If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.</p> <p>If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.</p> <p>This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:</p> <ul style="list-style-type: none"> <li>■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.</li> <li>■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.</li> <li>■ If NULL, then the procedure does not change the queue to queue property of the propagation.</li> </ul> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about queue-to-queue propagations</p>

## Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A propagation uses the rules for filtering.

### See Also:

- ["Operational Notes"](#) on page 144-9 and ["Propagation Rules for LCRs"](#) on page 144-11 for more information about the rules created by this procedure
- ["Propagation User"](#) on page 144-6

## Examples

The following is an example of a schema rule condition created for DML changes:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```

## ADD\_SCHEMA\_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `capture`, this procedure adds capture process rules for capturing changes to a specified schema. See ["Capture Process Rules for Changes in the Redo Log"](#) on page 144-10 for more information about these rules.
- When the `streams_type` parameter is set to `apply` and the `streams_name` parameter specifies the name of an apply process, this procedure adds apply process rules for applying logical change records (LCRs) in a queue that contain changes to a specified schema. The rules can specify that the LCRs must be from a particular source database. See ["Apply Process Rules for LCRs"](#) on page 144-12 for more information about these rules.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds messaging client rules for dequeuing persistent LCRs from a queue that contain changes to a specified schema. The rules can specify that the LCRs must be from a particular source database. See ["Messaging Client Rules for LCRs"](#) on page 144-14 for more information about these rules.

This procedure creates the specified capture process, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

---



---

**Caution:** If you add schema rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects in the schema that are not supported by Oracle Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

---



---



---



---

**Note:** Currently, messaging clients cannot dequeue buffered messages.

---



---

### Syntax

```
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES (
  schema_name          IN   VARCHAR2,
  streams_type         IN   VARCHAR2,
  streams_name        IN   VARCHAR2  DEFAULT NULL,
  queue_name          IN   VARCHAR2  DEFAULT 'streams_queue',
  include_dml         IN   BOOLEAN   DEFAULT TRUE,
  include_ddl         IN   BOOLEAN   DEFAULT FALSE,
  include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
  source_database     IN   VARCHAR2  DEFAULT NULL,
  dml_rule_name       OUT  VARCHAR2,
  ddl_rule_name       OUT  VARCHAR2,
  inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
  and_condition       IN   VARCHAR2  DEFAULT NULL);
```

```

DBMS_STREAMS_ADM.ADD_SCHEMA_RULES (
  schema_name      IN   VARCHAR2,
  streams_type     IN   VARCHAR2,
  streams_name     IN   VARCHAR2  DEFAULT NULL,
  queue_name      IN   VARCHAR2  DEFAULT 'streams_queue',
  include_dml     IN   BOOLEAN   DEFAULT TRUE,
  include_ddl     IN   BOOLEAN   DEFAULT FALSE,
  include_tagged_lcr IN  BOOLEAN   DEFAULT FALSE,
  source_database IN   VARCHAR2  DEFAULT NULL,
  inclusion_rule  IN   BOOLEAN   DEFAULT TRUE,
  and_condition   IN   VARCHAR2  DEFAULT NULL);

```

## Parameters

**Table 144–9 ADD\_SCHEMA\_RULES Procedure Parameters**

Parameter	Description
schema_name	<p>The name of the schema. For example, hr.</p> <p>You can specify a schema that does not yet exist, because Oracle Streams does not validate the existence of the schema.</p>
streams_type	<p>The type of Oracle Streams client:</p> <ul style="list-style-type: none"> <li>▪ Specify <code>capture</code> for a capture process.</li> <li>▪ Specify <code>apply</code> for an apply process.</li> <li>▪ Specify <code>dequeue</code> for a messaging client.</li> </ul>
streams_name	<p>The name of the capture process, apply process, or messaging client. Do not specify an owner.</p> <p>If NULL, if <code>streams_type</code> is <code>capture</code> or <code>dequeue</code>, and if one relevant capture process or messaging client for the queue exists, then the relevant Oracle Streams client is used. If no relevant Oracle Streams client exists for the queue, then an Oracle Streams client is created automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified <code>streams_type</code> for the queue exist, then the procedure raises an error.</p> <p>If NULL, if <code>streams_type</code> is <code>apply</code>, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:</p> <ul style="list-style-type: none"> <li>▪ If one existing apply process has the source database specified in <code>source_database</code> and uses the queue specified in <code>queue_name</code>, then the procedure uses this apply process.</li> <li>▪ If <code>source_database</code> is NULL and one existing apply process is using the queue specified in <code>queue_name</code>, then the procedure uses this apply process.</li> </ul> <p>If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.</p> <p>If NULL and multiple relevant apply processes exist, then the procedure raises an error.</p> <p>Each apply process and messaging client must have a unique name.</p>

**Table 144–9 (Cont.) ADD\_SCHEMA\_RULES Procedure Parameters**

Parameter	Description
queue_name	<p>The name of the local queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.</p> <p>For capture process rules, this is the queue into which a capture process enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.</p>
include_dml	<p>If TRUE, then the procedure creates a rule for DML changes. If FALSE, then the procedure does not create a DML rule. NULL is not permitted.</p>
include_ddl	<p>If TRUE, then the procedure creates a rule for DDL changes. If FALSE, then the procedure does not create a DDL rule. NULL is not permitted.</p>
include_tagged_lcr	<p>If TRUE, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to TRUE regardless of whether a redo entry or LCR has a non-NULL tag. If the rules are added to the positive rule set for the process, then a redo entry is always considered for capture, and an LCR is always considered for apply, regardless of whether the redo entry or LCR has a non-NULL tag. If the rules are added to a positive rule set, then setting this parameter to TRUE is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the process, then whether a redo entry or LCR is discarded does not depend on the tag.</p> <p>If FALSE, then the procedure adds a condition to each generated rule that causes the rule to evaluate to TRUE only if a redo entry or LCR has a NULL Oracle Streams tag. If the rules are added to the positive rule set for the process, then a redo entry is considered for capture, and an LCR is considered for apply, only when the redo entry or LCR contains a NULL tag. If the rules are added to a positive rule set, then setting this parameter to FALSE might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the process, then a redo entry or LCR can be discarded only if it has a NULL tag.</p> <p>Usually, specify TRUE for this parameter if the <code>inclusion_rule</code> parameter is set to FALSE.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>

**Table 144–9 (Cont.) ADD\_SCHEMA\_RULES Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database. If <code>NULL</code>, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>For capture process rules, specify <code>NULL</code> or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</p> <p>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database.</p> <p>For messaging client rules, specify <code>NULL</code> if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</p>
dml_rule_name	<p>If <code>include_dml</code> is <code>TRUE</code>, then this parameter contains the DML rule name.</p> <p>If <code>include_dml</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
ddl_rule_name	<p>If <code>include_ddl</code> is <code>TRUE</code>, then this parameter contains the DDL rule name.</p> <p>If <code>include_ddl</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
inclusion_rule	<p>If <code>inclusion_rule</code> is <code>TRUE</code>, then the procedure adds the rules to the positive rule set for the Oracle Streams client.</p> <p>If <code>inclusion_rule</code> is <code>FALSE</code>, then the procedure adds the rules to the negative rule set for the Oracle Streams client.</p> <p>In either case, the system creates the rule set if it does not exist.</p>

**Table 144–9 (Cont.) ADD\_SCHEMA\_RULES Procedure Parameters**

Parameter	Description
<code>and_condition</code>	<p>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</p> <pre>(system_condition) AND (and_condition)</pre> <p>The variable in the specified condition must be <code>:lcr</code>. For example, to specify that the schema rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</p> <pre>:lcr.get_tag() = HEXTORAW('02')</pre> <p>The <code>:lcr</code> in the specified condition is converted to <code>:dml</code> or <code>:ddl</code>, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</p> <p>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the <code>include_dml</code> parameter and FALSE for the <code>include_ddl</code> parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the <code>include_dml</code> parameter and TRUE for the <code>include_ddl</code> parameter.</p> <p><b>See Also:</b> <a href="#">Chapter 248, "Logical Change Record TYPES"</a></p>

## Usage Notes

This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the schema name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the schema name plus the sequence number is too long, then the schema name is truncated. A capture process, apply process, or messaging client uses the rules for filtering.

**See Also:**

- ["Operational Notes"](#) on page 144-9
- ["Security Model"](#) on page 144-5

## Examples

The following is an example of a schema rule condition created for DML changes:

```
( (:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```



## ADD\_SUBSET\_PROPAGATION\_RULES Procedure

This procedure adds propagation rules that propagate the logical change records (LCRs) related to a subset of the rows in the specified table in a source queue to a destination queue, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains three OUT parameters, and the other does not.

### Syntax

```
DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES (
  table_name          IN   VARCHAR2,
  dml_condition       IN   VARCHAR2,
  streams_name        IN   VARCHAR2  DEFAULT NULL,
  source_queue_name   IN   VARCHAR2,
  destination_queue_name IN  VARCHAR2,
  include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
  source_database     IN   VARCHAR2  DEFAULT NULL,
  insert_rule_name    OUT  VARCHAR2,
  update_rule_name    OUT  VARCHAR2,
  delete_rule_name    OUT  VARCHAR2,
  queue_to_queue      IN   BOOLEAN  DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES (
  table_name          IN   VARCHAR2,
  dml_condition       IN   VARCHAR2,
  streams_name        IN   VARCHAR2  DEFAULT NULL,
  source_queue_name   IN   VARCHAR2,
  destination_queue_name IN  VARCHAR2,
  include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
  source_database     IN   VARCHAR2  DEFAULT NULL,
  queue_to_queue      IN   BOOLEAN  DEFAULT NULL);
```

### Parameters

**Table 144–10 ADD\_SUBSET\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
table_name	The name of the table specified as [ <i>schema_name.</i> ] <i>object_name</i> . For example, <i>hr.employees</i> . If the schema is not specified, then the current user is the default.  The specified table must exist in the same database as the propagation. Also, the specified table cannot have any LOB, LONG, LONG RAW, or XMLType columns currently or in the future.
dml_condition	The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL.  For example, to specify rows in the <i>hr.employees</i> table where the salary is greater than 4000 and the job_id is SA_MAN, enter the following as the condition:  ' salary > 4000 and job_id = 'SA_MAN' '
	<b>Note:</b> The quotation marks in the preceding example are all single quotation marks.

**Table 144–10 (Cont.) ADD\_SUBSET\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
streams_name	<p>The name of the propagation. Do not specify an owner.</p> <p>If the specified propagation does not exist, then the procedure creates it automatically.</p> <p>If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation.</p> <p>If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.</p>
source_queue_name	<p>The name of the source queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the source queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a source queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p>
destination_queue_name	<p>The name of the destination queue, including a database link, specified as [<i>schema_name</i>.] <i>queue_name</i>[@<i>dblink_name</i>], if the destination queue is in a remote database. The queue must be ANYDATA type.</p> <p>For example, to specify a destination queue named <i>streams_queue</i> in the <i>strmadmin</i> schema and use a database link named <i>db2.net</i>, enter <i>strmadmin.streams_queue@db2.net</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p> <p>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</p> <p><b>Note:</b> Connection qualifiers are not allowed.</p>
include_tagged_lcr	<p>If TRUE, then an LCR is always considered for propagation, regardless of whether it has a non-NULL tag. This setting is appropriate for a full (for example, standby) copy of a database.</p> <p>If FALSE, then an LCR is considered for propagation only when the LCR contains a NULL tag. A setting of FALSE is often specified in update-anywhere configurations to avoid sending a change back to its source database.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>
source_database	<p>The global name of the source database. The source database is where the change originated. If NULL, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <i>DBS1</i> and the domain is <i>.NET</i>, then the procedure specifies <i>DBS1.NET</i> automatically.</p> <p>Oracle recommends that you specify a source database for propagation rules.</p>

**Table 144–10 (Cont.) ADD\_SUBSET\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
insert_rule_name	Contains the system-generated INSERT rule name. This rule handles inserts and updates that must be converted into inserts.
update_rule_name	Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.
delete_rule_name	Contains the system-generated DELETE rule name. This rule handles deletes and updates that must be converted into deletes
queue_to_queue	<p>If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.</p> <p>If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.</p> <p>This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:</p> <ul style="list-style-type: none"> <li>■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.</li> <li>■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.</li> <li>■ If NULL, then the procedure does not change the queue to queue property of the propagation.</li> </ul> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about queue-to-queue propagations</p>

## Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

Running this procedure generates three rules for the specified propagation: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only row LCRs that satisfy the condition specified for the `dml_condition` parameter are propagated. For UPDATE statements, the following variations are possible:

- If both the new and old values in a row LCR satisfy the specified `dml_condition`, then the row LCR is propagated without any changes.
- If neither the new or old values in a row LCR satisfy the specified `dml_condition`, then the row LCR is not propagated.
- If the old values for a row LCR satisfy the specified `dml_condition`, but the new values do not, then the update row LCR is converted into a delete row LCR.
- If the new values for a row LCR satisfy the specified `dml_condition`, but the old values do not, then the update row LCR is converted to an insert row LCR.

When an update is converted into an insert or a delete, it is called row migration.

A propagation uses the rules for filtering. If the propagation does not have a positive rule set, then the procedure creates a positive rule set automatically, and the rules for propagating changes to the table are added to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing positive rule set for the propagation are not affected. Additional rules can be added using either the DBMS\_STREAMS\_ADM package or the DBMS\_RULE\_ADM package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The ADD\_SUBSET\_RULES procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

When you create propagation subset rules for a table, you should create an unconditional supplemental log group at the source database with all the columns in the table. Supplemental logging is required if an update must be converted to an insert. The propagation rule must have all the column values to be able to perform this conversion correctly.

---



---

**Attention:** Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

---



---

**See Also:**

- ["Operational Notes"](#) on page 144-9 and ["Propagation Rules for LCRs"](#) on page 144-11 for more information about the rules created by this procedure
- ["Propagation User"](#) on page 144-6

## Examples

The following is an example of a rule condition created for filtering a row LCR containing an update operation when the `dml_condition` is `region_id = 2`, the `table_name` is `hr.regions`, and the `source_database` is `db1.net`:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','REGION_ID') IS NOT NULL)
AND (:dml.get_value('OLD','REGION_ID') IS NOT NULL)
AND (:dml.get_value('OLD','REGION_ID').AccessNumber()=2)
AND (:dml.get_value('NEW','REGION_ID').AccessNumber()=2)
```

## ADD\_SUBSET\_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `capture`, this procedure adds capture process rules for capturing changes to a subset of rows in a specified table. See ["Capture Process Rules for Changes in the Redo Log"](#) on page 144-10 for more information about these rules.
- When the `streams_type` parameter is set to `sync_capture`, this procedure adds rules for capturing changes to a subset of rows in a specified table. See ["Synchronous Capture Rules for DML Changes to Tables"](#) on page 144-11 for more information about these rules.
- When the `streams_type` parameter is set to `apply` and the `streams_name` parameter specifies the name of an apply process, this procedure adds apply process rules for applying logical change records (LCRs) in a queue that contain changes to a subset of rows in a specified table. The rules can specify that the LCRs must be from a particular source database. See ["Apply Process Rules for LCRs"](#) on page 144-12 for more information about these rules.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds messaging client rules for dequeuing persistent LCRs from a queue that contain changes to a subset of rows in a specified table. The rules can specify that the LCRs must be from a particular source database. See ["Messaging Client Rules for LCRs"](#) on page 144-14 for more information about these rules.

This procedure creates the specified capture process, synchronous capture, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains three `OUT` parameters, and the other does not.

---



---

### Note:

- Currently, messaging clients cannot dequeue buffered messages.
  - The invoking user must be granted the `DBA` role to create a synchronous capture.
- 
- 

## Syntax

```
DBMS_STREAMS_ADM.ADD_SUBSET_RULES (
  table_name          IN   VARCHAR2,
  dml_condition       IN   VARCHAR2,
  streams_type        IN   VARCHAR2 DEFAULT 'apply',
  streams_name        IN   VARCHAR2 DEFAULT NULL,
  queue_name          IN   VARCHAR2 DEFAULT 'streams_queue',
  include_tagged_lcr  IN   BOOLEAN  DEFAULT FALSE,
  source_database     IN   VARCHAR2 DEFAULT NULL,
  insert_rule_name    OUT  VARCHAR2,
  update_rule_name    OUT  VARCHAR2,
  delete_rule_name    OUT  VARCHAR2);
```

```
DBMS_STREAMS_ADM.ADD_SUBSET_RULES (
  table_name          IN   VARCHAR2,
  dml_condition       IN   VARCHAR2,
  streams_type        IN   VARCHAR2 DEFAULT 'apply',
```

```

streams_name      IN  VARCHAR2 DEFAULT NULL,
queue_name        IN  VARCHAR2 DEFAULT 'streams_queue',
include_tagged_lcr IN  BOOLEAN  DEFAULT FALSE,
source_database   IN  VARCHAR2 DEFAULT NULL);

```

## Parameters

**Table 144–11 ADD\_SUBSET\_RULES Procedure Parameters**

Parameter	Description
table_name	<p>The name of the table specified as [<i>schema_name.</i>] <i>object_name</i>. For example, <i>hr.employees</i>. If the schema is not specified, then the current user is the default.</p> <p>The specified table must exist in the same database as the capture process, synchronous capture, apply process, or messaging client. Also, the specified table cannot have any LOB, LONG, LONG RAW, or XMLType columns currently or in the future.</p>
dml_condition	<p>The subset condition. Specify this condition similar to the way you specify conditions in a WHERE clause in SQL.</p> <p>For example, to specify rows in the <i>hr.employees</i> table where the <i>salary</i> is greater than 4000 and the <i>job_id</i> is <i>SA_MAN</i>, enter the following as the condition:</p> <pre>' salary &gt; 4000 and job_id = 'SA_MAN' '</pre> <p><b>Note:</b> The quotation marks in the preceding example are all single quotation marks.</p>
streams_type	<p>The type of Oracle Streams client:</p> <ul style="list-style-type: none"> <li>▪ Specify <i>capture</i> for a capture process.</li> <li>▪ Specify <i>sync_capture</i> for a synchronous capture.</li> <li>▪ Specify <i>apply</i> for an apply process.</li> <li>▪ Specify <i>dequeue</i> for a messaging client.</li> </ul>

**Table 144–11 (Cont.) ADD\_SUBSET\_RULES Procedure Parameters**

Parameter	Description
streams_name	<p>The name of the capture process, synchronous capture, apply process, or messaging client. Do not specify an owner.</p> <p>If NULL, if streams_type is capture, sync_capture, or dequeue, and if one relevant capture process, synchronous capture, or messaging client for the queue exists, then the procedure uses the relevant Oracle Streams client. If no relevant Oracle Streams client exists for the queue, then the procedure creates an Oracle Streams client automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified streams_type for the queue exist, then the procedure raises an error.</p> <p>If NULL, if streams_type is apply, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:</p> <ul style="list-style-type: none"> <li>■ If one existing apply process has the source database specified in source_database and uses the queue specified in queue_name, then the procedure uses this apply process.</li> <li>■ If source_database is NULL and one existing apply process is using the queue specified in queue_name, then the procedure uses this apply process.</li> </ul> <p>If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.</p> <p>If NULL and multiple relevant apply processes exist, then the procedure raises an error.</p> <p>Each apply process and messaging client must have a unique name.</p>
queue_name	<p>The name of the local queue, specified as [ schema_name . ] queue_name. The current database must contain the queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a queue named streams_queue in the strmadmin schema, enter strmadmin.streams_queue for this parameter. If the schema is not specified, then the current user is the default.</p> <p>For capture process or synchronous capture rules, this is the queue into which a capture process or synchronous capture enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.</p>

**Table 144–11 (Cont.) ADD\_SUBSET\_RULES Procedure Parameters**

Parameter	Description
<code>include_tagged_lcr</code>	<p>If <code>TRUE</code>, then the Oracle Streams client performs its action regardless of the tag:</p> <ul style="list-style-type: none"> <li>■ A redo entry is always considered for capture by a capture process, regardless of whether the redo entry has a non-NULL tag.</li> <li>■ A change is always considered for capture by a synchronous capture, regardless of whether the session that makes the change has a non-NULL tag.</li> <li>■ An LCR is always considered for apply by an apply process or dequeue by a messaging client, regardless of whether redo entry or LCR has a non-NULL tag.</li> </ul> <p>If <code>FALSE</code>, then an Oracle Streams client performs its action only when the tag is <code>NULL</code>:</p> <ul style="list-style-type: none"> <li>■ A redo entry is considered for capture by a capture process only when the redo entry contains a <code>NULL</code> tag.</li> <li>■ A change is considered for capture by a synchronous capture only when the session that makes the change has a <code>NULL</code> tag.</li> <li>■ An LCR is considered for apply by an apply process or dequeue by a messaging client only if the LCR contains a <code>NULL</code> tag.</li> </ul> <p>A setting of <code>FALSE</code> is often specified in update-anywhere configurations to avoid sending a change back to its source database.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>
<code>source_database</code>	<p>The global name of the source database. If <code>NULL</code>, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>For capture process rules, specify <code>NULL</code> or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</p> <p>For synchronous capture rules, specify the name of the local database.</p> <p>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database.</p> <p>For messaging client rules, specify <code>NULL</code> if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</p>
<code>insert_rule_name</code>	<p>Contains the system-generated <code>INSERT</code> rule name. This rule handles inserts and updates that must be converted into inserts.</p>



**Table 144–11 (Cont.) ADD\_SUBSET\_RULES Procedure Parameters**

Parameter	Description
update_rule_name	Contains the system-generated UPDATE rule name. This rule handles updates that remain updates.
delete_rule_name	Contains the system-generated DELETE rule name. This rule handles deletes and updates that must be converted into deletes

## Usage Notes

Running this procedure generates three rules for the specified capture process, synchronous capture, apply process, or messaging client: one for INSERT statements, one for UPDATE statements, and one for DELETE statements. For INSERT and DELETE statements, only DML changes that satisfy the condition specified for the `dml_condition` parameter are captured, applied, or dequeued. For UPDATE statements, the following variations are possible:

- If both the new and old values in a DML change satisfy the specified `dml_condition`, then the DML change is captured, applied, or dequeued without any changes.
- If neither the new or old values in a DML change satisfy the specified `dml_condition`, then the DML change is not captured, applied, or dequeued.
- If the old values for a DML change satisfy the specified `dml_condition`, but the new values do not, then the DML change is converted into a delete.
- If the new values for a DML change satisfy the specified `dml_condition`, but the old values do not, then the DML change is converted to an insert.

When an update is converted into an insert or a delete, it is called row migration.

A capture process, synchronous capture, apply process, or messaging client uses the rules for filtering. If the Oracle Streams client does not have a positive rule set, then this procedure creates a positive rule set automatically, and adds the rules for the table to the positive rule set. A subset rule can be added to positive rule set only, not to a negative rule set. Other rules in an existing rule set for the process are not affected. Additional rules can be added using either the `DBMS_STREAMS_ADM` package or the `DBMS_RULE_ADM` package.

Rules for INSERT, UPDATE, and DELETE statements are created automatically when you run this procedure, and these rules are given a system-generated rule name. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. The `ADD_SUBSET_RULES` procedure is overloaded, and the system-generated rule names for INSERT, UPDATE, and DELETE statements are returned.

---



---

**Attention:** Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so might have unpredictable results because row migration would not be performed on LCRs that are not discarded by the negative rule set.

---



---

**See Also:**

- ["Operational Notes"](#) on page 144-9
- ["Security Model"](#) on page 144-5

**Examples**

The following is an example of a rule condition created for filtering DML changes containing an update operation when the `dml_condition` is `region_id = 2`, the `table_name` is `hr.regions`, and the `source_database` is `dbs1.net`:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'  
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'  
AND :dml.get_command_type()='UPDATE'  
AND (:dml.get_value('NEW','REGION_ID') IS NOT NULL)  
AND (:dml.get_value('OLD','REGION_ID') IS NOT NULL)  
AND (:dml.get_value('OLD','REGION_ID').AccessNumber()=2)  
AND (:dml.get_value('NEW','REGION_ID').AccessNumber()=2)
```

## ADD\_TABLE\_PROPAGATION\_RULES Procedure

This procedure adds table rules to the positive rule set for a propagation, or adds table rules to the negative rule set for a propagation, and creates the specified propagation if it does not exist.

This procedure is overloaded. One version of this procedure contains two OUT parameters, and the other does not.

### Syntax

```
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES (
    table_name           IN   VARCHAR2,
    streams_name        IN   VARCHAR2  DEFAULT NULL,
    source_queue_name   IN   VARCHAR2,
    destination_queue_name IN VARCHAR2,
    include_dml         IN   BOOLEAN   DEFAULT TRUE,
    include_ddl        IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
    source_database     IN   VARCHAR2  DEFAULT NULL,
    dml_rule_name       OUT  VARCHAR2,
    ddl_rule_name       OUT  VARCHAR2,
    inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
    and_condition       IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue     IN   BOOLEAN   DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES (
    table_name           IN   VARCHAR2,
    streams_name        IN   VARCHAR2  DEFAULT NULL,
    source_queue_name   IN   VARCHAR2,
    destination_queue_name IN VARCHAR2,
    include_dml         IN   BOOLEAN   DEFAULT TRUE,
    include_ddl        IN   BOOLEAN   DEFAULT FALSE,
    include_tagged_lcr  IN   BOOLEAN   DEFAULT FALSE,
    source_database     IN   VARCHAR2  DEFAULT NULL,
    inclusion_rule      IN   BOOLEAN   DEFAULT TRUE,
    and_condition       IN   VARCHAR2  DEFAULT NULL,
    queue_to_queue     IN   BOOLEAN   DEFAULT NULL);
```

### Parameters

**Table 144–12 ADD\_TABLE\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
table_name	The name of the table specified as [ <i>schema_name</i> .] <i>table_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.
streams_name	The name of the propagation. Do not specify an owner. If the specified propagation does not exist, then the procedure creates it automatically. If NULL and a propagation exists for the same source queue and destination queue (including database link), then the procedure uses this propagation. If NULL and no propagation exists for the same source queue and destination queue (including database link), then the procedure creates a propagation automatically with a system-generated name.

**Table 144–12 (Cont.) ADD\_TABLE\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
source_queue_name	<p>The name of the source queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the source queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a source queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p>
destination_queue_name	<p>The name of the destination queue, including a database link, specified as [<i>schema_name</i>.] <i>queue_name</i>[<i>@dblink_name</i>], if the destination queue is in a remote database. The queue must be ANYDATA type.</p> <p>For example, to specify a destination queue named <i>streams_queue</i> in the <i>strmadmin</i> schema and use a database link named <i>db2.net</i>, enter <i>strmadmin.streams_queue@db2.net</i> for this parameter.</p> <p>If the schema is not specified, then the current user is the default.</p> <p>If the database link is omitted, then the procedure uses the global name of the current database, and the source queue and destination queue must be in the same database.</p> <p><b>Note:</b> Connection qualifiers are not allowed.</p>
include_dml	<p>If <b>TRUE</b>, then the procedure creates a rule for DML changes. If <b>FALSE</b>, then the procedure does not create a DML rule. <b>NULL</b> is not permitted.</p>
include_ddl	<p>If <b>TRUE</b>, then the procedure creates a rule for DDL changes. If <b>FALSE</b>, then the procedure does not create a DDL rule. <b>NULL</b> is not permitted.</p> <p>The generated rule evaluates to <b>TRUE</b> for any DDL change that operates on the table or on an object that is part of the table, such as an index or trigger on the table. The rule evaluates to <b>FALSE</b> for any DDL change that either does not refer to the table or refers to the table in a subordinate way. For example, the rule evaluates to <b>FALSE</b> for changes that create synonyms or views based on the table. The rule also evaluates to <b>FALSE</b> for a change to a PL/SQL subprogram that refers to the table.</p>

**Table 144–12 (Cont.) ADD\_TABLE\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
include_tagged_lcr	<p>If <code>TRUE</code>, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to <code>TRUE</code> regardless of whether a logical change record (LCR) has a non-<code>NULL</code> tag. If the rules are added to the positive rule set for the propagation, then an LCR is always considered for propagation, regardless of whether it has a non-<code>NULL</code> tag. If the rules are added to a positive rule set, then setting this parameter to <code>TRUE</code> is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the propagation, then whether an LCR is discarded does not depend on the tag for the LCR.</p> <p>If <code>FALSE</code>, then the procedure adds a condition to each generated rule that causes the rule to evaluate to <code>TRUE</code> only if an LCR has a <code>NULL</code> Oracle Streams tag. If the rules are added to the positive rule set for the propagation, then an LCR is considered for propagation only when the LCR contains a <code>NULL</code> tag. If the rules are added to a positive rule set, then setting this parameter to <code>FALSE</code> might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the propagation, then an LCR can be discarded only if it has a <code>NULL</code> tag.</p> <p>Usually, specify <code>TRUE</code> for this parameter if the <code>inclusion_rule</code> parameter is set to <code>FALSE</code>.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>
source_database	<p>The global name of the source database. The source database is where the change originated. If <code>NULL</code>, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify <code>DBS1</code> and the domain is <code>.NET</code>, then the procedure specifies <code>DBS1.NET</code> automatically.</p> <p>Oracle recommends that you specify a source database for propagation rules.</p>
dml_rule_name	<p>If <code>include_dml</code> is <code>TRUE</code>, then this parameter contains the DML rule name.</p> <p>If <code>include_dml</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
ddl_rule_name	<p>If <code>include_ddl</code> is <code>TRUE</code>, then this parameter contains the DDL rule name.</p> <p>If <code>include_ddl</code> is <code>FALSE</code>, then this parameter contains a <code>NULL</code>.</p>
inclusion_rule	<p>If <code>inclusion_rule</code> is <code>TRUE</code>, then the procedure adds the rules to the positive rule set for the propagation.</p> <p>If <code>inclusion_rule</code> is <code>FALSE</code>, then the procedure adds the rules to the negative rule set for the propagation.</p> <p>In either case, the system creates the rule set if it does not exist.</p>

**Table 144–12 (Cont.) ADD\_TABLE\_PROPAGATION\_RULES Procedure Parameters**

Parameter	Description
<code>and_condition</code>	<p>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</p> <pre>(system_condition) AND (and_condition)</pre> <p>The variable in the specified condition must be <code>:lcr</code>. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</p> <pre>:lcr.get_tag() = HEXTORAW('02')</pre> <p>The <code>:lcr</code> in the specified condition is converted to <code>:dml</code> or <code>:ddl</code>, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</p> <p>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the <code>include_dml</code> parameter and FALSE for the <code>include_ddl</code> parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the <code>include_dml</code> parameter and TRUE for the <code>include_ddl</code> parameter.</p> <p><b>See Also:</b> <a href="#">Chapter 248, "Logical Change Record TYPES"</a></p>
<code>queue_to_queue</code>	<p>If TRUE or NULL, then a new propagation created by this procedure is a queue to queue propagation. A queue-to-queue propagation always has its own propagation job and uses a service for automatic failover when the destination queue is a buffered queue in an Oracle Real Application Clusters (Oracle RAC) database.</p> <p>If FALSE, then a new propagation created by this procedure is a queue-to-dblink propagation. A queue-to-dblink propagation can share a propagation job with other propagations that use the same database link and does not support automatic failover in an Oracle RAC environment.</p> <p>This procedure cannot change the queue to queue property of an exiting propagation. If the specified propagation exists, then the procedure behaves in the following way for each setting:</p> <ul style="list-style-type: none"> <li>■ If TRUE and the specified propagation is not a queue to queue propagation, then the procedure raises an error.</li> <li>■ If FALSE and the specified propagation is a queue to queue propagation, then the procedure raises an error.</li> <li>■ If NULL, then the procedure does not change the queue to queue property of the propagation.</li> </ul> <p><b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about queue-to-queue propagations</p>

## Usage Notes

This procedure configures propagation using the current user. Only one propagation is allowed between a particular source queue and destination queue.

This procedure creates DML and DDL rules automatically based on `include_dml` and `include_ddl` parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The

sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A propagation uses the rules for filtering.

**See Also:**

- ["Operational Notes"](#) on page 144-9 and ["Propagation Rules for LCRs"](#) on page 144-11 for more information about the rules created by this procedure
- ["Security Model"](#) on page 144-5

## Examples

The following is an example of a table rule condition created for filtering DML statements:

```
((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))  
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

## ADD\_TABLE\_RULES Procedure

This procedure adds rules to a rule set of one of the following types of Oracle Streams clients:

- When the `streams_type` parameter is set to `capture`, this procedure adds capture process rules for capturing changes to a specified table. See ["Capture Process Rules for Changes in the Redo Log"](#) on page 144-10 for more information about these rules.
- When the `streams_type` parameter is set to `sync_capture`, this procedure adds rules for capturing changes to a specified table. See ["Synchronous Capture Rules for DML Changes to Tables"](#) on page 144-11 for more information about these rules.
- When the `streams_type` parameter is set to `apply` and the `streams_name` parameter specifies the name of an apply process, this procedure adds apply process rules for applying logical change records (LCRs) in a queue that contain changes to a specified table. The rules can specify that the LCRs must be from a particular source database. See ["Apply Process Rules for LCRs"](#) on page 144-12 for more information about these rules.
- When the `streams_type` parameter is set to `dequeue`, this procedure adds messaging client rules for dequeuing persistent LCRs from a queue that contain changes to a specified table. The rules can specify that the LCRs must be from a particular source database. See ["Messaging Client Rules for LCRs"](#) on page 144-14 for more information about these rules.

This procedure creates the specified capture process, synchronous capture, apply process, or messaging client if it does not exist.

This procedure is overloaded. One version of this procedure contains two `OUT` parameters, and the other does not.

---



---

### Note:

- Currently, messaging clients cannot dequeue buffered messages.
  - The invoking user must be granted the `DBA` role to create a synchronous capture.
- 
- 

## Syntax

```
DBMS_STREAMS_ADM.ADD_TABLE_RULES (
    table_name          IN    VARCHAR2,
    streams_type        IN    VARCHAR2,
    streams_name        IN    VARCHAR2 DEFAULT NULL,
    queue_name          IN    VARCHAR2 DEFAULT 'streams_queue',
    include_dml         IN    BOOLEAN  DEFAULT TRUE,
    include_ddl         IN    BOOLEAN  DEFAULT FALSE,
    include_tagged_lcr  IN    BOOLEAN  DEFAULT FALSE,
    source_database     IN    VARCHAR2 DEFAULT NULL,
    dml_rule_name       OUT   VARCHAR2,
    ddl_rule_name       OUT   VARCHAR2,
    inclusion_rule      IN    BOOLEAN  DEFAULT TRUE,
    and_condition       IN    VARCHAR2 DEFAULT NULL);
```

```
DBMS_STREAMS_ADM.ADD_TABLE_RULES (
    table_name          IN    VARCHAR2,
```



```

streams_type      IN  VARCHAR2 ,
streams_name      IN  VARCHAR2 DEFAULT NULL ,
queue_name        IN  VARCHAR2 DEFAULT 'streams_queue' ,
include_dml       IN  BOOLEAN  DEFAULT TRUE ,
include_ddl       IN  BOOLEAN  DEFAULT FALSE ,
include_tagged_lcr IN  BOOLEAN  DEFAULT FALSE ,
source_database   IN  VARCHAR2 DEFAULT NULL ,
inclusion_rule     IN  BOOLEAN  DEFAULT TRUE ,
and_condition     IN  VARCHAR2 DEFAULT NULL) ;

```

## Parameters

**Table 144–13 ADD\_TABLE\_RULES Procedure Parameters**

Parameter	Description
table_name	<p>The name of the table specified as [<i>schema_name</i>.] <i>object_name</i>. For example, hr.employees. If the schema is not specified, then the current user is the default.</p> <p>You can specify a table that does not yet exist, because Oracle Streams does not validate the existence of the table.</p>
streams_type	<p>The type of Oracle Streams client:</p> <ul style="list-style-type: none"> <li>■ Specify <i>capture</i> for a capture process.</li> <li>■ Specify <i>sync_capture</i> for a synchronous capture.</li> <li>■ Specify <i>apply</i> for an apply process.</li> <li>■ Specify <i>dequeue</i> for a messaging client.</li> </ul>
streams_name	<p>The name of the capture process, synchronous capture, apply process, or messaging client. Do not specify an owner.</p> <p>If NULL, if <i>streams_type</i> is <i>capture</i>, <i>sync_capture</i>, or <i>dequeue</i>, and if one relevant capture process, synchronous capture, or messaging client for the queue exists, then the procedure uses the relevant Oracle Streams client. If no relevant Oracle Streams client exists for the queue, then the procedure creates an Oracle Streams client automatically with a system-generated name. If NULL and multiple Oracle Streams clients of the specified <i>streams_type</i> for the queue exist, then the procedure raises an error.</p> <p>If NULL, if <i>streams_type</i> is <i>apply</i>, and if one relevant apply process exists, then the procedure uses the relevant apply process. The relevant apply process is identified in one of the following ways:</p> <ul style="list-style-type: none"> <li>■ If one existing apply process has the source database specified in <i>source_database</i> and uses the queue specified in <i>queue_name</i>, then the procedure uses this apply process.</li> <li>■ If <i>source_database</i> is NULL and one existing apply process is using the queue specified in <i>queue_name</i>, then the procedure uses this apply process.</li> </ul> <p>If NULL and no relevant apply process exists, then the procedure creates an apply process automatically with a system-generated name.</p> <p>If NULL and multiple relevant apply processes exist, then the procedure raises an error.</p> <p>Each apply process and messaging client must have a unique name.</p>

**Table 144–13 (Cont.) ADD\_TABLE\_RULES Procedure Parameters**

Parameter	Description
queue_name	<p>The name of the local queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the queue, and the queue must be ANYDATA type.</p> <p>For example, to specify a queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.</p> <p>For capture process or synchronous capture rules, this is the queue into which a capture process or synchronous capture enqueues LCRs. For apply process rules, this is the queue from which an apply process dequeues messages. For messaging client rules, this is the queue from which a messaging client dequeues messages.</p>
include_dml	<p>If <code>TRUE</code>, then the procedure creates a DML rule for DML changes. If <code>FALSE</code>, then the procedure does not create a DML rule. <code>NULL</code> is not permitted.</p>
include_ddl	<p>If <code>TRUE</code>, then the procedure creates a DDL rule for DDL changes. If <code>FALSE</code>, then the procedure does not create a DDL rule. <code>NULL</code> is not permitted.</p> <p>The generated rule evaluates to <code>TRUE</code> for any DDL change that operates on the table or on an object that is part of the table, such as an index or trigger on the table. The rule evaluates to <code>FALSE</code> for any DDL change that either does not refer to the table or refers to the table in a subordinate way. For example, the rule evaluates to <code>FALSE</code> for changes that create synonyms or views based on the table. The rule also evaluates to <code>FALSE</code> for a change to a PL/SQL subprogram that refers to the table.</p>

**Table 144–13 (Cont.) ADD\_TABLE\_RULES Procedure Parameters**

Parameter	Description
<code>include_tagged_lcr</code>	<p>If <code>TRUE</code>, then the procedure does not add a condition regarding Oracle Streams tags to the generated rules. Therefore, these rules can evaluate to <code>TRUE</code> regardless of whether a redo entry, session, or LCR has a non-<code>NULL</code> tag. If the rules are added to the positive rule set for the Oracle Streams client, then the Oracle Streams client performs its action regardless of the tag:</p> <ul style="list-style-type: none"> <li>■ A redo entry is always considered for capture by a capture process, regardless of whether the redo entry has a non-<code>NULL</code> tag.</li> <li>■ A change is always considered for capture by a synchronous capture, regardless of whether the session that makes the change has a non-<code>NULL</code> tag.</li> <li>■ An LCR is always considered for apply by an apply process or dequeue by a messaging client, regardless of whether redo entry or LCR has a non-<code>NULL</code> tag.</li> </ul> <p>If the rules are added to a positive rule set, then setting this parameter to <code>TRUE</code> is appropriate for a full (for example, standby) copy of a database. If the rules are added to the negative rule set for the Oracle Streams client, then whether a database change is discarded does not depend on the tag.</p> <p>If <code>FALSE</code>, then the procedure adds a condition to each generated rule that causes the rule to evaluate to <code>TRUE</code> only if a redo entry, session, or LCR has a <code>NULL</code> Oracle Streams tag. If the rules are added to the positive rule set for an Oracle Streams client, then the Oracle Streams client performs its action only when the tag is <code>NULL</code>:</p> <ul style="list-style-type: none"> <li>■ A redo entry is considered for capture by a capture process only when the redo entry contains a <code>NULL</code> tag.</li> <li>■ A change is considered for capture by a synchronous capture only when the session that makes the change has a <code>NULL</code> tag.</li> <li>■ An LCR is considered for apply by an apply process or dequeue by a messaging client only if the LCR contains a <code>NULL</code> tag.</li> </ul> <p>If the rules are added to a positive rule set, then setting this parameter to <code>FALSE</code> might be appropriate in update-anywhere configurations to avoid sending a change back to its source database. If the rules are added to the negative rule set for the Oracle Streams client, then a database change can be discarded only if it has a <code>NULL</code> tag.</p> <p>A setting of <code>FALSE</code> is often specified in update-anywhere configurations to avoid sending a change back to its source database.</p> <p>Usually, specify <code>TRUE</code> for this parameter if the <code>inclusion_rule</code> parameter is set to <code>FALSE</code>.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i> for more information about tags</p>

**Table 144–13 (Cont.) ADD\_TABLE\_RULES Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database. If NULL, then the procedure does not add a condition regarding the source database to the generated rules.</p> <p>For capture process rules, specify NULL or the global name of the local database if you are creating a capture process locally at the source database. If you are adding rules to a downstream capture process rule set at a downstream database, then specify the source database of the changes that will be captured.</p> <p>For synchronous capture rules, specify the name of the local database.</p> <p>For apply process rules, specify the source database of the changes that will be applied by the apply process. The source database is the database where the changes originated. If an apply process applies captured messages, then the apply process can apply messages from only one capture process at one source database.</p> <p>For messaging client rules, specify NULL if you do not want the rules created by this procedure to have a condition for the source database. Specify a source database if you want the rules created by this procedure to have a condition for the source database. The source database is part of the information in an LCR, and user-constructed LCRs might or might not have this information.</p> <p>If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.</p>
dml_rule_name	<p>If include_dml is TRUE, then this parameter contains the DML rule name.</p> <p>If include_dml is FALSE, then this parameter contains a NULL.</p>
ddl_rule_name	<p>If include_ddl is TRUE, then this parameter contains the DDL rule name.</p> <p>If include_ddl is FALSE, then this parameter contains a NULL.</p>
inclusion_rule	<p>If inclusion_rule is TRUE, then the procedure adds the rules to the positive rule set for the Oracle Streams client.</p> <p>If inclusion_rule is FALSE, then the procedure adds the rules to the negative rule set for the Oracle Streams client. A synchronous capture cannot have a negative rule set. Specifying FALSE for a synchronous capture raises an error.</p> <p>In either case, the system creates the rule set if it does not exist.</p>

**Table 144–13 (Cont.) ADD\_TABLE\_RULES Procedure Parameters**

Parameter	Description
and_condition	<p>If non-NULL, appends the specified condition to the system-generated rule condition using an AND clause in the following way:</p> <pre>(system_condition) AND (and_condition)</pre> <p>The variable in the specified condition must be :lcr. For example, to specify that the table rules generated by the procedure evaluate to TRUE only if the Oracle Streams tag is the hexadecimal equivalent of '02', specify the following condition:</p> <pre>:lcr.get_tag() = HEXTORAW('02')</pre> <p>The :lcr in the specified condition is converted to :dml or :ddl, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule.</p> <p>Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify TRUE for the include_dml parameter and FALSE for the include_ddl parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify FALSE for the include_dml parameter and TRUE for the include_ddl parameter.</p> <p><b>See Also:</b> <a href="#">Chapter 248, "Logical Change Record TYPES"</a></p>

## Usage Notes

This procedure creates DML and DDL rules automatically based on include\_dml and include\_ddl parameter values, respectively. Each rule has a system-generated rule name that consists of the table name with a sequence number appended to it. The sequence number is used to avoid naming conflicts. If the table name plus the sequence number is too long, then the table name is truncated. A capture process, synchronous capture, apply process, or messaging client uses the rules for filtering.

### See Also:

- ["Operational Notes"](#) on page 144-9
- ["Security Model"](#) on page 144-5

## Examples

The following is an example of a table rule condition created for DML changes:

```
((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

## CLEANUP\_INSTANTIATION\_SETUP Procedure

This procedure removes an Oracle Streams replication configuration that was set up by the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures in this package. This procedure either remove the configuration directly, or it can generate a script that removes the configuration.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

---



---

**Attention:** When the `CLEANUP_INSTANTIATION_SETUP` procedure is run, the parameter values must match the parameter values specified when the corresponding `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures were run, except for the values of the following parameters: `perform_actions`, `script_name`, and `script_directory_object`.

---



---

### See Also:

- ["PRE\\_INSTANTIATION\\_SETUP Procedure"](#) on page 144-141
- ["POST\\_INSTANTIATION\\_SETUP Procedure"](#) on page 144-136
- ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

### Syntax

```
DBMS_STREAMS_ADM.CLEANUP_INSTANTIATION_SETUP (
    maintain_mode           IN VARCHAR2,
    tablespace_names       IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
    source_database        IN VARCHAR2,
    destination_database   IN VARCHAR2,
    perform_actions        IN BOOLEAN   DEFAULT TRUE,
    script_name            IN VARCHAR2   DEFAULT NULL,
    script_directory_object IN VARCHAR2   DEFAULT NULL,
    capture_name           IN VARCHAR2   DEFAULT NULL,
    capture_queue_table    IN VARCHAR2   DEFAULT NULL,
    capture_queue_name     IN VARCHAR2   DEFAULT NULL,
    capture_queue_user     IN VARCHAR2   DEFAULT NULL,
    propagation_name      IN VARCHAR2   DEFAULT NULL,
    apply_name             IN VARCHAR2   DEFAULT NULL,
    apply_queue_table      IN VARCHAR2   DEFAULT NULL,
    apply_queue_name       IN VARCHAR2   DEFAULT NULL,
    apply_queue_user       IN VARCHAR2   DEFAULT NULL,
    bi_directional         IN BOOLEAN   DEFAULT FALSE,
    change_global_name     IN BOOLEAN   DEFAULT FALSE);
```

## Parameters

**Table 144–14 CLEANUP\_INSTANTIATION\_SETUP Procedure Parameters**

Parameter	Description
maintain_mode	<p>Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ GLOBAL to clean up the Oracle Streams configuration that maintained the entire database in both the source and destination databases</li> <li>■ TRANSPORTABLE TABLESPACES to cleanup the Oracle Streams configuration that maintained a set of tablespaces at both the source and destination database</li> </ul>
tablespace_names	<p>If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database.</p> <p>A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects.</p> <p>If maintain_mode is set to GLOBAL, then specify an empty tablespace set.</p> <p>Regardless of the maintain_mode setting, an error is raised if the tablespace_names parameter is not set or is set to NULL.</p> <p><b>See Also:</b> <a href="#">TABLESPACE_SET Table Type</a> on page 149-9</p>
source_database	<p>The global name of the source database.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>
destination_database	<p>The global name of the destination database. A database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
perform_actions	<p>If TRUE, then this procedure performs the necessary actions to clean up the Oracle Streams configuration directly.</p> <p>If FALSE, then the procedure does not perform the necessary actions to clean up the Oracle Streams configuration directly.</p> <p>Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</p> <ul style="list-style-type: none"> <li>■ script_name</li> <li>■ script_directory_object</li> </ul>

**Table 144–14 (Cont.) CLEANUP\_INSTANTIATION\_SETUP Procedure Parameters**

Parameter	Description
<code>script_name</code>	<p>If non-NULL and the <code>perform_actions</code> parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to clean up the Oracle Streams configuration. If a file with the specified script name exists in the specified directory for the <code>script_directory_object</code> parameter, then the statements are appended to the existing file.</p> <p>If non-NULL and the <code>perform_actions</code> parameter is TRUE, then this procedure generates the specified script and performs the actions to clean up the Oracle Streams configuration directly.</p> <p>If NULL and the <code>perform_actions</code> parameter is TRUE, then this procedure directly performs the actions to clean up the Oracle Streams configuration without generating a script.</p> <p>If NULL and the <code>perform_actions</code> parameter is FALSE, then the procedure raises an error.</p>
<code>script_directory_object</code>	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the <code>script_name</code> parameter is NULL, then this parameter is ignored, and this procedure does not generate a script.</p> <p>If NULL and the <code>script_name</code> parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>
<code>capture_name</code>	<p>The name of the capture processes configured to capture changes in the Oracle Streams configuration. Do not specify an owner.</p> <p>If NULL, then the procedure automatically identifies the capture processes with system-generated names created by the <code>PRE_INSTANTIATION_SETUP</code> and <code>POST_INSTANTIATION_SETUP</code> procedures.</p>
<code>capture_queue_table</code>	<p>The name of the queue table for each queue used by a capture process, specified as [<i>schema_name</i>.] <i>queue_table_name</i>. For example, <code>strmadmin.streams_queue_table</code>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the procedure automatically identifies the capture queue tables with system-generated names created by the <code>PRE_INSTANTIATION_SETUP</code> and <code>POST_INSTANTIATION_SETUP</code> procedures.</p>
<code>capture_queue_name</code>	<p>The name of each queue used by a capture process, specified as [<i>schema_name</i>.] <i>queue_name</i>. For example, <code>strmadmin.streams_queue</code>.</p> <p>If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.</p> <p>If NULL, then the procedure automatically identifies the capture queues with system-generated names created by the <code>PRE_INSTANTIATION_SETUP</code> and <code>POST_INSTANTIATION_SETUP</code> procedures.</p>
<code>capture_queue_user</code>	<p>The name of the user who has <code>ENQUEUE</code> and <code>DEQUEUE</code> privileges for the queue at the source database. This user is a secure queue user of the queue.</p>



**Table 144–14 (Cont.) CLEANUP\_INSTANTIATION\_SETUP Procedure Parameters**

Parameter	Description
propagation_name	<p>The name of the propagations configured to propagate changes in the Oracle Streams configuration. Do not specify an owner.</p> <p>If NULL, then the procedure automatically identifies the propagations with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</p>
apply_name	<p>The name of the apply processes configured to apply changes in the Oracle Streams configuration. Do not specify an owner.</p> <p>If NULL, then the procedure automatically identifies the apply processes with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</p>
apply_queue_table	<p>The name of the queue table for each queue used by an apply process, specified as <i>[schema_name.]queue_table_name</i>. For example, <i>strmadmin.streams_queue_table</i>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the procedure automatically identifies the apply queue tables with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</p>
apply_queue_name	<p>The name of each queue used by an apply process, specified as <i>[schema_name.]queue_name</i>. For example, <i>strmadmin.streams_queue</i>.</p> <p>If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.</p> <p>If NULL, then the procedure automatically identifies the apply queues with system-generated names created by the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.</p>
apply_queue_user	<p>The name of the user who has ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user is a secure queue user of the queue.</p>
bi_directional	<p>Specify TRUE if the Oracle Streams replication configuration is bi-directional between the database specified in <i>source_database</i> and the database specified in <i>destination_database</i>.</p> <p>Specify FALSE if the Oracle Streams replication configuration is one way replication from the current database to the database specified in <i>destination_database</i>.</p>
change_global_name	<p>If TRUE, then the procedure changes the global name of the database specified in <i>destination_database</i> to match the global name of the current database.</p> <p>If FALSE, then the procedure does not change the global name of the database specified in <i>destination_database</i>.</p>

## DELETE\_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which deletes a column from a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

---



---

### Note:

- The `DELETE_COLUMN` procedure supports the same data types supported by Oracle Streams capture processes.
  - The `DELETE_COLUMN` procedure is useful when you want to delete a relatively small number of columns in a row LCR. To delete most of the columns in a row LCR and keep a relatively small number of columns, consider using the `KEEP_COLUMNS` procedure in this package.
  - Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.
- 
- 

### See Also:

- *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations and about the data types supported by Oracle Streams capture processes
- "[KEEP\\_COLUMNS Procedure](#)" on page 144-87

## Syntax

```
DBMS_STREAMS_ADM.DELETE_COLUMN(
  rule_name      IN  VARCHAR2,
  table_name     IN  VARCHAR2,
  column_name    IN  VARCHAR2,
  value_type     IN  VARCHAR2  DEFAULT '*',
  step_number    IN  NUMBER     DEFAULT 0,
  operation      IN  VARCHAR2  DEFAULT 'ADD');
```

## Parameters

**Table 144–15** *DELETE\_COLUMN Procedure Parameters*

Parameter	Description
<code>rule_name</code>	The name of the rule, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . If <code>NULL</code> , then the procedure raises an error.  For example, to specify a rule in the <code>hr</code> schema named <code>employees12</code> , enter <code>hr.employees12</code> . If the schema is not specified, then the current user is the default.

**Table 144–15 (Cont.) DELETE\_COLUMN Procedure Parameters**

Parameter	Description
table_name	The name of the table from which the column is deleted in the row LCR, specified as [ <i>schema_name</i> . ] <i>object_name</i> . For example, <i>hr.employees</i> . If the schema is not specified, then the current user is the default.
column_name	The name of the column deleted from each row LCR that satisfies the rule.
value_type	Specify 'NEW' to delete the column from the new values in the row LCR. Specify 'OLD' to delete the column from the old values in the row LCR. Specify '*' to delete the column from both the old and new values in the row LCR.
step_number	The order of execution of the transformation.  <b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule. See " <a href="#">Usage Notes</a> " on page 144-81 for more information about this parameter.

## Usage Notes

When 'REMOVE' is specified for the *operation* parameter, all of the delete column declarative rule-based transformations for the specified rule are removed that match the specified *table\_name*, *column\_name*, and *step\_number* parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the DELETE\_COLUMN procedure when one or more of these parameters is NULL:

table_name	column_name	step_number	Result
NULL	NULL	NULL	Remove all delete column transformations for the specified rule.
NULL	NULL	non-NULL	Remove all delete column transformations with the specified <i>step_number</i> for the specified rule.
NULL	non-NULL	non-NULL	Remove all delete column transformations with the specified <i>column_name</i> and <i>step_number</i> for the specified rule.
non-NULL	NULL	non-NULL	Remove all delete column transformations with the specified <i>table_name</i> and <i>step_number</i> for the specified rule.
NULL	non-NULL	NULL	Remove all delete column transformations with the specified <i>column_name</i> for the specified rule.
non-NULL	non-NULL	NULL	Remove all delete column transformations with the specified <i>table_name</i> and <i>column_name</i> for the specified rule.
non-NULL	NULL	NULL	Remove all delete column transformations with the specified <i>table_name</i> for the specified rule.

<b>table_name</b>	<b>column_name</b>	<b>step_number</b>	<b>Result</b>
non-NULL	non-NULL	non-NULL	Remove all delete column transformations with the specified <code>table_name</code> , <code>column_name</code> , and <code>step_number</code> for the specified rule.

## GET\_MESSAGE\_TRACKING Function

Returns the tracking label for the current session.

**See Also:** [SET\\_MESSAGE\\_TRACKING Procedure](#) on page 144-166

### Syntax

```
DBMS_STREAMS_ADM.GET_MESSAGE_TRACKING  
RETURN VARCHAR2;
```

## GET\_SCN\_MAPPING Procedure

This procedure gets information about the system change number (SCN) values to use for Oracle Streams capture and apply processes in an Oracle Streams replication environment. This information can be used for the following purposes:

- To recover transactions after point-in-time recovery is performed on a source database in a multiple source Oracle Streams environment
- To run flashback queries for the corresponding SCN at a source database and destination database in an Oracle Streams single source replication environment

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about point-in-time recovery and flashback queries in an Oracle Streams replication environment

### Syntax

```
DBMS_STREAMS_ADM.GET_SCN_MAPPING (
  apply_name          IN  VARCHAR2,
  src_pit_scn         IN  NUMBER,
  dest_instantiation_scn OUT NUMBER,
  dest_start_scn      OUT NUMBER,
  dest_skip_txn_ids   OUT DBMS_UTILITY.NAME_ARRAY);
```

### Parameters

**Table 144–16** GET\_SCN\_MAPPING Procedure Parameters

Parameter	Description
apply_name	Name of the apply process which applies logical change records (LCRs) from the source database. The procedure raises an error if the specified apply process does not exist.
src_pit_scn	The SCN at the source database. For point-in-time recovery, specify the point-in-time recovery SCN at the source database. If the specified SCN is greater than the source commit SCN of the last applied transaction, then NULL is returned for both dest_start_scn and dest_instantiation_scn. In this case, no values can be returned for these parameters because the corresponding transaction has not been applied at the destination database yet.
dest_instantiation_scn	The SCN at the destination database that corresponds to the specified src_pit_scn at the source database. For point-in-time recovery, use this value for the instantiation SCNs at the source database during recovery.
dest_start_scn	For point in time recovery, the SCN to use for the start_scn parameter for the recovery capture process.

**Table 144–16 (Cont.) GET\_SCN\_MAPPING Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>dest_skip_txn_ids</code>	<p>Transaction IDs of transactions that were skipped at the <code>dest_instantiation_scn</code> because the apply process was applying nondependent transactions out of order.</p> <p>For point in time recovery, these transaction IDs should be ignored by the recovery apply process.</p> <p>This parameter is relevant only if the <code>commit_serialization</code> for the apply process that applied these transactions was set to <code>DEPENDENT_TRANSACTIONS</code>, and the transactions were applied out of order.</p>

## GET\_TAG Function

This function gets the binary tag for all redo entries generated by the current session.

**See Also:**

- ["SET\\_TAG Procedure"](#) on page 144-171
- *Oracle Streams Replication Administrator's Guide* for more information about tags

### Syntax

```
DBMS_STREAMS_ADM.GET_TAG  
RETURN RAW;
```

### Examples

The following example illustrates how to display the current logical change record (LCR) tag as output:

```
SET SERVEROUTPUT ON  
DECLARE  
    raw_tag RAW(2000);  
BEGIN  
    raw_tag := DBMS_STREAMS_ADM.GET_TAG();  
    DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));  
END;  
/
```

You can also display the value by querying the DUAL view:

```
SELECT DBMS_STREAMS_ADM.GET_TAG FROM DUAL;
```



## KEEP\_COLUMNS Procedure

This procedure either adds or removes a declarative rule-based transformation which keeps a list of columns in a row logical change record (LCR) that satisfies the specified rule. The transformation deletes columns that are not in the list from the row LCR.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

This procedure is overloaded. The `column_list` parameter is type `VARCHAR2` and the `column_table` parameter is type `DBMS_UTILITY.LNAME_ARRAY`. These parameters enable you to enter the list of columns in different ways and are mutually exclusive.

---



---

### Note:

- The `KEEP_COLUMNS` procedure supports the same data types supported by Oracle Streams capture processes.
  - The `KEEP_COLUMNS` procedure is useful when you want to keep a relatively small number of columns in a row LCR. To keep most of the columns in a row LCR and delete a relatively small number of columns, consider using the `DELETE_COLUMN` procedure in this package.
  - Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.
- 
- 

### See Also:

- *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations and about the data types supported by Oracle Streams capture processes
- ["DELETE\\_COLUMN Procedure"](#) on page 144-80

## Syntax

```
DBMS_STREAMS_ADM.KEEP_COLUMNS (
  rule_name      IN  VARCHAR2,
  table_name     IN  VARCHAR2,
  column_list    IN  VARCHAR2,
  value_type     IN  VARCHAR2 DEFAULT '*',
  step_number    IN  NUMBER DEFAULT 0,
  operation      IN  VARCHAR2 DEFAULT 'ADD');
```

```
DBMS_STREAMS_ADM.KEEP_COLUMNS (
  rule_name      IN  VARCHAR2,
  table_name     IN  VARCHAR2,
  column_table   IN  DBMS_UTILITY.LNAME_ARRAY,
  value_type     IN  VARCHAR2 DEFAULT '*',
  step_number    IN  NUMBER DEFAULT 0,
```

```
operation      IN VARCHAR2 DEFAULT 'ADD');
```

## Parameters

**Table 144–17** KEEP\_COLUMNS Procedure Parameters

Parameter	Description
rule_name	The name of the rule, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . If NULL, then the procedure raises an error.  For example, to specify a rule in the hr schema named employees12, enter hr.employees12. If the schema is not specified, then the current user is the default.
table_name	The name of the table for which the columns are kept in the row LCR, specified as [ <i>schema_name.</i> ] <i>object_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.
column_list	The names of the columns kept for each row LCR that satisfies the rule. Specify a comma-delimited list of type VARCHAR2. The transformation removes columns that are not in the list from the row LCR.  If this parameter is set to NULL, and the column_table parameter is also set to NULL, then the procedure raises an error.
column_table	The names of the columns kept for each row LCR that satisfies the rule. Specify a PL/SQL associative array of type DBMS_UTILITY.LNAME_ARRAY, where each element is the name of a column. The first schema should be in position 1. The last position must be NULL.  The transformation removes columns that are not in the table from the row LCR.  If this parameter is set to NULL, and the column_list parameter is also set to NULL, then the procedure raises an error.
value_type	Specify 'NEW' to keep the columns in the new values in the row LCR. Specify 'OLD' to keep the columns in the old values in the row LCR. Specify '*' to keep the columns in both the old and new values in the row LCR.
step_number	The order of execution of the transformation.  <b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.  See " <a href="#">Usage Notes</a> " on page 144-88 for more information about this parameter.

## Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the keep columns declarative rule-based transformations for the specified rule are removed that match the specified table\_name, column\_list, column\_table, and step\_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the KEEP\_COLUMNS procedure when one or more of these parameters is NULL:

table_name	column_list/column_table	step_number	Result
NULL	NULL	NULL	Remove all keep columns transformations for the specified rule.

<b>table_name</b>	<b>column_list/column_table</b>	<b>step_number</b>	<b>Result</b>
NULL	NULL	non-NULL	Remove all keep columns transformations with the specified <code>step_number</code> for the specified rule.
NULL	non-NULL	non-NULL	Remove all keep columns transformations with the specified <code>column_list/column_table</code> and <code>step_number</code> for the specified rule.
non-NULL	NULL	non-NULL	Remove all keep columns transformations with the specified <code>table_name</code> and <code>step_number</code> for the specified rule.
NULL	non-NULL	NULL	Remove all keep columns transformations with the specified <code>column_list/column_table</code> for the specified rule.
non-NULL	non-NULL	NULL	Remove all keep columns transformations with the specified <code>table_name</code> and <code>column_list/column_table</code> for the specified rule.
non-NULL	NULL	NULL	Remove all keep columns transformations with the specified <code>table_name</code> for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all keep columns transformations with the specified <code>table_name</code> , <code>column_list/column_table</code> , and <code>step_number</code> for the specified rule.

## MAINTAIN\_CHANGE\_TABLE Procedure

This procedure configures an Oracle Streams environment that uses change handlers to record in a change table the data manipulation language (DML) changes made to a source table. Optionally, this procedure can also configure one-way replication of the table from the source database to the destination database. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

A change handler is a special type of statement DML handler that tracks table changes and was created by either this MAINTAIN\_CHANGE\_TABLE procedure or the DBMS\_APPLY\_ADM.SET\_CHANGE\_HANDLER procedure. Information about change handlers is stored in the ALL\_APPLY\_CHANGE\_HANDLERS and DBA\_APPLY\_CHANGE\_HANDLERS views.

The change table can reside in the same database as the source table or in a different database.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

---



---

**Note:** The environment configured by this procedure does not record or replicate data definition language (DDL) changes made to the source table.

---



---

**See Also:** ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

### Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_CHANGE_TABLE(
  change_table_name      IN VARCHAR2,
  source_table_name      IN VARCHAR2,
  column_type_list       IN VARCHAR2,
  extra_column_list      IN VARCHAR2  DEFAULT 'command_type, value_type',
  capture_values         IN VARCHAR2,
  options_string         IN VARCHAR2  DEFAULT NULL,
  script_name            IN VARCHAR2  DEFAULT NULL,
  script_directory_object IN VARCHAR2  DEFAULT NULL,
  perform_actions        IN BOOLEAN   DEFAULT TRUE,
  capture_name           IN VARCHAR2  DEFAULT NULL,
  propagation_name      IN VARCHAR2  DEFAULT NULL,
  apply_name             IN VARCHAR2  DEFAULT NULL,
  source_database        IN VARCHAR2  DEFAULT NULL,
  destination_database  IN VARCHAR2  DEFAULT NULL,
  keep_change_columns_only IN BOOLEAN  DEFAULT TRUE,
  execute_lcr            IN BOOLEAN   DEFAULT FALSE);
```

## Parameters

**Table 144–18 MAINTAIN\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
<code>change_table_name</code>	<p>The table that records changes to the source table. This table is maintained by Oracle Streams after configuration.</p> <p>Specify the table as <code>[<i>schema_name</i>.] <i>table_name</i></code>. For example, <code>hr.jobs_change_table</code>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the procedure raises an error.</p> <p>If the specified table exists at the database specified in the <code>destination_database</code> parameter, then the procedure raises an error.</p>
<code>source_table_name</code>	<p>The table at the source database for which changes are recorded.</p> <p>Specify the table as <code>[<i>schema_name</i>.] <i>table_name</i></code>. For example, <code>hr.jobs</code>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the procedure raises an error.</p>
<code>column_type_list</code>	<p>A list of the columns in the source table for which changes are recorded. Specify a comma-delimited list of each column and its data type.</p> <p>For example, specify the following for the <code>hr.jobs</code> table:</p> <pre>job_id VARCHAR2(10), job_title VARCHAR2(35), min_salary NUMBER(6), max_salary NUMBER(6)</pre> <p>The procedure automatically places columns with names that match the source database columns into an unconditional supplemental log group.</p> <p>If NULL, then the procedure raises an error.</p>

**Table 144–18 (Cont.) MAINTAIN\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
extra_column_list	<p>A comma-delimited list of metadata attributes to include in the change table. The column name for a metadata attribute is in the format of attribute name followed by a \$ symbol. For example, the <code>source_database_name</code> attribute is stored in the <code>source_database_name\$</code> column in the change table.</p> <p>The following metadata attributes can be included:</p> <ul style="list-style-type: none"> <li>▪ <code>value_type</code></li> <li>▪ <code>source_database_name</code></li> <li>▪ <code>command_type</code></li> <li>▪ <code>object_owner</code></li> <li>▪ <code>object_name</code></li> <li>▪ <code>tag</code></li> <li>▪ <code>transaction_id</code></li> <li>▪ <code>scn</code></li> <li>▪ <code>commit_scn</code></li> <li>▪ <code>compatible</code></li> <li>▪ <code>instance_number</code></li> <li>▪ <code>message_number</code></li> <li>▪ <code>row_text</code></li> <li>▪ <code>row_id</code></li> <li>▪ <code>serial#</code></li> <li>▪ <code>session#</code></li> <li>▪ <code>source_time</code></li> <li>▪ <code>thread#</code></li> <li>▪ <code>tx_name</code></li> <li>▪ <code>username</code></li> </ul> <p>All of these metadata attributes, except for <code>value_type</code> and <code>message_number</code>, are row LCR attributes that can be stored in row LCRs. For information about LCR attributes, see <i>Oracle Streams Concepts and Administration</i>.</p> <p>The <code>value_type\$</code> column in the change table contains either <code>OLD</code> or <code>NEW</code>, depending on whether the column value is the original column value or the new column value, respectively.</p> <p>The <code>message_number\$</code> column in the change table contains the identification number of each row LCR within a transaction. The message number increases incrementally for each row LCR within a transaction and shows the order of the row LCRs within a transaction.</p> <p>The procedure automatically configures the source database to place information about extra attributes, such as <code>serial#</code>, into the redo log so that the information can be captured and recorded.</p>

**Table 144–18 (Cont.) MAINTAIN\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
capture_values	<p>Specify which values to capture when update operations are performed on the source table:</p> <ul style="list-style-type: none"> <li>▪ old - To capture the original values for an updated column in the source table</li> <li>▪ new - To capture the new values for an updated column in the source table</li> <li>▪ * - To capture both the original and the new values for an updated column in the source table</li> </ul> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> For insert operations, only new column values can be captured. For delete operations, only old column values can be captured.</p>
options_string	<p>String of options passed to the CREATE TABLE statement that creates the change table. The string is appended to the generated CREATE TABLE statement after the closing parenthesis that defines the columns of the table. The string must be syntactically correct.</p>
script_name	<p>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to configure the environment. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file.</p> <p>If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to configure the replication environment directly.</p> <p>If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to configure the replication environment directly and does not generate a script.</p> <p>If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</p>
script_directory_object	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script.</p> <p>If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>
perform_actions	<p>If TRUE, then the procedure performs the necessary actions to configure the environment directly.</p> <p>If FALSE, then the procedure does not perform the necessary actions to configure the environment directly.</p> <p>Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</p> <ul style="list-style-type: none"> <li>▪ script_name</li> <li>▪ script_directory_object</li> </ul>

**Table 144–18 (Cont.) MAINTAIN\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
capture_name	<p>The name of each capture process configured to capture changes. Do not specify an owner.</p> <p>If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database to the positive capture process rule set.</p> <p>If NULL, then the system generates a name for each capture process it creates.</p> <p><b>Note:</b> The capture process name cannot be altered after the capture process is created.</p>
propagation_name	<p>The name of the propagation configured to propagate changes from the source database to the destination database. Do not specify an owner.</p> <p>If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the positive propagation rule set.</p> <p>If NULL, then the system generates a name for the propagation.</p> <p>If non-NULL and the source_database and destination_database are set to the same value, then this procedure raises an error. When the capture process and apply process are in the same database, they can use the same queue, and a propagation is not needed.</p> <p><b>Note:</b> The propagation name cannot be altered after the propagation is created.</p>
apply_name	<p>The name of each apply process configured to apply changes. Do not specify an owner.</p> <p>If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the positive apply process rule set.</p> <p>The specified name must not match the name of an existing messaging client at the destination database.</p> <p>If NULL, then the system generates a name for the apply process. When set to NULL, no apply process that applies changes from the source database can exist on the destination database. If an apply process that applies changes from the source database exists at the destination database, then specify a non-NULL value for this parameter.</p> <p><b>Note:</b> The apply process name cannot be altered after the apply process is created.</p>
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>



**Table 144–18 (Cont.) MAINTAIN\_CHANGE\_TABLE Procedure Parameters**

Parameter	Description
<code>destination_database</code>	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>
<code>keep_change_columns_only</code>	<p>If TRUE, then this procedure adds a declarative rule-based transformation which keeps the list of columns specified in the <code>column_type_list</code> parameter. The columns that are not specified in the <code>column_type_list</code> parameter are removed from each row LCR captured by the capture process.</p> <p>If FALSE, then this procedure does not create a declarative rule-based transformation, and all of the columns in the row LCRs are kept.</p> <p>Specify FALSE when information about columns that are not included in the <code>column_type_list</code> parameter is needed at the destination database. For example, if the <code>execute_lcr</code> parameter is set to TRUE and the configuration will replicate all of the columns in a source table, but the <code>column_type_list</code> parameter includes a subset of these columns, then the <code>keep_change_columns_only</code> parameter should be set to FALSE.</p> <p><b>Note:</b> When this parameter is set to TRUE, a declarative rule-based transformation is always created, even if the <code>column_type_list</code> includes all of the columns in the source table.</p>
<code>execute_lcr</code>	<p>If TRUE, then this procedure creates a change handler that executes each row LCR at the destination database.</p> <p>If FALSE, then the row LCRs are not executed at the destination database.</p>

## Usage Notes

The following are usage notes for this procedure:

### Types of Oracle Streams Environments Configured by the Procedure

This procedure can configure the following types of Oracle Streams environments:

- **Local capture and apply on one database:** Specify the same global name for the `source_database` and the `destination_database` parameter.
- **Local capture and remote apply:** Specify the global name of the local database for the `source_database` parameter and a remote database for the `destination_database` parameter.
- **Downstream capture and local apply:** Specify a remote database for the `source_database` parameter and the local database for the `destination_database` parameter.
- **Downstream capture and remote apply:** Specify a remote database for the `source_database` parameter and a remote database for the `destination_database` parameter.

**Optional One-Way Replication With This Procedure**

To configure one-way replication of the table, in addition to recording changes to the table, set the `execute_lcr` parameter to `TRUE`. The apply process executes each row LCR and applies the change in the row LCR to the replica table at the destination database. In this case, ensure that the source table is instantiated at the destination database before running the procedure. Specifically, the source table must be prepared for instantiation, the instantiation SCN must be set for the replica table at the destination database, and, usually, the source table replica table should be consistent.

**See Also:** *Oracle Streams Replication Administrator's Guide*

**Statement DML Handlers, the Change Table, and Row LCR Execution**

This procedure configures one or more statement DML handlers that perform the following actions:

- Record changes in the change table using the information in row LCRs.
- Execute row LCRs if the `execute_lcr` parameter is set to `TRUE`.

The procedure ensures that the row LCRs contain the required attributes to record the changes specified in the `capture_type_list`, `extra_column_list`, and `capture_values` parameters. The procedure adds the statement DML handlers to the apply process specified in the `apply_name` parameter.

**See Also:** [Chapter 147, "DBMS\\_STREAMS\\_HANDLER\\_ADM"](#)

## MAINTAIN\_GLOBAL Procedure

This procedure configures an Oracle Streams environment that replicates changes at the database level between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

---



---

### Note:

- This procedure automatically excludes database objects that are not supported by Oracle Streams from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
  - If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the destination database while the `MAINTAIN_GLOBAL` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.
  - A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.
- 
- 

**See Also:** ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

### Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_GLOBAL (
  source_directory_object      IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  source_database              IN VARCHAR2,
  destination_database         IN VARCHAR2,
  perform_actions              IN BOOLEAN   DEFAULT TRUE,
  script_name                  IN VARCHAR2   DEFAULT NULL,
  script_directory_object      IN VARCHAR2   DEFAULT NULL,
  dump_file_name               IN VARCHAR2   DEFAULT NULL,
  capture_name                 IN VARCHAR2   DEFAULT NULL,
  capture_queue_table          IN VARCHAR2   DEFAULT NULL,
  capture_queue_name           IN VARCHAR2   DEFAULT NULL,
  capture_queue_user           IN VARCHAR2   DEFAULT NULL,
  propagation_name            IN VARCHAR2   DEFAULT NULL,
  apply_name                   IN VARCHAR2   DEFAULT NULL,
  apply_queue_table            IN VARCHAR2   DEFAULT NULL,
  apply_queue_name             IN VARCHAR2   DEFAULT NULL,
  apply_queue_user             IN VARCHAR2   DEFAULT NULL,
  log_file                     IN VARCHAR2   DEFAULT NULL,
  bi_directional               IN BOOLEAN   DEFAULT FALSE,
  include_ddl                  IN BOOLEAN   DEFAULT FALSE,
```

```
instantiation          IN INTEGER          DEFAULT
                        DBMS_STREAMS_ADM.INSTANTIATION_FULL);
```

## Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters

**Table 144–19** MAINTAIN\_GLOBAL Procedure Parameters

Parameter	Description
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remains in this directory after the procedure completes.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>. In this case, specify <code>NULL</code> for the <code>source_directory_object</code> parameter.</p> <p>If <code>NULL</code> and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>. In these cases, specify <code>NULL</code> for the <code>destination_directory_object</code> parameter.</p> <p>If <code>NULL</code> and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>

**Table 144–19 (Cont.) MAINTAIN\_GLOBAL Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>
destination_database	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
dump_file_name	<p>The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the <code>source_directory_object</code> or <code>destination_directory_object</code> parameter, then the procedure raises an error.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>.</p> <p>If NULL and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL</code>, then the export dump file name is generated by the system. In this case, the export dump file name is <code>expatnn.dmp</code>, where <code>nn</code> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</p>
log_file	<p>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>.</p> <p>If NULL and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL</code>, then the log file name is the same name as the export dump file name with an extension of <code>.clg</code>.</p>

**Table 144–19 (Cont.) MAINTAIN\_GLOBAL Procedure Parameters**

Parameter	Description
instantiation	<p>Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation:</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_FULL</code> performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the <code>instantiation</code> parameter is set to this value, then the user who runs this procedure must have <code>EXECUTE</code> privilege on the <code>DBMS_FILE_TRANSFER</code> package.</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</code> performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the <code>instantiation</code> parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code> does not perform an instantiation. This setting is valid only if the <code>perform_actions</code> parameter is set to <code>FALSE</code>, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly. If you use the <code>RMAN DUPLICATE</code> or <code>CONVERT DATABASE</code> command for database instantiation, then the destination database cannot be the capture database.</p> <p>If this parameter is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK</code>, then the database objects being instantiated must exist at the source database.</p> <p>If an instantiated database object does not exist at the destination database, then it is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported.</p>

## MAINTAIN\_SCHEMAS Procedure

This procedure configures an Oracle Streams environment that replicates changes to specified schemas between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

This procedure is overloaded. One `schema_names` parameter is type `VARCHAR2` and the other `schema_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of schemas in different ways and are mutually exclusive.

---



---

### Note:

- This procedure automatically excludes database objects that are not supported by Oracle Streams in the schemas from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
  - If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the `MAINTAIN_SCHEMAS` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.
- 
- 

**See Also:** ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

## Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS (
  schema_names                IN VARCHAR2,
  source_directory_object     IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  source_database             IN VARCHAR2,
  destination_database        IN VARCHAR2,
  perform_actions             IN BOOLEAN   DEFAULT TRUE,
  script_name                  IN VARCHAR2   DEFAULT NULL,
  script_directory_object     IN VARCHAR2   DEFAULT NULL,
  dump_file_name              IN VARCHAR2   DEFAULT NULL,
  capture_name                 IN VARCHAR2   DEFAULT NULL,
  capture_queue_table         IN VARCHAR2   DEFAULT NULL,
  capture_queue_name          IN VARCHAR2   DEFAULT NULL,
  capture_queue_user          IN VARCHAR2   DEFAULT NULL,
  propagation_name           IN VARCHAR2   DEFAULT NULL,
  apply_name                   IN VARCHAR2   DEFAULT NULL,
  apply_queue_table           IN VARCHAR2   DEFAULT NULL,
  apply_queue_name            IN VARCHAR2   DEFAULT NULL,
  apply_queue_user            IN VARCHAR2   DEFAULT NULL,
  log_file                     IN VARCHAR2   DEFAULT NULL,
```

```

        bi_directional          IN BOOLEAN    DEFAULT FALSE,
        include_ddl            IN BOOLEAN    DEFAULT FALSE,
        instantiation          IN INTEGER     DEFAULT
                                DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA);

DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names              IN DBMS_UTILITY.UNCL_ARRAY,
    source_directory_object   IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database           IN VARCHAR2,
    destination_database      IN VARCHAR2,
    perform_actions          IN BOOLEAN    DEFAULT TRUE,
    script_name               IN VARCHAR2   DEFAULT NULL,
    script_directory_object   IN VARCHAR2   DEFAULT NULL,
    dump_file_name           IN VARCHAR2   DEFAULT NULL,
    capture_name              IN VARCHAR2   DEFAULT NULL,
    capture_queue_table      IN VARCHAR2   DEFAULT NULL,
    capture_queue_name       IN VARCHAR2   DEFAULT NULL,
    capture_queue_user       IN VARCHAR2   DEFAULT NULL,
    propagation_name         IN VARCHAR2   DEFAULT NULL,
    apply_name                IN VARCHAR2   DEFAULT NULL,
    apply_queue_table        IN VARCHAR2   DEFAULT NULL,
    apply_queue_name         IN VARCHAR2   DEFAULT NULL,
    apply_queue_user         IN VARCHAR2   DEFAULT NULL,
    log_file                  IN VARCHAR2   DEFAULT NULL,
    bi_directional          IN BOOLEAN    DEFAULT FALSE,
    include_ddl              IN BOOLEAN    DEFAULT FALSE,
    instantiation            IN INTEGER     DEFAULT
                                DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA);
    
```

## Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters that are not in [Table 144–20](#)

**Table 144–20** *MAINTAIN\_SCHEMAS Procedure Parameters*

Parameter	Description
schema_names	<p>The schemas to be configured for replication and maintained by Oracle Streams after configuration. The schemas can be specified in the following ways:</p> <ul style="list-style-type: none"> <li>■ Comma-delimited list of type VARCHAR2</li> <li>■ A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a schema. The first schema should be in position 1. The last position must be NULL.</li> </ul> <p>This procedure raises an error in any of the following cases:</p> <ul style="list-style-type: none"> <li>■ When a specified schema does not exist at the source database</li> <li>■ When the schema_names parameter is set to NULL</li> </ul>



**Table 144–20 (Cont.) MAINTAIN\_SCHEMAS Procedure Parameters**

Parameter	Description
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remains in this directory after the procedure completes.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>. In this case, specify <code>NULL</code> for the <code>source_directory_object</code> parameter.</p> <p>If <code>NULL</code> and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>. In this case, specify <code>NULL</code> for the <code>destination_directory_object</code> parameter.</p> <p>If <code>NULL</code> and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If <code>NULL</code>, then the procedure uses the global name of the local database.</p>

**Table 144–20 (Cont.) MAINTAIN\_SCHEMAS Procedure Parameters**

Parameter	Description
<code>destination_database</code>	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
<code>dump_file_name</code>	<p>The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the <code>source_directory_object</code> or <code>destination_directory_object</code> parameter, then the procedure raises an error.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>.</p> <p>If NULL and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA</code>, then the export dump file name is generated by the system. In this case, the export dump file name is <code>expatnn.dmp</code>, where <code>nn</code> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</p>
<code>capture_queue_user</code>	<p>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.</p> <p>If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the <code>DBMS_AQADM</code> package.</p>
<code>log_file</code>	<p>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>.</p> <p>If NULL and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA</code>, then the log file name is the same name as the export dump file name with an extension of <code>.clg</code>.</p>

**Table 144–20 (Cont.) MAINTAIN\_SCHEMAS Procedure Parameters**

Parameter	Description
instantiation	<p>Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation:</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA</code> performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for the shared database objects during import. If the <code>instantiation</code> parameter is set to this value, then the user who runs this procedure must have <code>EXECUTE</code> privilege on the <code>DBMS_FILE_TRANSFER</code> package.</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK</code> performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. The instantiation SCN is set for the shared database objects during import. If the <code>instantiation</code> parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code> does not perform an instantiation. This setting is valid only if the <code>perform_actions</code> parameter is set to <code>FALSE</code>, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly.</p> <p>If this parameter is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK</code>, then the database objects being instantiated must exist at the source database, and the tablespaces that contain the schemas must exist at the destination database.</p> <p>If an instantiated database object does not exist at the destination database, then it is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported.</p>

## MAINTAIN\_SIMPLE\_TABLESPACE Procedure

This procedure clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions. Run this procedure at the source database.

---



---

**Note:** This procedure is deprecated. It is replaced by the MAINTAIN\_SIMPLE\_TTS procedure.

---



---

**See Also:**

- ["Deprecated Subprograms"](#) on page 144-4
- [MAINTAIN\\_SIMPLE\\_TTS Procedure](#) on page 144-111

### Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SIMPLE_TABLESPACE (
    tablespace_name          IN VARCHAR2,
    source_directory_object  IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    destination_database     IN VARCHAR2,
    setup_streams            IN BOOLEAN   DEFAULT TRUE,
    script_name              IN VARCHAR2  DEFAULT NULL,
    script_directory_object  IN VARCHAR2  DEFAULT NULL,
    bi_directional           IN BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 144–21 MAINTAIN\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
tablespace_name	<p>The local simple tablespace to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespace must exist at the source database, but it must not exist at the destination database.</p> <p>A directory object must exist for the directory that contains the datafile for the tablespace. The user who invokes this procedure must have READ privilege on this directory object. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p> <p>If NULL, then the procedure raises an error.</p>
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are placed. These files remain in this directory after the procedure completes.</p> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>

**Table 144–21 (Cont.) MAINTAIN\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
destination_database	<p>The global name of the destination database. A database link from the source database to the destination database with the same name as the global name of the destination database must exist.</p> <p>If NULL, then the procedure raises an error.</p>
setup_streams	<p>If TRUE, then the procedure performs the necessary actions to maintain the tablespace directly.</p> <p>If FALSE, then the procedure does not perform the necessary actions to maintain the tablespace directly.</p> <p>Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</p> <ul style="list-style-type: none"> <li>▪ script_name</li> <li>▪ script_directory_object</li> </ul>
script_name	<p>If non-NULL and the setup_streams parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to maintain the specified tablespace. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file.</p> <p>If non-NULL and the setup_streams parameter is TRUE, then this procedure generates the specified script and performs the actions to maintain the specified tablespace directly.</p> <p>If NULL and the setup_streams parameter is TRUE, then this procedure does not generate a script and performs the actions to maintain the specified tablespace directly.</p> <p>If NULL and the setup_streams parameter is FALSE, then the procedure raises an error.</p>
script_directory_object	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script.</p> <p>If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>

**Table 144–21 (Cont.) MAINTAIN\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
bi_directional	<p>Specify <code>TRUE</code> to configure bi-directional replication between the current database and the database specified in <code>destination_database</code>. Both databases are configured as source and destination databases, a capture and apply process is configured at both databases, and propagations are configured between the databases to propagate messages.</p> <p>Specify <code>FALSE</code> to configure one way replication from the current database to the database specified in <code>destination_database</code>. A capture process is configured at the current database, a propagation is configured to propagate messages from the current database to the destination database, and an apply process is configured at the destination database.</p>

## Usage Notes

The specified tablespace must be a simple tablespace. A simple tablespace is a single, self-contained tablespace that uses only one datafile. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. This procedure cannot be used for a non simple tablespace or a set of tablespaces.

### DDL Changes Not Maintained

This procedure does not configure the Oracle Streams environment to maintain DDL changes to the tablespace nor to the database objects in the tablespace. For example, the Oracle Streams environment is not configured to replicate `ALTER TABLESPACE` statements on the tablespace, nor is it configured to replicate `ALTER TABLE` statements on tables in the tablespace. You can configure the Oracle Streams environment to maintain DDL changes manually or modify generated scripts to achieve this.

### Additional Documentation for this Procedure

The following documentation applies to the `MAINTAIN_SIMPLE_TABLESPACE` procedure:

- [Automatic Platform Conversion](#) on page 144-17
- *Oracle Streams Replication Administrator's Guide*

### Requirements for Running this Procedure

Meet the following requirements when run the `MAINTAIN_SIMPLE_TABLESPACE` procedure:

- Run the procedure at the source database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the `destination_database` parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- Grant the user who runs this procedure the `DBA` role. This user must have the necessary privileges to complete the following actions:
  - Create `ANYDATA` queues, capture processes, propagations, and apply processes.

- Specify supplemental logging
- Run subprograms in the DBMS\_STREAMS\_ADM and DBMS\_AQADM packages.
- Access the database specified in the `destination_database` parameter through a database link. This database link should have the same name as the global name of the destination database.
- Run subprograms in the DBMS\_STREAMS\_TABLESPACES\_ADM package
- The necessary privileges to run the `CLONE_SIMPLE_TABLESPACE` procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the source database. See [CLONE\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-18 for the list of required privileges.
- The necessary privileges to run the `ATTACH_SIMPLE_TABLESPACE` procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the destination database. See [ATTACH\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-11 for the list of required privileges.

To ensure that the user who runs this procedure has the necessary privileges, you should configure an Oracle Streams administrator at each database, and each database link should be created in the Oracle Streams administrator's schema.

Typically, the DBA role can be revoked from the user, if necessary, after the configuration is complete.

- If the `bi_directional` parameter is set to `TRUE`, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.
- Each specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes this procedure must have `READ` and `WRITE` privilege on each one.
- The databases configured by this procedure must be Oracle Database 10g Release 2 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10g Release 2 or later database.
  - The `setup_streams` parameter is set to `TRUE` to configure the Oracle Streams replication environment directly.
- The databases configured by this procedure must be Oracle Database 10g Release 1 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10g Release 2 or later database.
  - The `setup_streams` parameter is set to `FALSE` in this procedure, and the replication environment is configured with a generated script.

If the script configures an Oracle Database 10g Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10g Release 2 or later, such as queue-to-queue propagation.

- If the procedure is run at an Oracle Database 10g Release 1 database, then the databases configured by the procedure must be Oracle Database 10g Release 1 or later databases.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

**Default Values for Parameters Excluded From the MAINTAIN\_SIMPLE\_TABLESPACE Procedure**

This procedure uses the default values for the parameters in the MAINTAIN\_TABLESPACES procedure that do not exist in the MAINTAIN\_SIMPLE\_TABLESPACE procedure. For example, this procedure creates a capture process at the source database named `capture`, because that is the default value for the `capture_name` parameter in the MAINTAIN\_TABLESPACES procedure.

**See Also:** [MAINTAIN\\_TABLESPACES Procedure](#) on page 144-119

**Configuration Progress and Recoverability**

When this procedure is run with the `setup_streams` parameter set to `TRUE`, metadata about its configuration actions is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`. If the procedure stops because it encounters an error, then you can use the `RECOVER_OPERATION` procedure to complete the configuration after you correct the conditions that caused the error.

---

---

**Note:** When this procedure is run with the `setup_streams` parameter set to `FALSE`, these views are not populated. Also, the views are not populated when a script generated by this procedure is run.

---

---

**See Also:** ["RECOVER\\_OPERATION Procedure"](#) on page 144-147



## MAINTAIN\_SIMPLE\_TTS Procedure

This procedure clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

---



---

### Note:

- This procedure automatically excludes database objects that are not supported by Oracle Streams in the tablespace from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the DBA\_STREAMS\_UNsupported data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
  - This procedure replaces the deprecated MAINTAIN\_SIMPLE\_TABLESPACE procedure.
- 
- 

**See Also:** ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

### Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_SIMPLE_TTS (
  tablespace_name          IN VARCHAR2,
  source_directory_object  IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  source_database          IN VARCHAR2,
  destination_database     IN VARCHAR2,
  perform_actions          IN BOOLEAN  DEFAULT TRUE,
  script_name              IN VARCHAR2  DEFAULT NULL,
  script_directory_object  IN VARCHAR2  DEFAULT NULL,
  bi_directional           IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters that are not in [Table 144-22](#)

**Table 144–22 MAINTAIN\_SIMPLE\_TTS Procedure Parameters**

Parameter	Description
tablespace_name	<p>The local simple tablespace to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespace must exist at the source database, but it must not exist at the destination database.</p> <p>A directory object must exist for the directory that contains the datafile for the tablespace. The user who invokes this procedure must have <code>READ</code> privilege on this directory object. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p>
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are placed. These files remain in this directory after the procedure completes.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafile for the cloned tablespace are transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If <code>NULL</code>, then the procedure uses the global name of the local database.</p>

**Table 144–22 (Cont.) MAINTAIN\_SIMPLE\_TTS Procedure Parameters**

Parameter	Description
<code>destination_database</code>	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>

## Usage Notes

The specified tablespace must be a simple tablespace. A simple tablespace is a single, self-contained tablespace that uses only one datafile. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. This procedure cannot be used for a non simple tablespace or a set of tablespaces.

### DDL Changes Not Maintained

This procedure does not configure the Oracle Streams environment to maintain DDL changes to the tablespace nor to the database objects in the tablespace. For example, the Oracle Streams environment is not configured to replicate ALTER TABLESPACE statements on the tablespace, nor is it configured to replicate ALTER TABLE statements on tables in the tablespace. You can configure the Oracle Streams environment to maintain DDL changes manually or modify generated scripts to achieve this.

### Additional Privileges Required by the MAINTAIN\_SIMPLE\_TTS Procedure

In addition to the required privileges described in "[Requirements for Running These Procedures](#)" on page 144-18, the user who runs the MAINTAIN\_SIMPLE\_TTS procedure must have the necessary privileges to complete the following actions:

- Run subprograms in the DBMS\_STREAMS\_TABLESPACES\_ADM package
- The necessary privileges to run the CLONE\_SIMPLE\_TABLESPACE procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the source database. See [CLONE\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-18 for the list of required privileges.
- The necessary privileges to run the ATTACH\_SIMPLE\_TABLESPACE procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the destination database. See [ATTACH\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-11 for the list of required privileges.

### Default Values for Parameters Excluded From the MAINTAIN\_SIMPLE\_TTS Procedure

This procedure uses the default values for the parameters in the MAINTAIN\_TTS procedure that do not exist in the MAINTAIN\_SIMPLE\_TTS procedure. For example, this procedure automatically generates the capture process name, because NULL is the default value for the `capture_name` parameter in the MAINTAIN\_TTS procedure, and the procedure generates the capture process name when NULL is specified for `capture_name`.

**See Also:** [MAINTAIN\\_TTS Procedure](#) on page 144-126

## MAINTAIN\_TABLES Procedure

This procedure configures an Oracle Streams environment that replicates changes to specified tables between two databases. This procedure can either configure the environment directly, or it can generate a script that configures the environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

This procedure is overloaded. One `table_names` parameter is type `VARCHAR2` and the other `table_names` parameter is type `DBMS_UTILITY.UNCL_ARRAY`. These parameters enable you to enter the list of tables in different ways and are mutually exclusive.

---



---

**Note:** If the `bi_directional` parameter is set to `TRUE`, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the `MAINTAIN_TABLES` procedure, or the script generated by the procedure, is running. This restriction does not apply to the source database.

---



---

**See Also:** ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

### Syntax

```

DBMS_STREAMS_ADM.MAINTAIN_TABLES (
    table_names                IN VARCHAR2,
    source_directory_object    IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database            IN VARCHAR2,
    destination_database       IN VARCHAR2,
    perform_actions            IN BOOLEAN   DEFAULT TRUE,
    script_name                 IN VARCHAR2   DEFAULT NULL,
    script_directory_object    IN VARCHAR2   DEFAULT NULL,
    dump_file_name             IN VARCHAR2   DEFAULT NULL,
    capture_name                IN VARCHAR2   DEFAULT NULL,
    capture_queue_table        IN VARCHAR2   DEFAULT NULL,
    capture_queue_name         IN VARCHAR2   DEFAULT NULL,
    capture_queue_user         IN VARCHAR2   DEFAULT NULL,
    propagation_name          IN VARCHAR2   DEFAULT NULL,
    apply_name                  IN VARCHAR2   DEFAULT NULL,
    apply_queue_table          IN VARCHAR2   DEFAULT NULL,
    apply_queue_name           IN VARCHAR2   DEFAULT NULL,
    apply_queue_user           IN VARCHAR2   DEFAULT NULL,
    log_file                    IN VARCHAR2   DEFAULT NULL,
    bi_directional              IN BOOLEAN   DEFAULT FALSE,
    include_ddl                 IN BOOLEAN   DEFAULT FALSE,
    instantiation               IN INTEGER    DEFAULT
                                                                    DBMS_STREAMS_ADM.INSTANTIATION_TABLE);

DBMS_STREAMS_ADM.MAINTAIN_TABLES (
    table_names                IN DBMS_UTILITY.UNCL_ARRAY,
    source_directory_object    IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database            IN VARCHAR2,

```

```

destination_database      IN VARCHAR2,
perform_actions           IN BOOLEAN   DEFAULT TRUE,
script_name               IN VARCHAR2   DEFAULT NULL,
script_directory_object   IN VARCHAR2   DEFAULT NULL,
dump_file_name            IN VARCHAR2   DEFAULT NULL,
capture_name              IN VARCHAR2   DEFAULT NULL,
capture_queue_table       IN VARCHAR2   DEFAULT NULL,
capture_queue_name        IN VARCHAR2   DEFAULT NULL,
capture_queue_user        IN VARCHAR2   DEFAULT NULL,
propagation_name          IN VARCHAR2   DEFAULT NULL,
apply_name                IN VARCHAR2   DEFAULT NULL,
apply_queue_table         IN VARCHAR2   DEFAULT NULL,
apply_queue_name          IN VARCHAR2   DEFAULT NULL,
apply_queue_user          IN VARCHAR2   DEFAULT NULL,
log_file                  IN VARCHAR2   DEFAULT NULL,
bi_directional            IN BOOLEAN   DEFAULT FALSE,
include_ddl               IN BOOLEAN   DEFAULT FALSE,
instantiation              IN INTEGER   DEFAULT
                                DBMS_STREAMS_ADM.INSTANTIATION_TABLE);

```

## Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters that are not in [Table 144-23](#)

**Table 144-23** *MAINTAIN\_TABLES Procedure Parameters*

Parameter	Description
table_names	<p>The tables to be configured for replication and maintained by Oracle Streams after configuration. The tables can be specified in the following ways:</p> <ul style="list-style-type: none"> <li>Comma-delimited list of type VARCHAR2</li> <li>A PL/SQL associative array of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table. The first table should be in position 1. The last position must be NULL.</li> </ul> <p>Each table should be specified as [<i>schema_name.</i>] <i>table_name</i>. For example, hr.employees. If the schema is not specified, then the current user is the default.</p> <p>This procedure raises an error in any of the following cases:</p> <ul style="list-style-type: none"> <li>When a specified table does not exist at the source database</li> <li>When the table_names parameter is set to NULL</li> </ul>

**Table 144–23 (Cont.) MAINTAIN\_TABLES Procedure Parameters**

Parameter	Description
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. This file remain in this directory after the procedure completes.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>. In this case, specify <code>NULL</code> for the <code>source_directory_object</code> parameter.</p> <p>If <code>NULL</code> and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>. In this case, specify <code>NULL</code> for the <code>destination_directory_object</code> parameter.</p> <p>If <code>NULL</code> and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If <code>NULL</code>, then the procedure uses the global name of the local database.</p>

**Table 144–23 (Cont.) MAINTAIN\_TABLES Procedure Parameters**

Parameter	Description
<code>destination_database</code>	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
<code>dump_file_name</code>	<p>The name of the Data Pump export dump file. If a file with the specified file name exists in the specified directory for the <code>source_directory_object</code> or <code>destination_directory_object</code> parameter, then the procedure raises an error.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>.</p> <p>If NULL and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE</code>, then the export dump file name is generated by the system. In this case, the export dump file name is <code>expatnn.dmp</code>, where <code>nn</code> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</p>
<code>capture_queue_user</code>	<p>The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.</p> <p>If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.</p>
<code>log_file</code>	<p>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file.</p> <p>This parameter is ignored if <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code>.</p> <p>If NULL and <code>instantiation</code> is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE</code>, then the log file name is the same name as the export dump file name with an extension of <code>.clg</code>.</p>

**Table 144–23 (Cont.) MAINTAIN\_TABLES Procedure Parameters**

Parameter	Description
instantiation	<p>Specify whether to perform instantiation and, if instantiation is performed, the type of instantiation:</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE</code> performs a full Data Pump export at the source database and a Data Pump import of the export dump file at the destination database. If the <code>instantiation</code> parameter is set to this value, then the user who runs this procedure must have <code>EXECUTE</code> privilege on the <code>DBMS_FILE_TRANSFER</code> package.</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK</code> performs a full network Data Pump import. A network import means that Data Pump performs the import without using an export dump file. If the <code>instantiation</code> parameter is set to this value, then a database link from the destination database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p><code>DBMS_STREAMS_ADM.INSTANTIATION_NONE</code> does not perform an instantiation. This setting is valid only if the <code>perform_actions</code> parameter is set to <code>FALSE</code>, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly.</p> <p>If this parameter is set to <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE</code> or <code>DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK</code>, then the tables being instantiated must exist at the source database, and the tablespaces that contain the tables must exist at the destination database.</p> <p>If an instantiated database object does not exist at the destination database, then it is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported.</p> <p>Also, if an instantiated table does not exist at the destination database, then this procedure sets the instantiation SCN for the table. However, if an instantiated table exist at the destination database before instantiation, then this procedure does not set the instantiation SCN for the table. In this case, you must set the instantiation SCN for the table manually after the procedure completes.</p>



## MAINTAIN\_TABLESPACES Procedure

This procedure clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions. Run this procedure at the source database.

---



---

**Note:** This procedure is deprecated. It is replaced by the MAINTAIN\_TTS procedure.

---



---

### See Also:

- ["Deprecated Subprograms"](#) on page 144-4
- [MAINTAIN\\_TTS Procedure](#) on page 144-126

### Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_TABLESPACES (
  tablespace_names           IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
  source_directory_object    IN VARCHAR2,
  destination_directory_object IN VARCHAR2,
  destination_database       IN VARCHAR2,
  setup_streams              IN BOOLEAN   DEFAULT TRUE,
  script_name                 IN VARCHAR2  DEFAULT NULL,
  script_directory_object    IN VARCHAR2  DEFAULT NULL,
  dump_file_name              IN VARCHAR2  DEFAULT NULL,
  source_queue_table         IN VARCHAR2  DEFAULT 'streams_queue_table',
  source_queue_name          IN VARCHAR2  DEFAULT 'streams_queue',
  source_queue_user          IN VARCHAR2  DEFAULT NULL,
  destination_queue_table    IN VARCHAR2  DEFAULT 'streams_queue_table',
  destination_queue_name     IN VARCHAR2  DEFAULT 'streams_queue',
  destination_queue_user     IN VARCHAR2  DEFAULT NULL,
  capture_name               IN VARCHAR2  DEFAULT 'capture',
  propagation_name          IN VARCHAR2  DEFAULT NULL,
  apply_name                  IN VARCHAR2  DEFAULT NULL,
  log_file                    IN VARCHAR2  DEFAULT NULL,
  bi_directional              IN BOOLEAN  DEFAULT FALSE,
  include_ddl                 IN BOOLEAN  DEFAULT FALSE);
```

## Parameters

**Table 144–24** *MAINTAIN\_TABLESPACES Procedure Parameters*

Parameter	Description
tablespace_names	<p>The local tablespace set to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database.</p> <p>A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have <code>READ</code> privilege on these directory objects. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p> <p><b>See Also:</b> <a href="#">TABLESPACE_SET Table Type</a> on page 149-9</p>
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are placed. These files remain in this directory after the procedure completes.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
destination_database	<p>The global name of the destination database. A database link from the source database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If <code>NULL</code>, then the procedure raises an error.</p>

**Table 144–24 (Cont.) MAINTAIN\_TABLESPACES Procedure Parameters**

Parameter	Description
setup_streams	<p>If TRUE, then the procedure performs the necessary actions to maintain the tablespaces directly.</p> <p>If FALSE, then the procedure does not perform the necessary actions to maintain the tablespaces directly.</p> <p>Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</p> <ul style="list-style-type: none"> <li>■ script_name</li> <li>■ script_directory_object</li> </ul>
script_name	<p>If non-NULL and the setup_streams parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to maintain the specified tablespace set. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file.</p> <p>If non-NULL and the setup_streams parameter is TRUE, then this procedure generates the specified script and performs the actions to maintain the specified tablespace directly.</p> <p>If NULL and the setup_streams parameter is TRUE, then this procedure does not generate a script and performs the actions to maintain the specified tablespace set directly.</p> <p>If NULL and the setup_streams parameter is FALSE, then the procedure raises an error.</p>
script_directory_object	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script.</p> <p>If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
dump_file_name	<p>The name of the Data Pump export dump file that contains the specified tablespace set. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error.</p> <p>If NULL, then the export dump file name is generated by the system. In this case, the export dump file name is <i>expatnn.dmp</i>, where <i>nn</i> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</p>

**Table 144–24 (Cont.) MAINTAIN\_TABLESPACES Procedure Parameters**

Parameter	Description
source_queue_table	The name of the queue table for the queue at the source database, specified as [ <i>schema_name</i> .] <i>queue_table_name</i> . For example, <i>strmadmin.streams_queue_table</i> . If the schema is not specified, then the current user is the default.
source_queue_name	The name of the queue at the source database that will function as the ANYDATA queue, specified as [ <i>schema_name</i> .] <i>queue_name</i> . For example, <i>strmadmin.streams_queue</i> .  If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.
source_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the source database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.  If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
destination_queue_table	The name of the queue table for the queue at the destination database, specified as [ <i>schema_name</i> .] <i>queue_table_name</i> . For example, <i>strmadmin.streams_queue_table</i> . If the schema is not specified, then the current user is the default.
destination_queue_name	The name of the queue at the destination database that will function as the ANYDATA queue, specified as [ <i>schema_name</i> .] <i>queue_name</i> . For example, <i>strmadmin.streams_queue</i> .  If the schema is not specified, then the queue table owner is the default. The queue owner automatically has privileges to perform all queue operations on the queue.
destination_queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue at the destination database. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.  If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
capture_name	The name of each capture process configured to capture changes to the database objects in the tablespace set. Do not specify an owner.  If the specified name matches the name of an existing capture process, then the procedure uses the existing capture process and adds the rules for capturing changes to the database objects in the tablespace set to the positive capture process rule set.  <b>Note:</b> The capture process name cannot be altered after the capture process is created.

**Table 144–24 (Cont.) MAINTAIN\_TABLESPACES Procedure Parameters**

Parameter	Description
propagation_name	<p>The name of each propagation configured to propagate changes to the database objects in the tablespace set. Do not specify an owner.</p> <p>If the specified name matches the name of an existing propagation, then the procedure uses the existing propagation and adds the rules for propagating changes to the database objects in the tablespace set to the positive propagation rule set.</p> <p>If NULL, then the system generates a name for each propagation it creates.</p> <p><b>Note:</b> The propagation name cannot be altered after the propagation is created.</p>
apply_name	<p>The name of each apply process configured to apply changes to the database objects in the tablespace set. Do not specify an owner.</p> <p>If the specified name matches the name of an existing apply process, then the procedure uses the existing apply process and adds the rules for applying changes to the database objects in the tablespace set to the positive apply process rule set.</p> <p>The specified name must not match the name of an existing messaging client at the destination database.</p> <p>If NULL, then the system generates a name for each apply process it creates.</p> <p><b>Note:</b> The apply process name cannot be altered after the apply process is created.</p>
log_file	<p>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file.</p> <p>If NULL, then the log file name is the same name as the export dump file name with an extension of <code>.clg</code>.</p>
bi_directional	<p>Specify TRUE to configure bi-directional replication between the current database and the database specified in <code>destination_database</code>. Both databases are configured as source and destination databases, a capture and apply process is configured at both databases, and propagations are configured between the databases to propagate messages.</p> <p>Specify FALSE to configure one way replication from the current database to the database specified in <code>destination_database</code>. A capture process is configured at the current database, a propagation is configured to propagate messages from the current database to the destination database, and an apply process is configured at the destination database.</p>
include_ddl	<p>Specify TRUE to configure an Oracle Streams replication environment that maintains both DML and DDL changes.</p> <p>Specify FALSE to configure an Oracle Streams replication environment that maintains DML changes only. When this parameter is set to FALSE, DDL changes, such as ALTER TABLE, will not be replicated.</p>

## Usage Notes

The specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

**See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

### Additional Documentation for this Procedure

The following documentation applies to the MAINTAIN\_TABLESPACES procedure:

- [Automatic Platform Conversion](#) on page 144-17
- *Oracle Streams Replication Administrator's Guide*

### Requirements for Running this Procedure

Meet the following requirements when run the MAINTAIN\_TABLESPACES procedure:

- Run the procedure at the source database.
- Both databases must be open during configuration. If the procedure is generating a script only, then the database specified in the `destination_database` parameter does not need to be open when you run the procedure, but both databases must be open when you run the generated script.
- The user who runs this procedure should be granted the DBA role. This user must have the necessary privileges to complete the following actions:
  - Create ANYDATA queues, capture processes, propagations, and apply processes.
  - Specify supplemental logging
  - Run subprograms in the DBMS\_STREAMS\_ADM and DBMS\_AQADM packages.
  - Access the database specified in the `destination_database` parameter through a database link. This database link should have the same name as the global name of the destination database.
  - Run subprograms in the DBMS\_STREAMS\_TABLESPACES\_ADM package
  - The necessary privileges to run the CLONE\_TABLESPACES procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the source database. See [CLONE\\_TABLESPACES Procedure](#) on page 149-20 for the list of required privileges.
  - The necessary privileges to run the ATTACH\_TABLESPACES procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the destination database. See [ATTACH\\_TABLESPACES Procedure](#) on page 149-13 for the list of required privileges.

To ensure that the user who runs this procedure has the necessary privileges, you should configure an Oracle Streams administrator at each database, and each database link should be created in the Oracle Streams administrator's schema.

- If the `bi_directional` parameter is set to TRUE, then the corresponding user at the destination database must be able to use a database link to access the source database. This database link should have the same name as the global name of the source database.

- Each specified directory object must be created using the SQL statement `CREATE DIRECTORY`, and the user who invokes this procedure must have `READ` and `WRITE` privilege on each one.
- The databases configured by this procedure must be Oracle Database 10g Release 2 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10g Release 2 or later database.
  - The `setup_streams` parameter is set to `TRUE` to configure the Oracle Streams replication environment directly.
- The databases configured by this procedure must be Oracle Database 10g Release 1 or later databases when this procedure is run under the following conditions:
  - The procedure is run at an Oracle Database 10g Release 2 or later database.
  - The `setup_streams` parameter is set to `FALSE` in this procedure, and the replication environment is configured with a generated script.

If the script configures an Oracle Database 10g Release 1 database, then the script must be modified so that it does not configure features that are available only in Oracle Database 10g Release 2 or later, such as queue-to-queue propagation.

- If the procedure is run at an Oracle Database 10g Release 1 database, then the databases configured by the procedure must be Oracle Database 10g Release 1 or later databases.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

### Configuration Progress and Recoverability

When this procedure is run with the `setup_streams` parameter set to `TRUE`, metadata about its configuration actions is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`. If the procedure stops because it encounters an error, then you can use the `RECOVER_OPERATION` procedure to complete the configuration after you correct the conditions that caused the error.

---

---

**Note:** When this procedure is run with the `setup_streams` parameter set to `FALSE`, these views are not populated. Also, the views are not populated when a script generated by this procedure is run.

---

---

**See Also:** ["RECOVER\\_OPERATION Procedure"](#) on page 144-147

## MAINTAIN\_TTS Procedure

This procedure clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases. This procedure can either perform these actions directly, or it can generate a script that performs these actions.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

---



---

**Note:**

- This procedure automatically excludes database objects that are not supported by Oracle Streams in the tablespaces from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNUNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.
  - This procedure replaces the deprecated `MAINTAIN_TABLESPACES` procedure.
- 
- 

**See Also:** ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure

### Syntax

```
DBMS_STREAMS_ADM.MAINTAIN_TTS(
    tablespace_names          IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
    source_directory_object   IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    source_database           IN VARCHAR2,
    destination_database      IN VARCHAR2,
    perform_actions           IN BOOLEAN   DEFAULT TRUE,
    script_name               IN VARCHAR2   DEFAULT NULL,
    script_directory_object   IN VARCHAR2   DEFAULT NULL,
    dump_file_name            IN VARCHAR2   DEFAULT NULL,
    capture_name              IN VARCHAR2   DEFAULT NULL,
    capture_queue_table       IN VARCHAR2   DEFAULT NULL,
    capture_queue_name        IN VARCHAR2   DEFAULT NULL,
    capture_queue_user        IN VARCHAR2   DEFAULT NULL,
    propagation_name         IN VARCHAR2   DEFAULT NULL,
    apply_name                IN VARCHAR2   DEFAULT NULL,
    apply_queue_table         IN VARCHAR2   DEFAULT NULL,
    apply_queue_name          IN VARCHAR2   DEFAULT NULL,
    apply_queue_user          IN VARCHAR2   DEFAULT NULL,
    log_file                  IN VARCHAR2   DEFAULT NULL,
    bi_directional            IN BOOLEAN   DEFAULT FALSE,
    include_ddl               IN BOOLEAN   DEFAULT FALSE);
```



## Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters that are not in [Table 144-25](#)

**Table 144-25** *MAINTAIN\_TTS Procedure Parameters*

Parameter	Description
tablespace_names	<p>The local tablespace set to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database.</p> <p>A directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects. The directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p> <p>If NULL, then the procedure raises an error.</p> <p><b>See Also:</b> <a href="#">TABLESPACE_SET Table Type</a> on page 149-9</p>
source_directory_object	<p>The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are placed. These files remain in this directory after the procedure completes.</p> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>
destination_directory_object	<p>The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file and the datafiles that comprise the cloned tablespace set are transferred.</p> <p>If the source database and destination database run on the same computer system, then the source and destination directories must be different.</p> <p>If NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle ASM disk group.</p>

**Table 144–25 (Cont.) MAINTAIN\_TTS Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>
destination_database	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
dump_file_name	<p>The name of the Data Pump export dump file that contains the specified tablespace set. If a file with the specified file name exists in the specified directory for the source_directory_object or destination_directory_object parameter, then the procedure raises an error.</p> <p>If NULL, then the export dump file name is generated by the system. In this case, the export dump file name is <code>expatnn.dmp</code>, where <code>nn</code> is a sequence number. The sequence number is increased to produce an export dump file with a unique name in the source directory.</p>
log_file	<p>The name of the Data Pump export log file. This log file is placed in the same directory as the Data Pump export dump file.</p> <p>If NULL, then the log file name is the same name as the export dump file name with an extension of <code>.clg</code>.</p>

## Usage Notes

The specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

**See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

**Additional Privileges Required by the MAINTAIN\_TTS Procedure**

In addition to the required privileges described in "[Requirements for Running These Procedures](#)" on page 144-18, the user who runs the MAINTAIN\_TTS procedure must have the necessary privileges to complete the following actions:

- Run subprograms in the DBMS\_STREAMS\_TABLESPACES\_ADM package
- The necessary privileges to run the CLONE\_TABLESPACES procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the source database. See [CLONE\\_TABLESPACES Procedure](#) on page 149-20 for the list of required privileges.
- The necessary privileges to run the ATTACH\_TABLESPACES procedure in the DBMS\_STREAMS\_TABLESPACES\_ADM package at the destination database. See [ATTACH\\_TABLESPACES Procedure](#) on page 149-13 for the list of required privileges.

## MERGE\_STREAMS Procedure

This procedure merges a stream that is flowing from one capture process with a stream that is flowing from another capture process.

Typically, this procedure is used to merge two streams that were split using the `SPLIT_STREAMS` procedure in this package. The `SPLIT_STREAMS` procedure clones components of the original stream when it splits the streams. Therefore, the information in this section uses the following terminology:

- The stream before it was split off has the original queue, original capture process, and original propagation.
- The stream that was split off by the `SPLIT_STREAMS` procedure has a cloned queue, cloned capture process, and cloned propagation.

This procedure is called by the `MERGE_STREAMS_JOB` procedure. The `MERGE_STREAMS_JOB` procedure determines whether the streams are within a user-specified merge threshold so that the streams can be merged safely. If the streams are not within the merge threshold, then the `MERGE_STREAMS_JOB` procedure does nothing. Typically, it is best to run the `MERGE_STREAMS_JOB` procedure instead of running the `MERGE_STREAMS` procedure directly.

However, you can choose to run the `MERGE_STREAMS` procedure directly when the following conditions are met:

- The problem at the destination of the split stream has been corrected, and the destination queue can accept changes.
- The cloned capture process used by the split stream is started and is capturing changes.
- The apply process at the destination database is applying the changes captured by the cloned capture process.
- The `CAPTURE_MESSAGE_CREATE_TIME` in the `GV$STREAMS_CAPTURE` view of the cloned capture process has caught up to, or nearly caught up to, the `CAPTURE_MESSAGE_CREATE_TIME` of the original capture process. The cloned capture process might never completely catch up to the original capture process. Therefore, you can merge the split stream when the cloned capture process has nearly caught up to the original capture process.

The `MERGE_STREAMS` procedure performs the following actions:

1. Stops the cloned capture process.
2. Stops the original capture process.
3. Copies the cloned propagation back to the original propagation. The propagation has the same name as the original propagation after it is copied back.
4. Starts the original capture process from the lower SCN value of these two SCN values:
  - The acknowledged SCN of the cloned propagation.
  - The lowest acknowledged SCN of the other propagations that propagate changes captured by the original capture process.

When the original capture process is started, it might recapture changes that it already captured, or it might capture changes that were already captured by the cloned capture process. In either case, the relevant apply processes will discard any duplicate changes they receive.

5. Drops the cloned propagation.
6. Drops the cloned capture process.
7. Drops the cloned queue.

**See Also:**

- [MERGE\\_STREAMS\\_JOB Procedure](#) on page 144-133
- [SPLIT\\_STREAMS Procedure](#) on page 144-175

**Syntax**

```
DBMS_STREAMS_ADM.MERGE_STREAMS (
  cloned_propagation_name  IN  VARCHAR2,
  propagation_name        IN  VARCHAR2  DEFAULT NULL,
  queue_name              IN  VARCHAR2  DEFAULT NULL,
  perform_actions         IN  BOOLEAN   DEFAULT TRUE,
  script_name             IN  VARCHAR2  DEFAULT NULL,
  script_directory_object IN  VARCHAR2  DEFAULT NULL);
```

**Parameters****Table 144–26 MERGE\_STREAMS Procedure Parameters**

Parameter	Description
cloned_propagation_name	<p>The name of the cloned propagation used by the stream that was split off from the original stream using the <code>SPLIT_STREAMS</code> procedure. The name of the cloned propagation also identifies the cloned queue and capture process used by the cloned propagation.</p> <p>You must specify an existing propagation name. Do not specify an owner.</p>
propagation_name	<p>The name of the propagation that is merged back to the original stream.</p> <p>If <code>NULL</code>, then the name of the original propagation in the original stream is used. Specify <code>NULL</code> only if the streams were split using the <code>SPLIT_STREAMS</code> procedure.</p> <p>Specify a non-<code>NULL</code> value to use a name that is different from the original propagation name or if you are merging two streams that were not split by the <code>SPLIT_STREAMS</code> procedure. See "Usage Notes" on page 144-132 for more information.</p> <p>If a non-<code>NULL</code> value is specified, then an error is raised under either of the following conditions:</p> <ul style="list-style-type: none"> <li>■ The queue specified in the <code>queue_name</code> parameter does not exist.</li> <li>■ The queue specified in the <code>queue_name</code> parameter exists but is not used by a capture process.</li> </ul>

**Table 144–26 (Cont.) MERGE\_STREAMS Procedure Parameters**

Parameter	Description
queue_name	<p>The name of the queue that is the source queue for the propagation that is merged back.</p> <p>If NULL, then the existing, original queue is the source queue for the propagation that is merged back. Specify NULL only if the streams were split using the SPLIT_STREAMS procedure.</p> <p>Specify a non-NULL value if you are merging two streams that were not split by the SPLIT_STREAMS procedure. Specify the name of the existing queue used by the capture process that will capture changes in the merged stream. See "Usage Notes" on page 144-132 for more information.</p>
perform_actions	<p>If TRUE, then the procedure performs the necessary actions to merge the streams directly.</p> <p>If FALSE, then the procedure does not perform the necessary actions to merge the streams directly.</p> <p>Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</p> <ul style="list-style-type: none"> <li>▪ script_name</li> <li>▪ script_directory_object</li> </ul>
script_name	<p>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to merge the streams. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file.</p> <p>If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to split the stream directly.</p> <p>If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to merge the streams directly and does not generate a script.</p> <p>If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</p>
script_directory_object	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script.</p> <p>If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>

## Usage Notes

You can use the MERGE\_STREAMS procedure to merge two streams that were not split using the SPLIT\_STREAMS procedure. Merging streams in this way can save resources and improve performance when a single database is running two or more capture processes.

The DBA\_STREAMS\_SPLIT\_MERGE view contains information about split and merge operations.

## MERGE\_STREAMS\_JOB Procedure

This procedure determines whether the original capture process and the cloned capture process are within the specified merge threshold. If they are within the merge threshold, then this procedure runs the `MERGE_STREAMS` procedure to merge the two streams.

Typically, this procedure is used to merge two streams that were split using the `SPLIT_STREAMS` procedure in this package. The `SPLIT_STREAMS` procedure clones components of the original stream when it splits the streams. Therefore, the information in this section uses the following terminology:

- The stream before it was split off has the original queue, original capture process, and original propagation.
- The stream that was split off by the `SPLIT_STREAMS` procedure has a cloned queue, cloned capture process, and cloned propagation.

If the `auto_merge_threshold` parameter was set to a positive number in the `SPLIT_STREAMS` procedure that split the streams, then a merge job runs the `MERGE_STREAMS_JOB` procedure automatically according to its schedule. The schedule name is specified for the `schedule_name` parameter, and the merge job name is specified for the `merge_job_name` parameter when the `MERGE_STREAMS_JOB` procedure is run automatically. The merge job and its schedule were created by the `SPLIT_STREAMS` procedure.

If the `auto_merge_threshold` parameter was set to `NULL` or 0 (zero) in the `SPLIT_STREAMS` procedure that split the streams, then you can run the `MERGE_STREAMS_JOB` procedure manually. In this case, it is not run automatically.

### See Also:

- [MERGE\\_STREAMS Procedure](#) on page 144-130
- [SPLIT\\_STREAMS Procedure](#) on page 144-175
- *Oracle Streams Replication Administrator's Guide* for instructions on using the `MERGE_STREAMS_JOB` procedure

## Syntax

```
DBMS_STREAMS_ADM.MERGE_STREAMS_JOB(
  cloned_propagation_name      IN VARCHAR2,
  propagation_name            IN VARCHAR2 DEFAULT NULL,
  queue_name                   IN VARCHAR2 DEFAULT NULL,
  merge_threshold              IN NUMBER,
  schedule_name                IN VARCHAR2 DEFAULT NULL,
  merge_job_name               IN VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 144–27 MERGE\_STREAMS\_JOB Procedure Parameters**

Parameter	Description
cloned_propagation_name	<p>The name of the cloned propagation used by the stream that was split off from the original stream using the <code>SPLIT_STREAMS</code> procedure. The name of the cloned propagation also identifies the cloned queue and capture process used by the cloned propagation.</p> <p>You must specify an existing propagation name. Do not specify an owner.</p>
propagation_name	<p>The name of the propagation that is merged back to the original stream.</p> <p>If <code>NULL</code>, then the name of the original propagation in the original stream is used. Specify <code>NULL</code> only if the streams were split using the <code>SPLIT_STREAMS</code> procedure.</p> <p>Specify a non-<code>NULL</code> value to use a name that is different from the original propagation name or if you are merging two streams that were not split by the <code>SPLIT_STREAMS</code> procedure. See <a href="#">"Usage Notes"</a> on page 144-135 for more information.</p> <p>If a non-<code>NULL</code> value is specified, then an error is raised under either of the following conditions:</p> <ul style="list-style-type: none"> <li>■ The queue specified in the <code>queue_name</code> parameter does not exist.</li> <li>■ The queue specified in the <code>queue_name</code> parameter exists but is not used by a capture process.</li> </ul>
queue_name	<p>The name of the queue that is the source queue for the propagation that is merged back.</p> <p>If <code>NULL</code>, then the existing, original queue is the source queue for the propagation that is merged back. Specify <code>NULL</code> only if the streams were split using the <code>SPLIT_STREAMS</code> procedure.</p> <p>Specify a non-<code>NULL</code> value if you are merging two streams that were not split by the <code>SPLIT_STREAMS</code> procedure. Specify the name of the existing queue used by the capture process that will capture changes in the merged stream. See <a href="#">"Usage Notes"</a> on page 144-135 for more information.</p>
merge_threshold	<p>The merge threshold in seconds.</p> <p>The value of the <code>CAPTURE_MESSAGE_CREATE_TIME</code> column for each capture process in the <code>GV\$STREAMS_CAPTURE</code> dynamic performance view determines whether the streams are merged.</p> <p>Specifically, if the difference, in seconds, between the <code>CAPTURE_MESSAGE_CREATE_TIME</code> of the cloned capture process and the original capture process is less than or equal to the value specified for this parameter, then this procedure runs the <code>MERGE_STREAMS</code> procedure to merge the streams. If the difference is greater than the value specified by this parameter, then this procedure does nothing.</p>
schedule_name	<p>The name of the schedule for the merge job.</p> <p>If <code>NULL</code>, then no schedule name is specified. Typically, you set this parameter to <code>NULL</code> when the <code>auto_merge_threshold</code> parameter was set to <code>NULL</code> or 0 (zero) in the <code>SPLIT_STREAMS</code> procedure that split the streams.</p> <p>Specify <code>NULL</code> if you run this procedure manually.</p>



**Table 144–27 (Cont.) MERGE\_STREAMS\_JOB Procedure Parameters**

Parameter	Description
merge_job_name	<p>The name of the job that merges the streams.</p> <p>If NULL, then no merge job name is specified. Typically, you set this parameter to NULL when the auto_merge_threshold parameter was set to NULL or 0 (zero) in the SPLIT_STREAMS procedure that split the streams.</p> <p>Specify NULL if you run this procedure manually.</p>

## Usage Notes

You can use the MERGE\_STREAMS\_JOB procedure to merge two streams that were not split using the SPLIT\_STREAMS procedure. Merging streams in this way can save resources and improve performance when a single database is running two or more capture processes.

After the MERGE\_STREAMS\_JOB procedure completes, you can query the DBA\_CAPTURE and DBA\_PROPAGATION views to determine whether the streams were merged. If the streams were merged, then the cloned capture process and cloned propagation do not appear in these views.

If the streams were merged and the schedule\_name and merge\_job\_name parameters were non-NULL, then the specified schedule and merge job are deleted automatically.

The DBA\_STREAMS\_SPLIT\_MERGE view contains information about split and merge operations.

## POST\_INSTANTIATION\_SETUP Procedure

This procedure performs the actions required after instantiation to configure an Oracle Streams replication environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

To complete the Oracle Streams replication configuration, follow these steps:

1. Run the `PRE_INSTANTIATION_SETUP` procedure at the source database.
2. Perform any necessary instantiation actions.
3. Run the `POST_INSTANTIATION_SETUP` procedure at the source database.

Typically, the Oracle Streams replication environment configured using these steps serves one of the following purposes:

- Replicates changes to shared database objects to keep the database objects synchronized at different databases.
- Replicates changes to database objects during a database maintenance operation, such migrating a database to a different platform. In this case, use the `CLEANUP_INSTANTIATION_SETUP` procedure to remove the replication environment after the maintenance operation is complete.

---

---

**Attention:** When the `POST_INSTANTIATION_SETUP` procedure is run, the parameter values must match the parameter values specified when the corresponding `PRE_INSTANTIATION_SETUP` procedure was run, except for the values of the following parameters: `perform_actions`, `script_name`, `script_directory_object`, and `start_processes`.

---

---

---

---

**Note:** A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.

---

---

### See Also:

- ["PRE\\_INSTANTIATION\\_SETUP Procedure"](#) on page 144-141
- ["CLEANUP\\_INSTANTIATION\\_SETUP Procedure"](#) on page 144-76
- ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure
- *Oracle Streams Replication Administrator's Guide* for information about setting up an Oracle Streams replication environment
- *Oracle Streams Concepts and Administration* for information about completing database maintenance operations

## Syntax

```
DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP (
    maintain_mode          IN VARCHAR2,
```

tablespace_names	IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,	
source_database	IN VARCHAR2,	
destination_database	IN VARCHAR2,	
perform_actions	IN BOOLEAN	DEFAULT TRUE,
script_name	IN VARCHAR2	DEFAULT NULL,
script_directory_object	IN VARCHAR2	DEFAULT NULL,
capture_name	IN VARCHAR2	DEFAULT NULL,
capture_queue_table	IN VARCHAR2	DEFAULT NULL,
capture_queue_name	IN VARCHAR2	DEFAULT NULL,
capture_queue_user	IN VARCHAR2	DEFAULT NULL,
propagation_name	IN VARCHAR2	DEFAULT NULL,
apply_name	IN VARCHAR2	DEFAULT NULL,
apply_queue_table	IN VARCHAR2	DEFAULT NULL,
apply_queue_name	IN VARCHAR2	DEFAULT NULL,
apply_queue_user	IN VARCHAR2	DEFAULT NULL,
bi_directional	IN BOOLEAN	DEFAULT FALSE,
include_ddl	IN BOOLEAN	DEFAULT FALSE,
start_processes	IN BOOLEAN	DEFAULT FALSE,
instantiation_scn	IN NUMBER	DEFAULT NULL,
exclude_schemas	IN VARCHAR2	DEFAULT NULL,
exclude_flags	IN BINARY_INTEGER	DEFAULT NULL);

## Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters that are not in [Table 144-28](#)

**Table 144-28** *POST\_INSTANTIATION\_SETUP Procedure Parameters*

Parameter	Description
maintain_mode	Specify one of the following: <ul style="list-style-type: none"> <li>GLOBAL to maintain the entire database by configuring replication between the local database and the database specified in the destination_database parameter</li> <li>TRANSPORTABLE TABLESPACES to maintain a set of tablespaces by configuring replication between the local database and the database specified in the destination_database parameter</li> </ul>
tablespace_names	<p>If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database.</p> <p>Also, a directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects.</p> <p>If maintain_mode is set to GLOBAL, then specify an empty tablespace set.</p> <p>Regardless of the maintain_mode setting, an error is raised if the tablespace_names parameter is not set or is set to NULL.</p> <p><b>See Also:</b> <a href="#">TABLESPACE_SET Table Type</a> on page 149-9</p>

**Table 144–28 (Cont.) POST\_INSTANTIATION\_SETUP Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>
destination_database	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
start_processes	<p>If TRUE, then the procedure starts each capture process and apply process. Any disabled capture or apply process created by the PRE_INSTANTIATION_SETUP procedure also is started.</p> <p>If FALSE, then the procedure does not start any capture processes or apply processes.</p>
instantiation_scn	<p>Specify the instantiation SCN for the database objects at the destination database if the instantiation SCN was not set during instantiation. The instantiation SCN is not set automatically during RMAN instantiations, but the correct instantiation SCN value should be determined during an RMAN instantiation. See the <i>Oracle Streams Replication Administrator's Guide</i> for more information.</p> <p>Specify NULL if the instantiation SCN was set for the database objects at the destination database during instantiation. The instantiation SCN can be set during export/import instantiations.</p>
exclude_schemas	<p>A comma-delimited list of schemas to exclude from the Oracle Streams configuration. Schema rules are added to the negative rule sets of each capture process to exclude these schemas.</p> <p>Specify an asterisk (*) to exclude all of the schemas in the database.</p> <p>If NULL, then the procedure does not exclude any schemas in the database.</p> <p>This parameter is valid only if the MAINTAIN_MODE parameter is set to GLOBAL. If the MAINTAIN_MODE parameter is set to TRANSPORTABLE TABLESPACES, then the procedure ignores this parameter.</p>

**Table 144–28 (Cont.) POST\_INSTANTIATION\_SETUP Procedure Parameters**

Parameter	Description
exclude_flags	Specify what is excluded from the replication configuration in the schemas specified by the exclude_schemas parameter. This parameter works the same way in the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures. See "Usage Notes" on page 144-143 for the PRE_INSTANTIATION_SETUP procedure for more information.

## Usage Notes

The following sections contain usage notes for this procedure.

### Self-Contained Tablespace Sets

If the maintain\_mode parameter is set to TRANSPORTABLE TABLESPACES, then the specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

**See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

### Destination Database Renamed During RMAN Database Instantiation

If the maintain\_mode parameter is set to GLOBAL, then database instantiation is required before running the POST\_INSTANTIATION\_SETUP procedure. If the RMAN DUPLICATE or RMAN CONVERT DATABASE command is used for database instantiation, then the global name of the destination database can be renamed to the global name of the source database during instantiation. In this case, before you run the POST\_INSTANTIATION\_SETUP procedure, complete the following steps:

1. Rename the global name of the destination database back to the name specified in the destination\_database parameter.
2. At the destination database, drop and re-create any loopback database links that existed on the source and were cloned on the destination database. For example, suppose the source database dbs1.net has a database link that refers to itself. Suppose the destination database is dbs2.net. At the destination database, drop and re-create this database link as a loopback database link that refers to itself (dbs2.net).
3. At the destination database, drop any database links that were cloned from the source database and are from the source database to the destination database. For example, if the source database is dbs1.net and the destination database is dbs2.net, then drop any database links on the destination database that are from dbs1.net to dbs2.net.
4. Create a database link from the destination database to the source database with the same name as the global name of the source database. The database link must be accessible to the Oracle Streams administrator at the destination database.

This database link is required because the POST\_INSTANTIATION\_SETUP procedure runs the SET\_GLOBAL\_INSTANTIATION\_SCN procedure in the DBMS\_APPLY\_ADM package at the destination database, and the SET\_GLOBAL\_INSTANTIATION\_SCN procedure requires the database link. The instantiation

SCN is set to the value specified in the `instantiation_scn` parameter of the `POST_INSTANTIATION_SETUP` procedure.

---

---

**Note:** When the RMAN `DUPLICATE` or `CONVERT DATABASE` command is used for database instantiation, the destination database cannot be the capture database.

---

---

### Oracle Streams Components Removed From the Destination Database

If the `maintain_mode` parameter is set to `GLOBAL`, then database instantiation is required before running the `POST_INSTANTIATION_SETUP` procedure. During database instantiation, Oracle Streams components created by the `PRE_INSTANTIATION_SETUP` procedure, such as Oracle Streams clients and queues, can be copied from the source database to the destination database. The `POST_INSTANTIATION_SETUP` procedure removes the Stream components created by the `PRE_INSTANTIATION_SETUP` procedure from the destination database.

In some cases, rule sets and rules created by the `PRE_INSTANTIATION_SETUP` procedure might not be removed from the destination database. The `POST_INSTANTIATION_SETUP` procedure does not associate these rule sets and rules with any Stream clients in the destination database. Optionally, you can remove these rule sets and rules from the destination database after the `POST_INSTANTIATION_SETUP` procedure, or the script generated by the procedure, completes.

---

---

**Note:** The `POST_INSTANTIATION_SETUP` procedure only removes Oracle Streams components that were created by the `PRE_INSTANTIATION_SETUP` procedure. It does not remove Oracle Streams components that were created in a different way.

---

---

## PRE\_INSTANTIATION\_SETUP Procedure

This procedure performs the actions required before instantiation to configure an Oracle Streams replication environment.

Run this procedure at the capture database. The capture database is the database that captures changes made to the source database.

To complete the Oracle Streams replication configuration, follow these steps:

1. Run the `PRE_INSTANTIATION_SETUP` procedure at the database that will be the source database in the Stream replication environment.
2. Perform any necessary instantiation actions.
3. Run the `POST_INSTANTIATION_SETUP` procedure at the source database.

Typically, the Oracle Streams replication environment configured using these steps serves one of the following purposes:

- Replicates changes to shared database objects to keep the database objects synchronized at different databases.
- Replicates changes to database objects during a database maintenance operation, such migrating a database to a different platform. In this case, use the `CLEANUP_INSTANTIATION_SETUP` procedure to remove the replication environment after the maintenance operation is complete.

---



---

### Note:

- A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. This procedure does not configure replication for these schemas.
  - When the `RMAN DUPLICATE` or `CONVERT DATABASE` command is used for database instantiation, the destination database cannot be the capture database.
- 
- 

### See Also:

- ["POST\\_INSTANTIATION\\_SETUP Procedure"](#) on page 144-136
- ["CLEANUP\\_INSTANTIATION\\_SETUP Procedure"](#) on page 144-76
- ["Procedures That Configure an Oracle Streams Environment"](#) on page 144-16 for more information about this procedure
- *Oracle Streams Replication Administrator's Guide* for information about setting up an Oracle Streams replication environment
- *Oracle Streams Concepts and Administration* for information about completing database maintenance operations

## Syntax

```
DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
  maintain_mode          IN VARCHAR2,
  tablespace_names      IN DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET,
  source_database        IN VARCHAR2,
  destination_database  IN VARCHAR2,
```

```

perform_actions      IN BOOLEAN          DEFAULT TRUE,
script_name          IN VARCHAR2        DEFAULT NULL,
script_directory_object IN VARCHAR2    DEFAULT NULL,
capture_name         IN VARCHAR2        DEFAULT NULL,
capture_queue_table  IN VARCHAR2        DEFAULT NULL,
capture_queue_name   IN VARCHAR2        DEFAULT NULL,
capture_queue_user   IN VARCHAR2        DEFAULT NULL,
propagation_name     IN VARCHAR2        DEFAULT NULL,
apply_name           IN VARCHAR2        DEFAULT NULL,
apply_queue_table    IN VARCHAR2        DEFAULT NULL,
apply_queue_name     IN VARCHAR2        DEFAULT NULL,
apply_queue_user     IN VARCHAR2        DEFAULT NULL,
bi_directional       IN BOOLEAN          DEFAULT FALSE,
include_ddl          IN BOOLEAN          DEFAULT FALSE,
start_processes      IN BOOLEAN          DEFAULT FALSE,
exclude_schemas     IN VARCHAR2        DEFAULT NULL,
exclude_flags        IN BINARY_INTEGER  DEFAULT NULL);

```

## Parameters

**See Also:** ["Common Parameters for the Configuration Procedures"](#) on page 144-19 for descriptions of the procedure parameters that are not in [Table 144-29](#)

**Table 144-29** PRE\_INSTANTIATION\_SETUP Procedure Parameters

Parameter	Description
maintain_mode	Specify one of the following: <ul style="list-style-type: none"> <li>GLOBAL to maintain the entire database by configuring replication between the local database and the database specified in the destination_database parameter</li> <li>TRANSPORTABLE TABLESPACES to maintain a set of tablespaces by configuring replication between the local database and the database specified in the destination_database parameter</li> </ul>
tablespace_names	<p>If maintain_mode is set to TRANSPORTABLE TABLESPACES, then specify the local tablespace set to be cloned at the destination database and maintained by Oracle Streams.</p> <p>The tablespaces in the tablespace set must exist at the source database, but these tablespaces must not exist at the destination database.</p> <p>Also, a directory object must exist for each directory that contains the datafiles for the tablespace set. The user who invokes this procedure must have READ privilege on these directory objects.</p> <p>If maintain_mode is set to GLOBAL, then specify an empty tablespace set.</p> <p>Regardless of the maintain_mode setting, an error is raised if the tablespace_names parameter is not set or is set to NULL.</p>

**See Also:** [TABLESPACE\\_SET Table Type](#) on page 149-9



**Table 144–29 (Cont.) PRE\_INSTANTIATION\_SETUP Procedure Parameters**

Parameter	Description
source_database	<p>The global name of the source database.</p> <p>If the specified global name is the same as the global name of the local database, then the procedure configures a local capture process for the source database.</p> <p>If the specified global name is different from the global name of the local database, then the procedure configures a downstream capture process at the local database. In this case, a database link from the local database to the source database with the same name as the global name of the source database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure uses the global name of the local database.</p>
destination_database	<p>The global name of the destination database.</p> <p>If the local database is not the destination database, then a database link from the local database to the destination database with the same name as the global name of the destination database must exist and must be accessible to the user who runs the procedure.</p> <p>If NULL, then the procedure raises an error.</p>
capture_queue_table	<p>The name of the queue table for each queue used by a capture process, specified as [<i>schema_name</i>.] <i>queue_table_name</i>. For example, <i>strmadmin.streams_queue_table</i>. If the schema is not specified, then the current user is the default.</p> <p>If NULL, then the system generates a name for the queue table of each queue used by a capture process, and the current user is the owner of each queue table.</p>
start_processes	<p>If TRUE, then the procedure starts each capture process and apply process.</p> <p>If FALSE, then the procedure does not start any capture processes or apply processes.</p>
exclude_schemas	<p>A comma-delimited list of schemas to exclude from the Oracle Streams configuration. Schema rules are added to the negative rule sets of each capture process to exclude these schemas.</p> <p>Specify an asterisk (*) to exclude all of the schemas in the database.</p> <p>If NULL, then the procedure does not exclude any schemas in the database.</p> <p>This parameter is valid only if the <code>MAINTAIN_MODE</code> parameter is set to <code>GLOBAL</code>. If the <code>MAINTAIN_MODE</code> parameter is set to <code>TRANSPORTABLE TABLESPACES</code>, then the procedure ignores this parameter.</p>
exclude_flags	<p>Specify what to exclude from the replication configuration in the schemas specified by the <code>exclude_schemas</code> parameter. See "Usage Notes" on page 144-143 for more information.</p>

## Usage Notes

The following sections contain usage notes for this procedure.

### Self-Contained Tablespace Sets

If the `maintain_mode` parameter is set to `TRANSPORTABLE TABLESPACES`, then the specified set of tablespaces must be self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

**See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespace sets

### The `exclude_flags` Parameter

Specify one of the following values:

- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL` to exclude changes to the schemas and all of the database objects in the schemas
- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED` to exclude changes to the database objects that are not supported by Oracle Streams in the schemas

If both of these values are specified, then the procedure raises an error.

In addition to `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL` or `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED`, specify one or both of the following values:

- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML` to exclude data manipulation language (DML) changes made to the excluded database objects
- `DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL` to exclude data definition language (DDL) changes made to the excluded database objects

Use the plus sign (+) to specify more than one of these values. For example, to maintain DML changes to the tables in a schemas specified by the `exclude_schemas` parameter but exclude DDL changes to these schemas and the database objects in these schemas, specify the following for this parameter:

```
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_FULL +
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL
```

To exclude DML and DDL changes made to unsupported database objects in the schemas specified by the `exclude_schemas` parameter, specify the following for this parameter:

```
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL
```

Rules for the excluded database objects are added to the negative rule set of each capture process. Therefore, changes to the excluded database objects will not be captured and replicated.

This parameter is valid only if the `maintain_mode` parameter is set to `GLOBAL` and the `exclude_schemas` parameter is set to a non-NULL value. If the `maintain_mode` parameter is set to `GLOBAL` and the `exclude_schemas` parameter is set to a NULL, then the procedure ignores this parameter. If the `maintain_mode` parameter is set to `TRANSPORTABLE TABLESPACES`, then this the procedure ignores this parameter and excludes any database objects in the specified tablespace set that are not supported by Oracle Streams from the Oracle Streams configuration automatically.

Also, if schemas are specified in the `exclude_schemas` parameter, but the `exclude_flags` parameter is set to NULL, then the procedure does not add any rules to the

negative rule set of any capture process, and the procedure includes the schemas specified in the `exclude_schemas` parameter in the replication environment.

## PURGE\_SOURCE\_CATALOG Procedure

This procedure removes all Oracle Streams data dictionary information at the local database for the specified object. You can use this procedure to remove Oracle Streams metadata that is not needed currently and will not be needed in the future.

### Syntax

```
DBMS_STREAMS_ADM.PURGE_SOURCE_CATALOG(
  source_database      IN  VARCHAR2,
  source_object_name  IN  VARCHAR2,
  source_object_type  IN  VARCHAR2);
```

### Parameters

**Table 144–30** *PURGE\_SOURCE\_CATALOG Procedure Parameters*

Parameter	Description
source_database	The global name of the source database containing the object.  If you do not include the domain name, then the procedure appends it to the database name automatically. For example, if you specify DBS1 and the domain is .NET, then the procedure specifies DBS1.NET automatically.
source_object_name	The name of the object specified as [ <i>schema_name</i> .] <i>object_name</i> . For example, hr.employees. If the schema is not specified, then the current user is the default.
source_object_type	Type of the object. Currently, TABLE is the only possible object type.

### Usage Notes

The global name of the source database containing the object must be specified for the `source_database` parameter. If the current database is not the source database for the object, then the procedure removes data dictionary information about the object from the current database, not the source database.

For example, suppose changes to the `hr.employees` table at the `dbs1.net` source database are being applied to the `hr.employees` table at the `dbs2.net` destination database. Also, suppose `hr.employees` at `dbs2.net` is not a source at all. In this case, specifying `dbs2.net` as the `source_database` for this table results in an error. However, specifying `dbs1.net` as the `source_database` for this table while running the `PURGE_SOURCE_CATALOG` procedure at the `dbs2.net` database removes data dictionary information about the table at `dbs2.net`.

Do not run this procedure at a database if either of the following conditions are true:

- Logical change records (LCRs) captured by the capture process for the object are or might be applied locally without reinstantiating the object.
- LCRs captured by the capture process for the object are or might be forwarded by the database without reinstantiating the object.

---

**Note:** These conditions do not apply to LCRs that were not created by the capture process. That is, these conditions do not apply to user-created LCRs.

---

## RECOVER\_OPERATION Procedure

This procedure provides options for operations that stopped because they encountered an errors. These operations include split and merge operations, Oracle Streams replication configuration operations, and Oracle Streams change table configuration operations. This procedure either rolls forward the operation, rolls back the operation, or purges all of the metadata about the operation.

This procedure only can perform these actions for the following operations:

- Split and merge operations using:
  - The `split_threshold` and `merge_threshold` capture process parameters set to non-NULL values to enable automatic split and merge
  - [SPLIT\\_STREAMS Procedure](#)
  - [MERGE\\_STREAMS\\_JOB Procedure](#)
- Change table configuration operations performed by the [MAINTAIN\\_CHANGE\\_TABLE Procedure](#)
- Replication configuration operations performed by the following procedures:
  - [MAINTAIN\\_GLOBAL Procedure](#)
  - [MAINTAIN\\_SCHEMAS Procedure](#)
  - [MAINTAIN\\_SIMPLE\\_TABLESPACE Procedure](#)
  - [MAINTAIN\\_SIMPLE\\_TTS Procedure](#)
  - [MAINTAIN\\_TABLES Procedure](#)
  - [MAINTAIN\\_TABLESPACES Procedure](#)
  - [MAINTAIN\\_TTS Procedure](#)
  - [PRE\\_INSTANTIATION\\_SETUP Procedure](#)
  - [POST\\_INSTANTIATION\\_SETUP Procedure](#)

Information about the operation is stored in the following data dictionary views when the operation is in process:

- `DBA_RECOVERABLE_SCRIPT`
- `DBA_RECOVERABLE_SCRIPT_PARAMS`
- `DBA_RECOVERABLE_SCRIPT_BLOCKS`
- `DBA_RECOVERABLE_SCRIPT_ERRORS`

For split and merge operations, the data dictionary views are populated at the database that contains the capture process. For the configuration operations, the data dictionary views are populated at the database where the replication configuration procedure was run.

When the operation completes successfully, metadata about the operation is moved from the `DBA_RECOVERABLE_SCRIPT` view to the `DBA_RECOVERABLE_SCRIPT_HIST` view. The other views, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`, retain information about the operation until it is purged automatically after 30 days.

When one of these operations encounters an error and stops, metadata about the operation remains in these views. In this case, you can either roll forward, roll back, or

purge the metadata about the operation using the RECOVER\_OPERATION procedure. If you choose to roll forward the operation, then correct conditions that caused the errors reported in DBA\_RECOVERABLE\_SCRIPT\_ERRORS before proceeding.

For split and merge operations, run the RECOVER\_OPERATION procedure at the database that contains the capture process. For the configuration operations, run the RECOVER\_OPERATION procedure at the database where the replication configuration procedure was run.

---



---

**Note:**

- Regarding the configuration operations, the procedure must configure the environment directly (`perform_actions => TRUE`), not by generating a script, for information about the operation to be stored in the recoverable views and for the operation to be managed by the RECOVER\_OPERATION procedure.
  - To run the RECOVER\_OPERATION procedure, both databases must be Oracle Database 10g Release 2 or later databases.
- 
- 

**See Also:** *Oracle Streams Replication Administrator's Guide*

## Syntax

```
DBMS_STREAMS_ADM.RECOVER_OPERATION(
    script_id      IN RAW,
    operation_mode IN VARCHAR2 DEFAULT 'FORWARD');
```

## Parameters

**Table 144–31 RECOVER\_OPERATION Procedure Parameters**

Parameter	Description
<code>script_id</code>	The operation id of the operation that is being rolled forward, rolled back, or purged. Query the <code>SCRIPT_ID</code> column of the <code>DBA_RECOVERABLE_SCRIPT</code> data dictionary view to determine the operation id.
<code>operation_mode</code>	<p>If <code>FORWARD</code>, then the procedure rolls forward the operation. Specify <code>FORWARD</code> to try to complete the operation.</p> <p>If <code>ROLLBACK</code>, then the procedure rolls back all of the actions performed in the operation. If the rollback is successful, then this option also moves the metadata about the operation from the <code>DBA_RECOVERABLE_SCRIPT</code> view to the <code>DBA_RECOVERABLE_SCRIPT_HIST</code> view. The other views retain information about the operation for 30 days.</p> <p>If <code>PURGE</code>, then the procedure moves the metadata about the operation from the <code>DBA_RECOVERABLE_SCRIPT</code> view to the <code>DBA_RECOVERABLE_SCRIPT_HIST</code> view without rolling the operation back. The other views retain information about the operation for 30 days.</p>

## REMOVE\_QUEUE Procedure

This procedure removes the specified ANYDATA queue.

Specifically, this procedure performs the following actions:

1. Waits until all current enqueue and dequeue transactions commit.
2. Stops the queue, which means that no further enqueues into the queue or dequeues from the queue are allowed.
3. Drops the queue.
4. If the `drop_unused_queue_table` parameter is set to `TRUE`, then drops the queue table if it is empty and no other queues are using it.
5. If the `cascade` parameter is set to `TRUE`, then drops all of the Oracle Streams clients that are using the queue.

---

**Note:** The specified queue must be a ANYDATA queue.

---

### Syntax

```
DBMS_STREAMS_ADM.REMOVE_QUEUE(
  queue_name          IN  VARCHAR2,
  cascade             IN  BOOLEAN  DEFAULT FALSE,
  drop_unused_queue_table IN  BOOLEAN  DEFAULT TRUE);
```

### Parameters

**Table 144–32 REMOVE\_QUEUE Procedure Parameters**

Parameter	Description
<code>queue_name</code>	The name of the queue to remove, specified as [ <i>schema_name</i> .] <i>queue_name</i> . For example, <code>strmadmin.streams_queue</code> . If the schema is not specified, then the current user is the default.
<code>cascade</code>	If <code>TRUE</code> , then the procedure drops any Oracle Streams clients that use the queue.  If <code>FALSE</code> , then the procedure raises an error if there are any Oracle Streams clients that use the queue. Before you run this procedure with the <code>cascade</code> parameter set to <code>FALSE</code> , make sure no Oracle Streams clients are using the queue currently.
<code>drop_unused_queue_table</code>	If <code>TRUE</code> and the queue table for the queue is empty, then the procedure drops the queue table. The queue table is not dropped if it contains any messages or if it is used by another queue.  If <code>FALSE</code> , then the procedure does not drop the queue table.

## REMOVE\_RULE Procedure

This procedure removes the specified rule or all rules from the rule set associated with the specified capture process, synchronous capture, apply process, propagation, or messaging client.

If this procedure results in an empty positive rule set for a messaging client, then the procedure drops the messaging client automatically.

---



---

**Note:** If a rule was automatically created by the system, and you want to drop the rule, then you should use this procedure to remove the rule instead of the `DBMS_RULE_ADM.DROP_RULE` procedure. If you use the `DBMS_RULE_ADM.DROP_RULE` procedure, then some metadata about the rule might remain.

---



---

### Syntax

```
DBMS_STREAMS_ADM.REMOVE_RULE (
    rule_name          IN  VARCHAR2,
    streams_type       IN  VARCHAR2,
    streams_name       IN  VARCHAR2,
    drop_unused_rule   IN  BOOLEAN  DEFAULT TRUE,
    inclusion_rule     IN  BOOLEAN  DEFAULT TRUE);
```

### Parameters

**Table 144–33 REMOVE\_RULE Procedure Parameters**

Parameter	Description
<code>rule_name</code>	<p>The name of the rule to remove, specified as <code>[ schema_name . ] rule_name</code>. If <code>NULL</code>, then the procedure removes all rules from the specified capture process, synchronous capture, apply process, propagation, or messaging client rule set.</p> <p>For example, to specify a rule in the <code>hr</code> schema named <code>prop_rule1</code>, enter <code>hr.prop_rule1</code>. If the schema is not specified, then the current user is the default.</p>
<code>streams_type</code>	<p>The type of Oracle Streams client:</p> <ul style="list-style-type: none"> <li>■ Specify <code>capture</code> for a capture process.</li> <li>■ Specify <code>sync_capture</code> for a synchronous capture.</li> <li>■ Specify <code>propagation</code> for a propagation.</li> <li>■ Specify <code>apply</code> for an apply process.</li> <li>■ Specify <code>dequeue</code> for a messaging client.</li> </ul>
<code>streams_name</code>	<p>The name of the Oracle Streams client, which can be a capture process, synchronous capture, propagation, apply process, or messaging client. Do not specify an owner.</p> <p>If the specified Oracle Streams client does not exist, but there is metadata in the data dictionary that associates the rule with this client, then the procedure removes the metadata.</p> <p>If the specified Oracle Streams client does not exist, and there is no metadata in the data dictionary that associates the rule with this client, then the procedure raises an error.</p>



**Table 144–33 (Cont.) REMOVE\_RULE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>drop_unused_rule</code>	<p>If <code>TRUE</code> and the rule is not in any rule set, then the procedure drops the rule from the database.</p> <p>If <code>TRUE</code> and the rule exists in any rule set, then the procedure does not drop the rule from the database.</p> <p>If <code>FALSE</code>, then the procedure does not drop the rule from the database.</p>
<code>inclusion_rule</code>	<p>If <code>inclusion_rule</code> is <code>TRUE</code>, then the procedure removes the rule from the positive rule set for the Oracle Streams client.</p> <p>If <code>inclusion_rule</code> is <code>FALSE</code>, then the procedure removes the rule from the negative rule set for the Oracle Streams client.</p>

## REMOVE\_STREAMS\_CONFIGURATION Procedure

This procedure removes the Oracle Streams configuration at the local database.

### Syntax

```
DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION;
```

### Usage Notes

Specifically, this procedure performs the following actions at the local database:

- Drops all capture processes
- If any tables have been prepared for instantiation, then aborts preparation for instantiation for the table using the `ABORT_TABLE_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package
- If any schemas have been prepared for instantiation, then aborts preparation for instantiation for the schema using the `ABORT_SCHEMA_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package
- If the database has been prepared for instantiation, then aborts preparation for instantiation for the database using the `ABORT_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package
- Drops propagations that were created using either the `DBMS_STREAMS_ADM` package or the `DBMS_PROPAGATION_ADM` package. Before a propagation is dropped, its propagation job is disabled. Does not drop propagations that were created using the `DBMS_AQADM` package.
- Disables all propagation jobs used by propagations
- Drops all apply processes. If there are apply errors in the error queue for an apply process, then this procedure deletes these apply errors before it drops the apply process.
- Removes specifications for DDL handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers
- Removes specifications for message handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers
- Removes specifications for precommit handlers used by apply processes, but does not delete the PL/SQL procedures used by these handlers
- Removes the instantiation SCN and ignore SCN for each apply object and schema and for the entire database
- Removes messaging clients
- Unsets message notification specifications that were set using the `SET_MESSAGE_NOTIFICATION` procedure in the `DBMS_STREAMS_ADM` package
- Removes specifications for procedure DML handlers and error handlers, but does not delete the PL/SQL procedures used by these handlers
- Removes update conflict handlers
- Removes specifications for substitute key columns for apply tables
- Drops rule sets and rules that were created using the `DBMS_STREAMS_ADM` package.

- Drops unused rule sets that were used by capture processes, propagations, apply processes, and messaging clients, and removes the rules in these rule sets. These rules and rule sets are removed regardless of whether they were created using the DBMS\_STREAMS\_ADM package or the DBMS\_RULE\_ADM package.

This procedure stops capture processes and apply processes before it drops them.

This procedure does not drop rule sets or rules if they meet both of the following conditions:

- The rule sets or rules were created using the DBMS\_RULE\_ADM package.
- The rule sets or rules were not used by a capture process, propagation, apply process, or messaging client.

---

---

**Attention:** Running this procedure is dangerous. You should run this procedure only if you are sure you want to remove the entire Oracle Streams configuration at a database.

---

---

---

---

**Note:**

- Running this procedure repeatedly does not cause errors. If the procedure fails to complete, then you can run it again.
  - This procedure commits multiple times.
- 
- 

**See Also:**

- [STOP\\_CAPTURE Procedure](#) on page 32-50 in the DBMS\_CAPTURE\_ADM package
- [STOP\\_APPLY Procedure](#) on page 21-78 in the DBMS\_APPLY\_ADM package
- [REMOVE\\_RULE Procedure](#) on page 144-150 in the DBMS\_STREAMS\_ADM package

## RENAME\_COLUMN Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a column in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

---



---

**Note:**

- The `RENAME_COLUMN` procedure supports the same data types supported by Oracle Streams capture processes.
  - Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.
- 
- 

**See Also:** *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations and about the data types supported by Oracle Streams capture processes

### Syntax

```
DBMS_STREAMS_ADM.RENAME_COLUMN(
    rule_name          IN  VARCHAR2,
    table_name         IN  VARCHAR2,
    from_column_name   IN  VARCHAR2,
    to_column_name     IN  VARCHAR2,
    value_type         IN  VARCHAR2  DEFAULT '*',
    step_number        IN  NUMBER     DEFAULT 0,
    operation          IN  VARCHAR2   DEFAULT 'ADD');
```

### Parameters

**Table 144–34** *RENAME\_COLUMN Procedure Parameters*

Parameter	Description
<code>rule_name</code>	The name of the rule, specified as <code>[schema_name.] rule_name</code> . If <code>NULL</code> , then the procedure raises an error.  For example, to specify a rule in the <code>hr</code> schema named <code>employees12</code> , enter <code>hr.employees12</code> . If the schema is not specified, then the current user is the default.
<code>table_name</code>	The name of the table in which the column is renamed in the row LCR, specified as <code>[schema_name.] object_name</code> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default.
<code>from_column_name</code>	The name of the column to be renamed in each row LCR that satisfies the rule.
<code>to_column_name</code>	The new name of the column in each row LCR that satisfies the rule.

**Table 144–34 (Cont.) RENAME\_COLUMN Procedure Parameters**

Parameter	Description
value_type	Specify 'NEW' to rename the column in the new values in the row LCR.  Specify 'OLD' to rename the column in the old values in the row LCR.  Specify '*' to rename the column in both the old and new values in the row LCR.
step_number	The order of execution of the transformation.  <b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about transformation ordering
operation	Specify 'ADD' to add the transformation to the rule.  Specify 'REMOVE' to remove the transformation from the rule.

## Usage Notes

When 'REMOVE' is specified for the `operation` parameter, all of the rename column declarative rule-based transformations for the specified rule are removed that match the specified `table_name`, `column_name`, and `step_number` parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the `RENAME_COLUMN` procedure when one or more of these parameters is NULL:

table_name	from_column_name	to_column_name	step_number	Result
NULL	NULL	NULL	NULL	Remove all rename column transformations for the specified rule.
NULL	NULL	NULL	non-NULL	Remove all rename column transformations with the specified <code>step_number</code> for the specified rule.
NULL	NULL	non-NULL	non-NULL	Remove all rename column transformations with the specified <code>to_column_name</code> and <code>step_number</code> for the specified rule.
NULL	non-NULL	non-NULL	non-NULL	Remove all rename column transformations with the specified <code>table_name</code> and <code>step_number</code> for the specified rule.
NULL	NULL	non-NULL	NULL	Remove all rename column transformations with the specified <code>column_name</code> for the specified rule.
non-NULL	NULL	non-NULL	NULL	Remove all rename column transformations with the specified <code>table_name</code> and <code>column_name</code> for the specified rule.
NULL	non-NULL	NULL	NULL	Remove all rename column transformations with the specified <code>table_name</code> for the specified rule.

<b>table_name</b>	<b>from_column_name</b>	<b>to_column_name</b>	<b>step_number</b>	<b>Result</b>
NULL	non-NULL	non-NULL	NULL	Remove all rename column transformations with the specified <code>table_name</code> , <code>column_name</code> , and <code>step_number</code> for the specified rule.
non-NULL	NULL	non-NULL	NULL	Remove all rename column transformations with the specified <code>table_name</code> , <code>column_name</code> , and <code>step_number</code> for the specified rule.
non-NULL	non-NULL	non-NULL	NULL	Remove all rename column transformations with the specified <code>table_name</code> , <code>column_name</code> , and <code>step_number</code> for the specified rule.
non-NULL	non-NULL	non-NULL	non-NULL	Remove all rename column transformations with the specified <code>table_name</code> , <code>column_name</code> , and <code>step_number</code> for the specified rule.

## RENAME\_SCHEMA Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a schema in a row logical change record (LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

---

**Note:** Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

---

**See Also:** *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

### Syntax

```
DBMS_STREAMS_ADM.RENAME_SCHEMA(
  rule_name          IN  VARCHAR2,
  from_schema_name  IN  VARCHAR2,
  to_schema_name    IN  VARCHAR2,
  step_number       IN  NUMBER    DEFAULT 0,
  operation         IN  VARCHAR2  DEFAULT 'ADD');
```

### Parameters

**Table 144–35** *RENAME\_SCHEMA Procedure Parameters*

Parameter	Description
<code>rule_name</code>	The name of the rule, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . If <code>NULL</code> , then the procedure raises an error.  For example, to specify a rule in the <code>hr</code> schema named <code>employees12</code> , enter <code>hr.employees12</code> . If the schema is not specified, then the current user is the default.
<code>from_schema_name</code>	The name of the schema to be renamed in each row LCR that satisfies the rule.
<code>to_schema_name</code>	The new name of the schema in each row LCR that satisfies the rule.
<code>step_number</code>	The order of execution of the transformation.  <b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about transformation ordering
<code>operation</code>	Specify <code>'ADD'</code> to add the transformation to the rule.  Specify <code>'REMOVE'</code> to remove the transformation from the rule.

### Usage Notes

When `'REMOVE'` is specified for the `operation` parameter, all of the rename schema declarative rule-based transformations for the specified rule are removed that match the specified `from_schema_name`, `to_schema_name`, and `step_number` parameters. Nulls specified for these parameters act as wildcards. The following table

lists the behavior of the RENAME\_SCHEMA procedure when one or more of these parameters is NULL:

<b>from_schema_name</b>	<b>to_schema_name</b>	<b>step_number</b>	<b>Result</b>
NULL	NULL	NULL	Remove all rename schema transformations for the specified rule.
NULL	NULL	non-NULL	Remove all rename schema transformations with the specified step_number for the specified rule.
NULL	non-NULL	non-NULL	Remove all rename schema transformations with the specified to_schema_name and step_number for the specified rule.
non-NULL	NULL	non-NULL	Remove all rename schema transformations with the specified from_schema_name and step_number for the specified rule.
NULL	non-NULL	NULL	Remove all rename schema transformations with the specified to_schema_name for the specified rule.
non-NULL	non-NULL	NULL	Remove all rename schema transformations with the specified from_schema_name and to_schema_name for the specified rule.
non-NULL	NULL	NULL	Remove all rename schema transformations with the specified from_schema_name for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all rename schema transformations with the specified from_schema_name, to_schema_name, and step_number for the specified rule.



## RENAME\_TABLE Procedure

This procedure either adds or removes a declarative rule-based transformation which renames a table in a row logical change record (row LCR) that satisfies the specified rule.

For the transformation to be performed when the specified rule evaluates to `TRUE`, the rule must be in the positive rule set of an Oracle Streams client. Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients.

---

**Note:** Declarative transformations can transform row LCRs only. These row LCRs can be captured by a capture process, captured by a synchronous capture, or constructed and enqueued by an application. Therefore, a DML rule must be specified when you run this procedure. If a DDL is specified, then the procedure raises an error.

---

**See Also:** *Oracle Streams Concepts and Administration* for more information about declarative rule-based transformations

### Syntax

```
DBMS_STREAMS_ADM.RENAME_TABLE(
  rule_name          IN  VARCHAR2,
  from_table_name    IN  VARCHAR2,
  to_table_name      IN  VARCHAR2,
  step_number       IN  NUMBER   DEFAULT 0,
  operation          IN  VARCHAR2 DEFAULT 'ADD');
```

### Parameters

**Table 144–36** *RENAME\_TABLE Procedure Parameters*

Parameter	Description
<code>rule_name</code>	The name of the rule, specified as [ <i>schema_name.</i> ] <i>rule_name</i> . If <code>NULL</code> , then the procedure raises an error.  For example, to specify a rule in the <code>hr</code> schema named <code>employees12</code> , enter <code>hr.employees12</code> . If the schema is not specified, then the current user is the default.
<code>from_table_name</code>	The name of the table to be renamed in each row LCR that satisfies the rule, specified as [ <i>schema_name.</i> ] <i>object_name</i> . For example, <code>hr.employees</code> . If the schema is not specified, then the current user is the default.
<code>to_table_name</code>	The new name of the table in each row LCR that satisfies the rule, specified as [ <i>schema_name.</i> ] <i>object_name</i> . For example, <code>humres.staff</code> .  The transformation can rename the table only, the schema only, or the table and the schema. If the schema is not specified, then the current user is the default.
<code>step_number</code>	The order of execution of the transformation.  <b>See Also:</b> <i>Oracle Streams Concepts and Administration</i> for more information about transformation ordering

**Table 144–36 (Cont.) RENAME\_TABLE Procedure Parameters**

Parameter	Description
operation	Specify 'ADD' to add the transformation to the rule. Specify 'REMOVE' to remove the transformation from the rule.

### Usage Notes

When 'REMOVE' is specified for the operation parameter, all of the rename table declarative rule-based transformations for the specified rule are removed that match the specified from\_table\_name, to\_table\_name, and step\_number parameters. Nulls specified for these parameters act as wildcards. The following table lists the behavior of the RENAME\_TABLE procedure when one or more of these parameters is NULL:

from_table_name	to_table_name	step_number	Result
NULL	NULL	NULL	Remove all rename table transformations for the specified rule.
NULL	NULL	non-NULL	Remove all rename table transformations with the specified step_number for the specified rule.
NULL	non-NULL	non-NULL	Remove all rename table transformations with the specified to_table_name and step_number for the specified rule.
non-NULL	NULL	non-NULL	Remove all rename table transformations with the specified from_table_name and step_number for the specified rule.
NULL	non-NULL	NULL	Remove all rename table transformations with the specified to_table_name for the specified rule.
non-NULL	non-NULL	NULL	Remove all rename table transformations with the specified from_table_name and to_table_name for the specified rule.
non-NULL	NULL	NULL	Remove all rename table transformations with the specified from_table_name for the specified rule.
non-NULL	non-NULL	non-NULL	Remove all rename table transformations with the specified from_table_name, to_table_name, and step_number for the specified rule.

## SET\_MESSAGE\_NOTIFICATION Procedure

This procedure sets a notification for messages that can be dequeued by a specified Oracle Streams messaging client from a specified queue. A notification is sent when a message is enqueued into the specified queue and the specified messaging client can dequeue the message because the message satisfies its rule sets.

---



---

### Note:

- Currently, messaging clients cannot dequeue buffered messages.
  - The DBMS\_AQ package can also configure notifications. The DBMS\_AQ package provides some notification features that are not available in DBMS\_STREAMS\_ADM package, such as buffered message notifications and notification grouping by time.
- 
- 

### Syntax

```
DBMS_STREAMS_ADM.SET_MESSAGE_NOTIFICATION(
  streams_name      IN  VARCHAR2,
  notification_action  IN  VARCHAR2,
  notification_type   IN  VARCHAR2      DEFAULT 'PROCEDURE',
  notification_context IN  ANYDATA      DEFAULT NULL,
  include_notification IN  BOOLEAN      DEFAULT TRUE,
  queue_name         IN  VARCHAR2      DEFAULT 'streams_queue');
```

### Parameters

**Table 144–37 SET\_MESSAGE\_NOTIFICATION Procedure Parameters**

Parameter	Description
streams_name	The name of the Oracle Streams messaging client. Do not specify an owner.  For example, if the user strmadmin is the messaging client, then specify strmadmin.

**Table 144–37 (Cont.) SET\_MESSAGE\_NOTIFICATION Procedure Parameters**

Parameter	Description
notification_action	<p>The action to be performed on message notification. Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ For URL notifications, specify a URL without the prefix <code>http://</code>. For example, to specify the URL <code>http://www.company.com:8080</code>, enter the following: <code>www.company.com:8080</code></li> <li>■ For email notifications, specify an email address. For example, to specify an the email address <code>xyz@company.com</code>, enter the following: <code>xyz@company.com</code></li> <li>■ For PL/SQL procedure notifications, specify an existing user-defined PL/SQL procedure in the form <code>[schema_name.]procedure_name</code>. If the <code>schema_name</code> is not specified, then the user who invokes the SET_MESSAGE_NOTIFICATION procedure is the default. The procedure must be a PLSQLCALLBACK data structure. For example, to specify a procedure named <code>notify_orders</code> in the <code>oe</code> schema, enter the following: <code>oe.notify_orders</code></li> </ul> <p><b>See Also:</b> <a href="#">Examples</a> on page 144-164 for more information about message notification procedures</p>
notification_type	<p>The type of notification. Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ HTTP if you specified a URL for <code>notification_action</code></li> <li>■ MAIL if you specified an email address for <code>notification_action</code></li> <li>■ PROCEDURE if you specified a user-defined procedure for <code>notification_action</code></li> </ul> <p>The type must match the specification for the <code>notification_action</code> parameter.</p>
notification_context	<p>The context of the notification. The context must be specified using RAW data type information. For example, to specify the hexadecimal equivalent of 'FF', enter the following: <code>ANYDATA.ConvertRaw(HEXTORAW('FF'))</code></p> <p>The notification context is passed the PL/SQL procedure in procedure notifications and is not relevant for mail or HTTP notifications.</p>
include_notification	<p>If TRUE, then the procedure adds this notification for the specified <code>streams_name</code> and <code>queue_name</code>. That is, specifying TRUE turns on the notification for the <code>streams_name</code> and <code>queue_name</code>.</p> <p>If FALSE, then the procedure removes this notification for the specified <code>streams_name</code> and <code>queue_name</code>. That is, specifying FALSE turns off the notification for the <code>streams_name</code> and <code>queue_name</code>. If you specify FALSE, then this procedure ignores any specified values for the <code>notification_action</code> or <code>notification_context</code> parameters.</p>

**Table 144-37 (Cont.) SET\_MESSAGE\_NOTIFICATION Procedure Parameters**

Parameter	Description
queue_name	<p>The name of a local ANYDATA queue, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the queue. The specified queue must be a ANYDATA queue.</p> <p>For example, to specify a queue named <i>streams_queue</i> in the <i>strmadmin</i> schema, enter <i>strmadmin.streams_queue</i> for this parameter. If the schema is not specified, then the current user is the default.</p>

## Usage Notes

You can specify one of the following types of notifications:

- An email address to which message notifications are sent. When a relevant message is enqueued into the queue, an email with the message properties is mailed to the specified email address.
- A PL/SQL procedure to be invoked on a notification. When a relevant message is enqueued into the queue, the specified PL/SQL procedure is invoked with the message properties. This PL/SQL procedure can dequeue the message.
- An HTTP URL to which the notification is posted. When a relevant message is enqueued into the queue, a notification with the message properties is posted to the specified URL specified.

A client does not need to be connected to the database to receive a notification.

If you register for email notifications, then you should use the DBMS\_AQELM package to set the host name and port name for the SMTP server that will be used by the database to send email notifications. If required, then you should set the send-from email address, which is set by the database as the *sent from* field. You need a Java-enabled database to use this feature.

If you register for HTTP notifications, you might want to use the DBMS\_AQELM package to set the host name and port number for the proxy server and a list of no-proxy domains that will be used by the database to post HTTP notifications.

Each notification is an AQXMLNotification, which includes of the following:

- *notification\_options*, which includes the following:
  - *destination* - The destination queue from which the message was dequeued
  - *consumer\_name* - The name of the messaging client that dequeued the message
- *message\_set* - The set of message properties

**See Also:**

- The documentation for the DBMS\_AQELM package for more information on email notifications and HTTP notifications
- *Oracle Database 2 Day + Data Replication and Integration Guide* for an example that configures message notification to automatically dequeue of messages of interest
- *Oracle Streams Advanced Queuing User's Guide* and *Oracle XML DB Developer's Guide* for more information about message notifications and XML
- *Oracle Streams Concepts and Administration* for more information about how rules are used in Oracle Streams

**Examples**

If you use a message notification procedure, then this PL/SQL procedure must have the following signature:

```
PROCEDURE procedure_name(
    context IN ANYDATA,
    reginfo IN SYS.AQ$_REG_INFO,
    descr   IN SYS.AQ$_DESCRIPTOR);
```

Here, *procedure\_name* stands for the name of the procedure. The procedure is a PLSQLCALLBACK data structure that specifies the user-defined PL/SQL procedure to be invoked on message notification.

The following is a simple example of a notification procedure that dequeues a message of type `oe.user_msg` using the message identifier and consumer name sent by the notification. To complete the example, first create the type:

```
CREATE TYPE oe.user_msg AS OBJECT(
    object_name  VARCHAR2(30),
    object_owner VARCHAR2(30),
    message      VARCHAR2(50));
/
```

Next, create the procedure:

```
CREATE OR REPLACE PROCEDURE oe.notification_dequeue(
    context ANYDATA,
    reginfo SYS.AQ$_REG_INFO,
    descr   SYS.AQ$_DESCRIPTOR)
AS
    dequeue_options DBMS_AQ.DEQUEUE_OPTIONS_T;
    message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
    message_handle RAW(16);
    message ANYDATA;
    oe_message oe.user_msg;
    rc PLS_INTEGER;
BEGIN
    -- Get the message identifier and consumer name from the descriptor
    dequeue_options.msgid := descr.msg_id;
    dequeue_options.consumer_name := descr.consumer_name;
    -- Dequeue the message
    DBMS_AQ.DEQUEUE(
        queue_name => descr.queue_name,
        dequeue_options => dequeue_options,
        message_properties => message_properties,
```

```
    payload          => message,  
    msgid           => message_handle);  
rc := message.getobject(oe_message);  
COMMIT;  
END;  
/
```

**See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about PLSQLCALLBACK data structures

## SET\_MESSAGE\_TRACKING Procedure

Sets the tracking label for logical change records (LCRs) produced by the current session. This procedure affects only the current session. Any LCRs produced by the current session are tracked, including captured LCRs and persistent LCRs.

---

**Note:** The tracking label set by this procedure does not track non-LCR messages.

---

**See Also:** [GET\\_MESSAGE\\_TRACKING Function](#) on page 144-83

### Syntax

```
DBMS_STREAMS_ADM.SET_MESSAGE_TRACKING (
  tracking_label IN VARCHAR2 DEFAULT 'Streams_tracking',
  actions       IN NUMBER   DEFAULT DBMS_STREAMS_ADM.ACTION_MEMORY);
```

### Parameters

**Table 144–38 SET\_MESSAGE\_TRACKING Procedure Parameters**

Parameter	Description
tracking_label	<p>The label used to track the LCRs produced by the session.</p> <p>Set this parameter to NULL to stop message tracking in the current session.</p> <p>The size limit for a label is 4,000 bytes.</p>
actions	<p>When DBMS_STREAMS_ADM.ACTION_MEMORY is specified, the LCRs are tracked in memory, and the V\$STREAMS_MESSAGE_TRACKING dynamic performance view is populated with information about the LCRs.</p> <p>Currently, DBMS_STREAMS_ADM.ACTION_MEMORY is the only valid setting for this parameter.</p> <p>The value specified for this parameter is an enumerated constant. Enumerated constants must be prefixed with the package name.</p>



## SET\_RULE\_TRANSFORM\_FUNCTION Procedure

This procedure sets or removes the transformation function name for a custom rule-based transformation.

### Syntax

```
DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
  rule_name          IN  VARCHAR2,
  transform_function IN  VARCHAR2);
```

### Parameters

**Table 144–39 SET\_RULE\_TRANSFORM\_FUNCTION Procedure Parameters**

Parameter	Description
<code>rule_name</code>	<p>The name of the rule whose rule-based transformation function you are setting or removing, specified as <code>[schema_name.]rule_name</code>.</p> <p>For example, to specify a rule in the <code>hr</code> schema named <code>prop_rule1</code>, enter <code>hr.prop_rule1</code>. If the schema is not specified, then the current user is the default.</p>
<code>transform_function</code>	<p>Either the name of the transformation function to be used in the rule-based transformation for the rule or <code>NULL</code>.</p> <p>If you specify a transformation function name, then specify an existing function in one of the following forms:</p> <ul style="list-style-type: none"> <li>▪ <code>[schema_name.]function_name</code></li> <li>▪ <code>[schema_name.]package_name.function_name</code></li> </ul> <p>If the function is in a package, then you must specify the <code>package_name</code>. For example, to specify a function in the <code>transform_pkg</code> package in the <code>hr</code> schema named <code>executive_to_management</code>, enter <code>hr.transform_pkg.executive_to_management</code>. An error is returned if the specified procedure does not exist.</p> <p>If the <code>schema_name</code> is not specified, then the user who invokes the rule-based transformation function is the default.</p> <p>If you specify <code>NULL</code>, then the <code>SET_RULE_TRANSFORM_FUNCTION</code> procedure removes the current custom rule-based transformation from the rule.</p>

### Usage Notes

The following sections contain usage notes for this procedure:

- [Transformation Function Signature](#)
- [Rule Action Context](#)
- [User Who Calls the Transformation Function](#)
- [Function Verification](#)

#### Transformation Function Signature

A custom rule-based transformation function always operates on one message, but it can return one message or many messages. A custom rule-based transformation function that returns one message is a one-to-one transformation function. A one-to-one transformation function must have the following signature:

```
FUNCTION user_function (  
    parameter_name IN ANYDATA)  
RETURN ANYDATA;
```

Here, *user\_function* stands for the name of the function and *parameter\_name* stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an ANYDATA encapsulation of a message.

A custom rule-based transformation function that can return multiple messages is a one-to-many transformation function. A one-to-many transformation function must have the following signature:

```
FUNCTION user_function (  
    parameter_name IN ANYDATA)  
RETURN STREAMS$_ANYDATA_ARRAY;
```

Here, *user\_function* stands for the name of the function and *parameter\_name* stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an array that contains zero or more ANYDATA encapsulations of a message. If the array contains zero ANYDATA encapsulations of a message, then the original message is discarded.

The STREAMS\$\_ANYDATA\_ARRAY type is an Oracle-supplied type that has the following definition:

```
CREATE OR REPLACE TYPE SYS.STREAMS$_ANYDATA_ARRAY  
    AS VARRAY(2147483647) of ANYDATA  
/
```

The following restrictions apply to custom rule-based transformations that use one-to-many functions:

- Rules that are associated with one-to-many functions are supported for Oracle Streams capture processes only. These rules must not be added to rule sets used by other Oracle Streams clients, including propagations, apply processes, and messaging clients.
- One-to-many functions only can operate on row logical change records (row LCRs). They cannot operate on DDL LCRs.
- Row LCRs returned by a one-to-many function cannot contain piecewise LOB, LONG, or LONG RAW operations.
- The one-to-many function must return row LCRs in the correct order. The order of row LCRs in the array (starting from index 1) is the order that the row LCRs will be executed in the transaction.

When an apply process dequeues row LCRs that are the result of a transformation by a one-to-many function, the apply process uses the instantiation SCN of the LCR passed to the one-to-many function for all of row LCRs.

**Note:**

- An error is raised if a one-to-one or one-to-many transformation function returns NULL.
- Only one custom rule-based transformation can be specified for a particular rule. You cannot specify both a one-to-one and a one-to-many transformation function for the same rule.
- For any LCR constructed and returned by a custom rule-based transformation, the `source_database_name`, `transaction_id`, and `scn` parameter values must match the values in the original LCR. Oracle automatically specifies the values in the original LCR for these parameters, even if an attempt is made to construct LCRs with different values.

**Rule Action Context**

This procedure modifies the specified rule's action context to specify the transformation. A rule's action context is optional information associated with a rule that is interpreted by the client of the rules engine after the rule evaluates to TRUE for a message. The client of the rules engine can be a user-created application or an internal feature of Oracle, such as Oracle Streams. The Oracle Streams clients include capture processes, synchronous captures, propagations, apply processes, and messaging clients. The information in an action context is an object of type `SYS.RE$NV_LIST`, which consists of a list of name-value pairs.

A custom rule-based transformation in Oracle Streams always consists of the following name-value pair in an action context:

- If the function is a one-to-one transformation function, then the name is `STREAMS$_TRANSFORM_FUNCTION`. If the function is a one-to-many transformation function, then the name is `STREAMS$_ARRAY_TRANS_FUNCTION`.
- The value is a `ANYDATA` instance containing a PL/SQL function name specified as a `VARCHAR2`. This function performs the transformation.

**User Who Calls the Transformation Function**

The user that calls the transformation function must have `EXECUTE` privilege on the function. The following list describes which user calls the transformation function:

- If a transformation is specified for a rule used by a capture process, then the user who calls the transformation function is the capture user for the capture process.
- If a transformation is specified for a rule used by a synchronous capture, then the user who calls the transformation function is the capture user for the synchronous capture.
- If a transformation is specified for a rule used by a propagation, then the user who calls the transformation function is the owner of the source queue for the propagation.
- If a transformation is specified on a rule used by an apply process, then the user who calls the transformation function is the apply user for the apply process.
- If a transformation is specified on a rule used by a messaging client, then the user who calls the transformation function is the user who invokes the messaging client.

**Function Verification**

This procedure does not verify that the specified transformation function exists. If the function does not exist, then an error is raised when an Oracle Streams client tries to invoke the transformation function.

## SET\_TAG Procedure

This procedure sets the binary tag for all redo entries subsequently generated by the current session. Each redo entry generated by DML or DDL statements in the current session will have this tag. This procedure affects only the current session.

### See Also:

- ["GET\\_TAG Function"](#) on page 144-86
- *Oracle Streams Replication Administrator's Guide* for more information about tags

## Syntax

```
DBMS_STREAMS_ADM.SET_TAG(
    tag IN RAW DEFAULT NULL);
```

## Parameters

**Table 144–40 SET\_TAG Procedure Parameters**

Parameter	Description
tag	The binary tag for all subsequent redo entries generated by the current session. A raw value is a sequence of bytes, and a byte is a sequence of bits.  By default, the tag for a session is NULL.  The size limit for a tag value is 2000 bytes.

## Usage Notes

To set the tag to the hexadecimal value of '17' in the current session, run the following procedure:

```
EXEC DBMS_STREAMS_ADM.SET_TAG(tag => HEXTORAW('17'));
```

The following are considerations for the SET\_TAG procedure:

- This procedure is not transactional. That is, the effects of SET\_TAG cannot be rolled back.
- If the SET\_TAG procedure is run to set a non-NULL session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the SET\_TAG procedure in a session. A data dictionary build happens when the DBMS\_CAPTURE\_ADM.BUILD procedure is run. The BUILD procedure can be run automatically when a capture process is created.

**See Also:** [BUILD Procedure](#) on page 32-18

## SET\_UP\_QUEUE Procedure

This procedure creates a queue table and a ANYDATA queue.

### Syntax

```
DBMS_STREAMS_ADM.SET_UP_QUEUE(
  queue_table IN VARCHAR2 DEFAULT 'streams_queue_table',
  storage_clause IN VARCHAR2 DEFAULT NULL,
  queue_name IN VARCHAR2 DEFAULT 'streams_queue',
  queue_user IN VARCHAR2 DEFAULT NULL,
  comment IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 144–41 SET\_UP\_QUEUE Procedure Parameters**

Parameter	Description
queue_table	<p>The name of the queue table specified as [<i>schema_name.</i>] <i>queue_table_name</i>. For example, <i>strmadmin.streams_queue_table</i>. If the schema is not specified, then the current user is the default.</p> <p>If the queue table owner is not specified, then the procedure specifies the user who runs this procedure automatically as the queue table owner.</p> <p>Queue table names can be a maximum of 24 bytes.</p>
storage_clause	<p>The storage clause for queue table</p> <p>The storage parameter is included in the CREATE TABLE statement when the queue table is created. You can specify any valid table storage clause.</p> <p>If a tablespace is not specified here, then the procedure creates the queue table and all its related objects in the default user tablespace of the user who runs this procedure. If a tablespace is specified here, then the procedure creates the queue table and all its related objects in the tablespace specified in the storage clause.</p> <p>If NULL, then the procedure uses the storage characteristics of the tablespace in which the queue table is created.</p> <p><b>See Also:</b> <i>Oracle Database SQL Language Reference</i> for more information about storage clauses</p>
queue_name	<p>The name of the queue that will function as the ANYDATA queue, specified as [<i>schema_name.</i>] <i>queue_name</i>. For example, <i>strmadmin.streams_queue</i>.</p> <p>If the schema is not specified, then the procedure uses the queue table owner. The owner of the queue table must also be the owner of the queue. The queue owner automatically has privileges to perform all queue operations on the queue.</p> <p>If the schema is not specified for this parameter, and the queue table owner is not specified in <i>queue_table</i>, then the current user is the default.</p> <p>Queue names can be a maximum of 24 bytes.</p>

**Table 144–41 (Cont.) SET\_UP\_QUEUE Procedure Parameters**

Parameter	Description
queue_user	The name of the user who requires ENQUEUE and DEQUEUE privileges for the queue. This user also is configured as a secure queue user of the queue. The queue user cannot grant these privileges to other users because they are not granted with the GRANT option.  If NULL, then the procedure does not grant any privileges. You can also grant queue privileges to the appropriate users using the DBMS_AQADM package.
comment	The comment for the queue

## Usage Notes

Set up includes the following actions:

- If the specified queue table does not exist, then this procedure runs the CREATE\_QUEUE\_TABLE procedure in the DBMS\_AQADM package to create the queue table with the specified storage clause. If this procedure creates the queue table, then it creates a multiple consumer ANYDATA queue that is both a secure queue and a transactional queue.

Also, if the database is Oracle Database 10g release 2 or later, the `sort_list` setting in CREATE\_QUEUE\_TABLE is set to `commit_time`. If the database is a release before Oracle Database 10g release 2, the `sort_list` setting in CREATE\_QUEUE\_TABLE is set to `enq_time`.

- If the specified queue table exists, then the queue uses the properties of the existing queue table.
- If the specified queue name does not exist, then this procedure runs the CREATE\_QUEUE procedure in the DBMS\_AQADM package to create the queue.
- This procedure starts the queue.
- If a queue user is specified, then this procedure configures this user as a secure queue user of the queue and grants ENQUEUE and DEQUEUE privileges on the queue to the specified queue user.

To configure the queue user as a secure queue user, this procedure creates an Advanced Queuing agent with the same name as the user name, if one does not exist. If an agent with this name exists and is associated with the queue user only, then it is used. SET\_UP\_QUEUE then runs the ENABLE\_DB\_ACCESS procedure in the DBMS\_AQADM package, specifying the agent and the user.

---



---

### Note:

- To enqueue messages into and dequeue messages from a queue, a queue user must have EXECUTE privilege on the DBMS\_STREAMS\_MESSAGING package or the DBMS\_AQ package. The SET\_UP\_QUEUE procedure does not grant this privilege.
  - If the agent that SET\_UP\_QUEUE tries to create exists and is associated with a user other than the user specified by `queue_user`, then the procedure raises an error. In this case, rename or remove the existing agent, and retry SET\_UP\_QUEUE.
- 
-

**See Also:** *Oracle Streams Concepts and Administration* for more information about secure queue users



## SPLIT\_STREAMS Procedure

This procedure splits one stream flowing from a capture process off from all of the other streams flowing from the capture process.

This procedure is intended for an Oracle Streams replication environment in which a capture process captures changes that are propagated to two or more destination databases. When one destination of a propagation stops accepting the captured changes, the changes remain in the capture process's queue. The queue can grow and begin to spill messages to the hard disk, degrading the performance of the Oracle Streams environment. A destination might stop accepting changes for several reasons. For example, the destination database might be down.

Specifically, this procedure performs the following actions:

1. Creates a new queue at the database running the capture process. The new queue is called the cloned queue because it is a clone of the queue used by the original stream. The new queue will be used by the new, cloned capture process, and it will be the source queue for the new, cloned propagation.
2. Creates a new propagation that propagates messages from the source queue created in Step 1 to the existing destination queue. The new propagation is called the cloned propagation because it is a clone of the propagation used by the original stream. The cloned propagation uses the same rule set as the original propagation.
3. Stops the capture process.
4. Queries the acknowledge SCN for the original propagation. The acknowledged SCN is the last SCN acknowledged by the apply process that applies the changes sent by the propagation.
5. Creates a new capture process. The new capture process is called the cloned capture process because it is a clone of the capture process used by the original stream. The procedure sets the start SCN for the cloned capture process to the value of the acknowledged SCN queried in Step 4. The cloned capture process uses the same rule set as the original capture process.
6. Drops the original propagation.
7. Starts the original capture process with the start SCN set to the acknowledged SCN queried in Step 4.
8. If the `auto_merge_threshold` parameter is set to a positive number, then creates an Oracle Scheduler job to run the `MERGE_STREAMS_JOB` procedure at set intervals according to its schedule. When the two streams are within the specified merge threshold, the `MERGE_STREAMS_JOB` procedure runs the `MERGE_STREAMS` procedure to merge the streams automatically.

After the `SPLIT_STREAMS` procedure has finished running, the cloned capture process is disabled. When the problem at the destination database is solved, and the destination queue can accept changes, you should start the cloned capture process using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

The `DBA_STREAMS_SPLIT_MERGE` view contains the name of each cloned component and information about the split and merge operation.

---



---

**Note:** If the original capture process is a downstream capture process, then you must configure the cloned capture process to read the redo log from the source database before you start the cloned capture process.

---



---

**See Also:**

- ["MERGE\\_STREAMS Procedure"](#) on page 144-130
- ["MERGE\\_STREAMS\\_JOB Procedure"](#) on page 144-133
- *Oracle Streams Replication Administrator's Guide* for instructions on using the SPLIT\_STREAMS procedure

**Syntax**

```
DBMS_STREAMS_ADM.SPLIT_STREAMS (
  propagation_name          IN          VARCHAR2,
  cloned_propagation_name   IN          VARCHAR2  DEFAULT NULL,
  cloned_queue_name         IN          VARCHAR2  DEFAULT NULL,
  cloned_capture_name       IN          VARCHAR2  DEFAULT NULL,
  perform_actions           IN          BOOLEAN   DEFAULT TRUE,
  script_name               IN          VARCHAR2  DEFAULT NULL,
  script_directory_object   IN          VARCHAR2  DEFAULT NULL,
  auto_merge_threshold      IN          NUMBER    DEFAULT NULL,
  schedule_name             IN OUT     VARCHAR2,
  merge_job_name            IN OUT     VARCHAR2);
```

**Parameters****Table 144–42 SPLIT\_STREAMS Procedure Parameters**

Parameter	Description
propagation_name	The name of the propagation that cannot send messages to its destination queue. The specified propagation is the propagation for the stream that is being split off from the other streams. You must specify an existing propagation name. Do not specify an owner.
cloned_propagation_name	The name of the new propagation created by this procedure for the stream that is split off. If NULL, then the system generates a propagation name.
cloned_queue_name	The name of the new queue created by this procedure for the stream that is split off. If NULL, then the system generates a queue name.
cloned_capture_name	The name of the new capture process created by this procedure for the stream that is split off. If NULL, then the system generates a capture process name.

**Table 144-42 (Cont.) SPLIT\_STREAMS Procedure Parameters**

Parameter	Description
perform_actions	<p>If TRUE, then the procedure performs the necessary actions to split the stream directly.</p> <p>If FALSE, then the procedure does not perform the necessary actions to split the stream directly.</p> <p>Specify FALSE when this procedure is generating a script that you can edit and then run. The procedure raises an error if you specify FALSE and either of the following parameters is NULL:</p> <ul style="list-style-type: none"> <li>▪ script_name</li> <li>▪ script_directory_object</li> </ul>
script_name	<p>If non-NULL and the perform_actions parameter is FALSE, then specify the name of the script generated by this procedure. The script contains all of the statements used to split the stream. If a file with the specified script name exists in the specified directory for the script_directory_object parameter, then the procedure appends the statements to the existing file.</p> <p>If non-NULL and the perform_actions parameter is TRUE, then the procedure generates the specified script and performs the actions to split the stream directly.</p> <p>If NULL and the perform_actions parameter is TRUE, then the procedure performs the actions to split the stream directly and does not generate a script.</p> <p>If NULL and the perform_actions parameter is FALSE, then the procedure raises an error.</p>
script_directory_object	<p>The directory object for the directory on the local computer system into which the generated script is placed.</p> <p>If the script_name parameter is NULL, then the procedure ignores this parameter and does not generate a script.</p> <p>If NULL and the script_name parameter is non-NULL, then the procedure raises an error.</p> <p><b>Note:</b> The specified directory object cannot point to an Oracle Automatic Storage Management (ASM) disk group.</p>

**Table 144–42 (Cont.) SPLIT\_STREAMS Procedure Parameters**

Parameter	Description
auto_merge_threshold	<p>If a positive number is specified, then the stream that was split off is automatically merged back into all of the other streams flowing from the capture process by an Oracle Scheduler job. The job runs the MERGE_STREAMS_JOB procedure at set intervals according to its schedule. The value of the CAPTURE_MESSAGE_CREATE_TIME column for each capture process in the GV\$STREAMS_CAPTURE dynamic performance view determines when the streams are merged. Specifically, if the difference, in seconds, between CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process and the original capture process is less than or equal to the value specified for the auto_merge_threshold parameter, then the two streams are merged automatically. The cloned capture process must be started before the split stream can be merged back with the original stream.</p> <p>If NULL or 0 (zero) is specified, then the split stream is not merged back with the original stream automatically. To merge the split stream with the original stream, run the MERGE_STREAM procedure manually when the CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process catches up to, or nearly catches up to, the CAPTURE_MESSAGE_CREATE_TIME of the original capture process.</p> <p>The CAPTURE_MESSAGE_CREATE_TIME records the time when a captured change was recorded in the redo log.</p>
schedule_name	<p>The Oracle Scheduler schedule name, specified as [<i>schema_name</i>.] <i>schedule_name</i>. For example, <i>strmadmin.merge_schedule</i>. If the schema is not specified, then the current user is the default.</p> <p>If auto_merge_threshold is a non-NULL positive number, then the schedule is used by the job that will automatically merge the streams at the appropriate time. You can specify a schedule name to adhere to naming conventions or to track the schedule more easily.</p> <p>If NULL and auto_merge_threshold is a non-NULL positive number, then the system generates a schedule name.</p> <p>If auto_merge_threshold is NULL or 0 (zero), then this parameter must be NULL.</p> <p>If this procedure creates a schedule, the schedule starts when the procedure completes. You can modify the schedule to control how often the merge job is run.</p> <p>If an existing schedule name is specified, an error is raised.</p>
merge_job_name	<p>The Oracle Scheduler job name, specified as [<i>schema_name</i>.] <i>merge_job_name</i>. For example, <i>strmadmin.merge_job</i>. If the schema is not specified, then the current user is the default.</p> <p>If auto_merge_threshold is a non-NULL positive number, then the job will automatically merge the streams at the appropriate time. Specify a merge job name to adhere to naming conventions or to track the job more easily.</p> <p>If NULL and auto_merge_threshold is a non-NULL positive number, then the system generates a job name.</p> <p>If auto_merge_threshold is NULL or 0 (zero), then this parameter must be NULL.</p> <p>If an existing job name is specified, an error is raised.</p>

**See Also:** *Oracle Database Administrator's Guide* for information about Oracle Scheduler



---

---

## DBMS\_STREAMS\_ADVISOR\_ADM

The `DBMS_STREAMS_ADVISOR_ADM` package, one of a set of Oracle Streams packages, provides an interface to gather information about an Oracle Streams environment and advise database administrators based on the information gathered. This package is part of the Oracle Streams Performance Advisor.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS\\_ADVISOR\\_ADM](#)
  - Overview
  - Security Model
  - Constants
  - Views
- [Summary of DBMS\\_STREAMS\\_ADVISOR\\_ADM Subprograms](#)

**See Also:** *Oracle Streams Concepts and Administration* for instructions about using this package

## Using DBMS\_STREAMS\_ADVISOR\_ADM

This section contains topics which relate to using the DBMS\_STREAMS\_ADVISOR\_ADM package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)



## Overview

The DBMS\_STREAMS\_ADVISOR\_ADM package enables you to gather and analyze information about an Oracle Streams environment. You can use this information in the following ways:

- To populate data dictionary views with an Oracle Streams topology that contains information about the Oracle Streams components at one or more databases
- To examine the Oracle Streams components at one or more databases in your environment and the ways in which information flows through streams that include these components
- To analyze the performance of the Oracle Streams components in your environment
- To detect performance problems with Oracle Streams components and correct these problems

**See Also:** *Oracle Streams Concepts and Administration* for instructions about using this package

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

## Constants

The DBMS\_STREAMS\_ADVISOR\_ADM package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS\_DBMS\_ADVISOR\_ADM.CAPTURE\_TYPE.

**Table 145–1 DBMS\_STREAMS\_ADVISOR\_ADM Parameters With Enumerated Constants**

Parameter	Option	Type	Description
component_type	<ul style="list-style-type: none"> <li>■ CAPTURE_TYPE</li> <li>■ PROPAGATION_SENDER_TYPE</li> <li>■ PROPAGATION_RECEIVER_TYPE</li> <li>■ APPLY_TYPE</li> <li>■ QUEUE_TYPE</li> </ul>	NUMBER	<p>CAPTURE_TYPE indicates that the Oracle Streams component is a capture process. The constant number for this option is 1.</p> <p>PROPAGATION_SENDER_TYPE indicates that the Oracle Streams component is a propagation sender. The constant number for this option is 2.</p> <p>PROPAGATION_RECEIVER_TYPE indicates that the Oracle Streams component is a propagation receiver. The constant number for this option is 3.</p> <p>APPLY_TYPE indicates that the Oracle Streams component is an apply process. The constant number for this option is 4.</p> <p>QUEUE_TYPE indicates that the Oracle Streams component is a queue. The constant number for this option is 5.</p>

## Views

The DBMS\_STREAMS\_ADVISOR\_ADM package uses the following views:

- DBA\_STREAMS\_TP\_COMPONENT contains information about each Oracle Streams component at each database.
- DBA\_STREAMS\_TP\_COMPONENT\_LINK contains information about how messages flow between Oracle Streams components.
- DBA\_STREAMS\_TP\_COMPONENT\_STAT contains temporary performance statistics about each Oracle Streams component.
- DBA\_STREAMS\_TP\_DATABASE contains information about each database that contains Oracle Streams components.
- DBA\_STREAMS\_TP\_PATH\_BOTTLENECK contains temporary information about Oracle Streams components that might be slowing down the flow of a stream.
- DBA\_STREAMS\_TP\_PATH\_STAT contains temporary performance statistics about each stream path that exists in the Oracle Streams topology.

The topology information is stored permanently in the following data dictionary views: DBA\_STREAMS\_TP\_DATABASE, DBA\_STREAMS\_TP\_COMPONENT, and DBA\_STREAMS\_TP\_COMPONENT\_LINK.

However, the following views contain temporary information: DBA\_STREAMS\_TP\_COMPONENT\_STAT, DBA\_STREAMS\_TP\_PATH\_BOTTLENECK, and DBA\_STREAMS\_TP\_PATH\_STAT. Some of the data in these views is retained only for the user session that runs the ANALYZE\_CURRENT\_PERFORMANCE procedure. When this user session ends, this temporary information is purged.

**See Also:**

- *Oracle Database Reference*
- *Oracle Streams Concepts and Administration* for sample queries that use these views

## Operational Notes

This section contains the following operational notes for the DBMS\_STREAMS\_ADVISOR\_ADM package:

- [Oracle Streams Components Analyzed by the DBMS\\_STREAMS\\_ADVISOR\\_ADM Package](#)
- [General Steps for Running the Oracle Streams Performance Advisor and Analyzing the Information](#)

### Oracle Streams Components Analyzed by the DBMS\_STREAMS\_ADVISOR\_ADM Package

The DBMS\_STREAMS\_ADVISOR\_ADM analyzes the following Oracle Streams components at the specified databases:

- Capture processes
- Propagations
- Apply processes
- Queues

The DBMS\_STREAMS\_ADVISOR\_ADM package does not analyze the following Oracle Streams components:

- Synchronous captures
- Messaging clients

### General Steps for Running the Oracle Streams Performance Advisor and Analyzing the Information

To use the DBMS\_STREAMS\_ADVISOR\_ADM package, complete the following general steps:

1. Identify the database that you will use to gather the information.

An administrative user at this database must meet the following requirements:

- The user must have access to a database link to each database that contains Oracle Streams components.
- The user must have been granted privileges using the DBMS\_STREAMS\_AUTH.GRANT\_ADMIN\_PRIVILEGE procedure, and each database link must connect to a user at the remote database that has been granted privileges using the DBMS\_STREAMS\_AUTH.GRANT\_ADMIN\_PRIVILEGE procedure.

In an Oracle Streams environment, the Oracle Streams administrator uses this package.

If no database in your environment meets these requirements, then choose a database, configure the necessary database links, and grant the necessary privileges to the users before proceeding.

2. Connect as the administrative user to the database you identified in Step 1, and remain connected to the session while you complete the remaining steps.
3. Run the ANALYZE\_CURRENT\_PERFORMANCE procedure.
4. Optionally, allow messages to flow in the environment for some time.
5. Optionally, rerun the ANALYZE\_CURRENT\_PERFORMANCE procedure one or more times.

6. Query the data dictionary views listed in "[Views](#)" on page 145-6 to analyze the Oracle Streams environment.
7. If you want to update the information in the data dictionary views or if you add new Oracle Streams components to any database in the environment, repeat Steps 2-6.

---

---

**Note:** When you exit the user session, the rate, bandwidth, event, and flow control statistics are purged from the data dictionary views.

---

---

## Summary of DBMS\_STREAMS\_ADVISOR\_ADM Subprograms

**Table 145-2 DBMS\_STREAMS\_ADVISOR\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ANALYZE_CURRENT_PERFORMANCE Procedure</a> on page 145-10	Gathers information about the Oracle Streams components at one or more databases in your environment and analyzes Oracle Streams performance based on the information gathered

## ANALYZE\_CURRENT\_PERFORMANCE Procedure

This procedure gathers information about the Oracle Streams components at one or more databases in your environment and analyzes Oracle Streams performance based on the information gathered.

The performance analyses includes:

- Calculating bottleneck components for each separate stream
- Calculating the throughput of each Oracle Streams component
- Calculating the latency of each Oracle Streams component
- Calculating the top wait event of each Oracle Streams component
- Calculating the message rate of each stream
- Calculating the transaction rate of each stream

The procedure places the gathered information in data dictionary views.

---



---

**Note:** The parameters in this procedure must all be either non-NULL or NULL.

---



---

### See Also:

- *Oracle Streams Concepts and Administration* for instructions on using this procedure
- ["Views"](#) on page 145-6

## Syntax

```
DBMS_STREAMS_ADVISOR_ADM.ANALYZE_CURRENT_PERFORMANCE (
  component_name IN VARCHAR2 DEFAULT NULL,
  component_db   IN VARCHAR2 DEFAULT NULL,
  component_type IN NUMBER   DEFAULT NULL);
```

## Parameters

**Table 145–3 ANALYZE\_CURRENT\_PERFORMANCE Procedure Parameters**

Parameter	Description
component_name	The name of the Oracle Streams component to analyze. For example, to analyze a capture process named <code>capture01</code> , then specify <code>capture01</code> .  If NULL, then all of the Oracle Streams components are analyzed, and the other two parameters must also be NULL.
component_db	The global name of the database that contains the component specified in the <code>component_name</code> parameter. For example, if the <code>db.net</code> database contains the component, then specify <code>db.net</code> .  If NULL, then all of the Oracle Streams components are analyzed, and the other two parameters must also be NULL.



**Table 145-3 (Cont.) ANALYZE\_CURRENT\_PERFORMANCE Procedure Parameters**

Parameter	Description
component_type	<p>The type of the component specified in the component_name parameter. If the component_name parameter is non-NULL, then specify one of the following:</p> <ul style="list-style-type: none"> <li>■ DBMS_STREAMS_ADVISOR_ADM.CAPTURE_TYPE</li> <li>■ DBMS_STREAMS_ADVISOR_ADM.PROPAGATION_SENDER_TYPE</li> <li>■ DBMS_STREAMS_ADVISOR_ADM.PROPAGATION_RECEIVER_TYPE</li> <li>■ DBMS_STREAMS_ADVISOR_ADM.APPLY_TYPE</li> <li>■ DBMS_STREAMS_ADVISOR_ADM.QUEUE_TYPE</li> </ul> <p>See "Constants" on page 145-5 for information about these constants.</p> <p>If NULL, then all of the Oracle Streams components are analyzed, and the other two parameters must also be NULL.</p>



---

---

## DBMS\_STREAMS\_AUTH

The DBMS\_STREAMS\_AUTH package, one of a set of Oracle Streams packages, provides subprograms for granting privileges to Oracle Streams administrators and revoking privileges from Oracle Streams administrators.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS\\_AUTH](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_STREAMS\\_AUTH Subprograms](#)

---

## Using DBMS\_STREAMS\_AUTH

This section contains topics which relate to using the DBMS\_STREAMS\_AUTH package.

- [Overview](#)
- [Security Model](#)

## Overview

This package provides subprograms for granting privileges to Oracle Streams administrators and revoking privileges from Oracle Streams administrators.

**See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Oracle Streams administrators

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting `EXECUTE` on this package to selected users or roles.
- Granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, connect as an administrative user who can create users, grant privileges, and create tablespaces when using this package.

---

## Summary of DBMS\_STREAMS\_AUTH Subprograms

**Table 146–1 DBMS\_STREAMS\_AUTH Package Subprograms**

Subprogram	Description
<a href="#">GRANT_ADMIN_PRIVILEGE Procedure</a> on page 146-6	Either grants the privileges needed by a user to be an Oracle Streams administrator directly, or generates a script that you can use to grant these privileges
<a href="#">GRANT_REMOTE_ADMIN_ACCESS Procedure</a> on page 146-9	Enables a remote Oracle Streams administrator to perform administrative actions at the local database by connecting to the grantee using a database link
<a href="#">REVOKE_ADMIN_PRIVILEGE Procedure</a> on page 146-10	Either revokes Oracle Streams administrator privileges from a user directly, or generates a script that you can use to revoke these privileges
<a href="#">REVOKE_REMOTE_ADMIN_ACCESS Procedure</a> on page 146-12	Disables a remote Oracle Streams administrator from performing administrative actions by connecting to the grantee using a database link

---

**Note:** All subprograms commit unless specified otherwise.

---

## GRANT\_ADMIN\_PRIVILEGE Procedure

This procedure either grants the privileges needed by a user to be an Oracle Streams administrator directly, or generates a script that you can use to grant these privileges.

### Syntax

```
DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE (
  grantee          IN  VARCHAR2,
  grant_privileges IN  BOOLEAN   DEFAULT TRUE,
  file_name        IN  VARCHAR2  DEFAULT NULL,
  directory_name   IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 146–2 GRANT\_ADMIN\_PRIVILEGE Procedure Parameters**

Parameter	Description
grantee	The user to whom privileges are granted
grant_privileges	<p>If TRUE, then the procedure grants the privileges to the specified grantee directly, and adds the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view with YES for both the LOCAL_PRIVILEGES column and the ACCESS_FROM_REMOTE column. If the user has an entry in this data dictionary view, then the procedure does not make another entry, and no error is raised. If TRUE and any of the grant statements fail, then the procedure raises an error.</p> <p>If FALSE, then the procedure does not grant the privileges to the specified grantee directly, and does not add the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view.</p> <p>You specify FALSE when the procedure is generating a file that you will edit and then run. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure raises an error.</p>
file_name	<p>The name of the file generated by the procedure. The file contains all of the statements that grant the privileges. If a file with the specified file name exists in the specified directory name, then the grant statements are appended to the existing file.</p> <p>If NULL, then the procedure does not generate a file.</p>
directory_name	<p>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have WRITE privilege on the directory object.</p> <p>If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file.</p> <p>If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</p>

### Usage Notes

The user who runs the procedure must be an administrative user who can grant privileges to other users.

Specifically, the procedure grants the following privileges to the specified user:

- The RESTRICTED SESSION system privilege



- EXECUTE on the following packages:
  - DBMS\_APPLY\_ADM
  - DBMS\_AQ
  - DBMS\_AQADM
  - DBMS\_AQIN
  - DBMS\_AQELM
  - DBMS\_CAPTURE\_ADM
  - DBMS\_FLASHBACK
  - DBMS\_LOCK
  - DBMS\_PROPAGATION\_ADM
  - DBMS\_RULE\_ADM
  - DBMS\_STREAMS\_ADM
  - DBMS\_STREAMS\_ADVISOR\_ADM
  - DBMS\_STREAMS\_HANDLER\_ADM
  - DBMS\_STREAMS\_MESSAGING
  - DBMS\_TRANSFORM
- Privileges to enqueue messages into and dequeue messages from any queue
- Privileges to manage any queue
- Privileges to create, alter, and execute any of the following types of objects in the user's own schema and in other schemas:
  - Evaluation contexts
  - Rule sets
  - Rules

In addition, the grantee can grant these privileges to other users.

- SELECT\_CATALOG\_ROLE
- SELECT privilege on data dictionary views related to Oracle Streams
- The ability to allow a remote Oracle Streams administrator to perform administrative actions through a database link by connecting to the grantee. This ability is enabled by running the GRANT\_REMOTE\_ADMIN\_ACCESS procedure in this package.

---

---

**Note:**

- To view all of the statements run by the procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.
  - This procedure does not grant any roles to the grantee.
  - This procedure grants only the privileges necessary to configure and administer an Oracle Streams environment. You can grant more privileges to the grantee if necessary.
- 
-

**See Also:**

- [GRANT\\_REMOTE\\_ADMIN\\_ACCESS Procedure](#) on page 146-9
- *Oracle Streams Replication Administrator's Guide* for more information about configuring an Oracle Streams administrator

## GRANT\_REMOTE\_ADMIN\_ACCESS Procedure

This procedure enables a remote Oracle Streams administrator to perform administrative actions at the local database by connecting to the grantee using a database link.

### Syntax

```
DBMS_STREAMS_AUTH.GRANT_REMOTE_ADMIN_ACCESS (
  grantee IN VARCHAR2);
```

### Parameters

**Table 146–3 GRANT\_REMOTE\_ADMIN\_ACCESS Procedure Parameter**

Parameter	Description
grantee	The user who allows remote access. The procedure adds the grantee to the DBA_STREAMS_ADMINISTRATOR data dictionary view with YES for the ACCESS_FROM_REMOTE column. If the user has an entry in this data dictionary view, then the procedure does not make another entry. Instead, it updates the ACCESS_FROM_REMOTE column to YES.

### Usage Notes

Typically, you run the procedure and specify a grantee at a local source database if a downstream capture process captures changes originating at the local source database. The Oracle Streams administrator at a downstream capture database administers the source database using this connection. You can also run the procedure at a database running an apply process so that a remote Oracle Streams administrator can set instantiation SCNs at the local database.

---

**Note:** The GRANT\_ADMIN\_PRIVILEGE procedure runs this procedure.

---

**See Also:** [GRANT\\_ADMIN\\_PRIVILEGE Procedure](#) on page 146-6

## REVOKE\_ADMIN\_PRIVILEGE Procedure

This procedure either revokes Oracle Streams administrator privileges from a user directly, or generates a script that you can use to revoke these privileges.

### Syntax

```
DBMS_STREAMS_AUTH.REVOKE_ADMIN_PRIVILEGE(
  grantee          IN VARCHAR2,
  revoke_privileges IN BOOLEAN   DEFAULT TRUE,
  file_name       IN VARCHAR2   DEFAULT NULL,
  directory_name  IN VARCHAR2   DEFAULT NULL);
```

### Parameters

**Table 146–4 REVOKE\_ADMIN\_PRIVILEGE Procedure Parameters**

Parameter	Description
grantee	The user from whom privileges are revoked
revoke_privileges	<p>If TRUE, then the procedure revokes the privileges from the specified user directly, and removes the user from the DBA_STREAMS_ADMINISTRATOR data dictionary view. If the user does not have a record in this data dictionary view, then the procedure does not remove a record from the view, and no error is raised. If TRUE and any of the revoke statements fail, then the procedure raises an error. A revoke statement will fail if the user is not granted the privilege that is being revoked.</p> <p>If FALSE, then the procedure does not revoke the privileges to the specified user directly, and does not remove the user from the DBA_STREAMS_ADMINISTRATOR data dictionary view.</p> <p>You specify FALSE when the procedure is generating a file that you will edit and then run. If you specify FALSE and either the file_name or directory_name parameter is NULL, then the procedure does not raise an error.</p>
file_name	<p>The name of the file generated by this procedure. The file contains all of the statements that revoke the privileges. If a file with the specified file name exists in the specified directory name, then the revoke statements are appended to the existing file.</p> <p>If NULL, then the procedure does not generate a file.</p>
directory_name	<p>The directory into which the generated file is placed. The specified directory must be a directory object created using the SQL statement CREATE DIRECTORY. If you specify a directory, then the user who invokes the procedure must have WRITE privilege on the directory object.</p> <p>If the file_name parameter is NULL, then this parameter is ignored, and the procedure does not generate a file.</p> <p>If NULL and the file_name parameter is non-NULL, then the procedure raises an error.</p>

### Usage Notes

The user who runs this procedure must be an administrative user who can revoke privileges from other users. Specifically, this procedure revokes the privileges granted by running the GRANT\_ADMIN\_PRIVILEGE procedure in this package.

---

---

**Note:** To view all of the statements run by this procedure in detail, you can use the procedure to generate a script and then view the script in a text editor.

---

---

**See Also:** [GRANT\\_ADMIN\\_PRIVILEGE Procedure](#) on page 146-6

## REVOKE\_REMOTE\_ADMIN\_ACCESS Procedure

This procedure disables a remote Oracle Streams administrator from performing administrative actions by connecting to the grantee using a database link.

---

**Note:** The REVOKE\_ADMIN\_PRIVILEGE procedure runs this procedure.

---

**See Also:** [REVOKE\\_ADMIN\\_PRIVILEGE Procedure](#) on page 146-10

### Syntax

```
DBMS_STREAMS_AUTH.REVOKE_REMOTE_ADMIN_ACCESS (
  grantee IN VARCHAR2);
```

### Parameters

**Table 146-5 REVOKE\_REMOTE\_ADMIN\_ACCESS Procedure Parameter**

Parameter	Description
grantee	<p>The user for whom access from a remote Oracle Streams administrator is disabled.</p> <p>If a row for the grantee exists in the DBA_STREAMS_ADMINISTRATOR data dictionary view, then the procedure updates the ACCESS_FROM_REMOTE column for the grantee to NO. If, after this update, both the LOCAL_PRIVILEGES column and the ACCESS_FROM_REMOTE column are NO for the grantee, then the procedure removes the grantee from the view.</p> <p>If no row for the grantee exists in the DBA_STREAMS_ADMINISTRATOR data dictionary view, then the procedure does not update the view and does not raise an error.</p>

---

---

## DBMS\_STREAMS\_HANDLER\_ADM

The DBMS\_STREAMS\_HANDLER\_ADM package, one of a set of Oracle Streams packages, provides interfaces to manage statement DML handlers.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS\\_HANDLER\\_ADM](#)
  - Overview
  - Security Model
  - Views
  - Operational Notes
- [Summary of DBMS\\_STREAMS\\_HANDLER\\_ADM Subprograms](#)

## Using DBMS\_STREAMS\_HANDLER\_ADM

This section contains topics that relate to using the DBMS\_STREAMS\_HANDLER\_ADM package.

- [Overview](#)
- [Security Model](#)
- [Views](#)
- [Operational Notes](#)



## Overview

A statement DML handler runs one or more data manipulation language (DML) statements on row logical change records (row LCRs) that are dequeued by an apply process. A single statement DML handler can include multiple statements, and you control the execution order of the statements.

Statement DML handlers are similar to procedure DML handlers for apply processes. Both statement DML handlers and procedure DML handlers provide custom processing of row changes that are encapsulated in row LCRs. Statement DML handlers and procedure DML handlers both run when an apply process dequeues a row LCR. However, statement DML handlers have the following advantages over procedure DML handlers:

- Statement DML handlers typically perform better than procedure DML handlers because statement DML handlers do not require PL/SQL processing.
- The syntax for statement DML handlers is same as DML syntax. Statement DML handlers do not require PL/SQL programming. Procedure DML handlers require PL/SQL programming.
- Statement DML handlers do not require the manipulation of `ANYDATA` values to access the information in row LCRs. Typically, procedure DML handlers must manipulate `ANYDATA` values.
- A statement DML handler can coexist with an error handler for same operation on the same database object. In contrast, you cannot specify both a procedure DML handler and an error handler for the same operation on the same database object.

---

---

**Note:** You can specify multiple statement DML handlers for the same operation on the database object. In this case, the statement DML handlers can execute in any order, and each statement DML handler receives a copy of the original row LCR that was dequeued by the apply process.

---

---

**See Also:** *Oracle Streams Concepts and Administration*

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

## Views

The DBMS\_STREAMS\_HANDLER\_ADM package uses the views listed in the *Oracle Database Reference*.

- DBA\_APPLY\_DML\_HANDLERS
- DBA\_STREAMS\_STMTS
- DBA\_STREAMS\_STMT\_HANDLERS

## Operational Notes

The following sections contain operational notes about the `DBMS_STREAMS_HANDLER_ADM` package:

- [Statement Execution Order](#)
- [Supported SQL Statements](#)
- [Supported Row LCR Column Attributes](#)
- [Supported Row LCR Attributes](#)
- [Supported Row LCR Extra Attributes](#)
- [Supported Row LCR Method](#)

### Statement Execution Order

Each statement in a statement DML handler has a unique execution sequence number. When a statement DML handler is invoked, it executes its statements in order from the statement with the lowest execution sequence number to the statement with the highest execution sequence number.

### Supported SQL Statements

You can use statement DML handlers for any valid DML operation on a row logical change record (row LCR). For example, a statement DML handler can audit the DML changes made to a table.

The following SQL statements are supported in statement DML handlers:

- `INSERT`
- `UPDATE`
- `DELETE`
- `MERGE`

In addition, define variables are not supported in the SQL statements in a statement DML handler.

However, the SQL statements in a statement DML handler can include calls to member subprograms for the row LCR type (`LCR$_ROW_RECORD`), such as `ADD_COLUMN`, `DELETE_COLUMN`, `KEEP_COLUMNS`, and `RENAME_COLUMN`.

---

---

**Note:** A statement DML handler cannot modify the value of a column in a row LCR.

---

---

**See Also:** ["LCR\\$\\_ROW\\_RECORD Type"](#) on page 248-15

### Supported Row LCR Column Attributes

Statements in statement DML handlers can contain the row LCR column attributes described in [Table 147-1](#).

**Table 147-1 Row LCR Column Attributes**

Attribute	Description
new	Returns the new column value in a row LCR. If the new value does not exist, then this attribute returns the old value.
new_exists	Returns TRUE if a new column value exists in a row LCR. Returns FALSE if a new column value does not exist in a row LCR.
new_only	Returns the new column value in a row LCR. If the new value does not exist, then this attribute returns NULL and does not return the old column value.
old	Returns the old column value in a row LCR.
old_exists	Returns TRUE if an old column value exists in a row LCR. Returns FALSE if an old column value does not exist in a row LCR.

Specify these attributes in the following way in a statement:

```
:attribute.column_name
```

For example, to specify the `new_only` attribute for the `salary` column, enter the following in a statement:

```
:new_only.salary
```

## Supported Row LCR Attributes

Statements in statement DML handlers can contain the row LCR attributes described in [Table 147-2](#).

**Table 147-2 Row LCR Attributes**

Attribute	Description
command_type	Returns the type of DML statement that produced the change, either INSERT, UPDATE, or DELETE. DBMS_LOB piecewise LOB operations are not supported by statement DML handlers.
commit_scn	Returns the commit system change number (SCN) of the transaction to which the LCR belongs.
compatible	Returns the minimal database compatibility required to support the LCR.
instance_number	Returns the instance number of the database instance that made the change that is encapsulated in the LCR. Typically, the instance number is relevant in an Oracle Real Application Clusters (Oracle RAC) configuration.
object_owner	Returns the schema name that contains the table with the changed row.
object_name	Returns the name of the table that contains the changed row.
scn	Returns the SCN at the time when the change was made.
source_database_name	Returns the name of the source database where the row change occurred.
source_time	Returns the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.
tag	Returns a raw tag that can be used to track the LCR.

**Table 147-2 (Cont.) Row LCR Attributes**

Attribute	Description
<code>transaction_id</code>	Returns the identifier of the transaction in which the DML statement was run.

Specify these attributes in the following way in a statement:

```
:attribute_name
```

For example, to specify the `source_database_name` attribute for a row LCR, enter the following in a statement:

```
:source_database_name
```

### Supported Row LCR Extra Attributes

Statements in statement DML handlers can contain the row LCR extra attributes described in [Table 147-3](#).

**Table 147-3 Row LCR Extra Attributes**

Attribute	Description
<code>row_id</code>	Returns the rowid of the row changed in a row LCR.
<code>serial#</code>	Returns the serial number of the session that performed the change captured in the LCR.
<code>session#</code>	Returns the identifier of the session that performed the change captured in the LCR.
<code>thread#</code>	Returns the thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle RAC configuration.
<code>tx_name</code>	Returns the name of the transaction that includes the LCR.
<code>username</code>	Returns the name of the current user who performed the change captured in the LCR.

Specify these attributes in the following way in a statement:

```
:extra_attribute.attribute_name
```

For example, to specify the `row_id` extra attribute for a row LCR, enter the following in a statement:

```
:extra_attribute.row_id
```

### Supported Row LCR Method

A statement in a statement DML handler can include a call to the `EXECUTE` member procedure for row LCRs. The `EXECUTE` member procedure executes the row LCR under the security domain of the current user.

A statement that runs the `EXECUTE` member procedure can be placed anywhere in the execution sequence order of the statement DML handler. It is not necessary to execute a row LCR unless the goal is to apply the changes in the row LCR to a table in addition to performing any other SQL statements in the statement DML handler.

When you call the `EXECUTE` member procedure in a statement, the `conflict_resolution` parameter controls whether any conflict resolution defined for the table

using the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package is used to resolve conflicts resulting from the execution of the LCR:

```
:lcr.execute TRUE|FALSE
```

A `TRUE` argument indicates that conflict resolution is used. A `FALSE` argument indicates that conflict resolution is not used.

For example, to use conflict resolution, enter the following in a statement:

```
:lcr.execute TRUE
```

An error is raised if this parameter is not specified or is set to `NULL`.

**See Also:**

- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Streams Replication Administrator's Guide*

---

## Summary of DBMS\_STREAMS\_HANDLER\_ADM Subprograms

**Table 147-4** *DBMS\_STREAMS\_HANDLER\_ADM Package Subprograms*

Subprogram	Description
<a href="#">ADD_STMT_TO_HANDLER Procedure</a> on page 147-11	Adds a statement to a statement DML handler
<a href="#">CREATE_STMT_HANDLER Procedure</a> on page 147-12	Creates a statement DML handler
<a href="#">DROP_STMT_HANDLER Procedure</a> on page 147-13	Drops a statement DML handler
<a href="#">REMOVE_STMT_FROM_HANDLER Procedure</a> on page 147-14	Removes a statement from a statement DML handler

---

---

**Note:** The subprograms in this package do not commit.

---

---



## ADD\_STMT\_TO\_HANDLER Procedure

This procedure adds a statement to a statement DML handler.

### Syntax

```
DBMS_STREAMS_HANDLER_ADM.ADD_STMT_TO_HANDLER (
  handler_name      IN VARCHAR2,
  statement         IN CLOB,
  execution_sequence IN NUMBER  DEFAULT NULL);
```

### Parameters

**Table 147-5 ADD\_STMT\_TO\_HANDLER Procedure Parameters**

Parameter	Description
handler_name	The name of the statement DML handler.
statement	The text of the SQL statement to add to the statement DML handler. If NULL, then the procedure raises an error.
execution_sequence	The position of the statement in the statement DML handler at which a SQL statement is to be set to execute. Statements are executed in order from the lowest execution sequence number to the highest execution sequence number.  You can specify a positive or negative integer or decimal, or you can specify 0 (zero).  If you specify an execution sequence number that is used by an existing statement in the statement DML handler, then the statement in the statement parameter replaces the existing statement.  If NULL, then the statement is added to the statement DML handler with an execution sequence number that is larger than the execution sequence number for any statement in the statement DML handler.

## CREATE\_STMT\_HANDLER Procedure

This procedure creates a statement DML handler.

### Syntax

```
DBMS_STREAMS_HANDLER_ADM.CREATE_STMT_HANDLER(  
    handler_name IN VARCHAR2,  
    comment      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 147–6 CREATE\_STMT\_HANDLER Procedure Parameters**

Parameter	Description
handler_name	The name of the statement DML handler.
comment	A comment for the statement DML handler. If NULL, then no comment is recorded for the statement DML handler.

## DROP\_STMT\_HANDLER Procedure

This procedure drops a statement DML handler.

### Syntax

```
DBMS_STREAMS_HANDLER_ADM.DROP_STMT_HANDLER(  
    handler_name IN VARCHAR2);
```

### Parameters

**Table 147-7** *DROP\_STMT\_HANDLER Procedure Parameters*

Parameter	Description
handler_name	The name of the statement DML handler.

## REMOVE\_STMT\_FROM\_HANDLER Procedure

This procedure removes a statement from a statement DML handler.

### Syntax

```
DBMS_STREAMS_HANDLER_ADM.REMOVE_STMT_FROM_HANDLER(  
    handler_name      IN VARCHAR2,  
    execution_sequence IN NUMBER  DEFAULT NULL);
```

### Parameters

**Table 147–8 REMOVE\_STMT\_FROM\_HANDLER Procedure Parameters**

Parameter	Description
handler_name	The name of the statement DML handler.
execution_ sequence	The position of the statement to remove. You can specify a positive or negative integer or decimal, or you can specify 0 (zero). If NULL, the procedure removes the last statement in the statement DML handler. If the specified execution sequence number does not exist for the statement DML handler, then the procedure raises an error.

---

---

## DBMS\_STREAMS\_MESSAGING

The `DBMS_STREAMS_MESSAGING` package, one of a set of Oracle Streams packages, provides interfaces to enqueue messages into and dequeue messages from a `ANYDATA` queue.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS\\_MESSAGING](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_STREAMS\\_MESSAGING Subprograms](#)

## Using DBMS\_STREAMS\_MESSAGING

This section contains topics which relate to using the DBMS\_CAPTURE\_ADM package.

- [Overview](#)
- [Security Model](#)

## Overview

This package provides interfaces to enqueue messages into and dequeue messages from a ANYDATA queue.

---

---

**Note:** Currently, messaging clients cannot dequeue buffered messages. In addition, the DBMS\_STREAMS\_MESSAGING package cannot be used to enqueue messages into or dequeue messages from a buffered queue. However, you can use the DBMS\_AQ package to enqueue and dequeue buffered messages.

---

---

### See Also:

- *Oracle Database 2 Day + Data Replication and Integration Guide* for more information about Oracle Streams and for an example that uses the procedures in this package
- *Oracle Streams Advanced Queuing User's Guide* for more information about queues, messaging, and the DBMS\_AQ package

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator



---

## Summary of DBMS\_STREAMS\_MESSAGING Subprograms

**Table 148–1** *DBMS\_STREAMS\_MESSAGING Package Subprograms*

Subprogram	Description
<a href="#">DEQUEUE Procedure</a> on page 148-6	Uses the specified Oracle Streams messaging client to dequeue a message from the specified queue
<a href="#">ENQUEUE Procedure</a> on page 148-8	The current user enqueues a message into the specified queue

---

**Note:** The subprograms in this package do not commit.

---

## DEQUEUE Procedure

This procedure uses the specified Oracle Streams messaging client to dequeue a message from the specified queue.

This procedure is overloaded. One version of this procedure contains the `msgid` OUT parameter, and the other does not.

### Syntax

```
DBMS_STREAMS_MESSAGING.DEQUEUE (
  queue_name      IN   VARCHAR2,
  streams_name    IN   VARCHAR2,
  payload         OUT  ANYDATA,
  dequeue_mode    IN   VARCHAR2          DEFAULT 'REMOVE',
  navigation      IN   VARCHAR2          DEFAULT 'NEXT MESSAGE',
  wait           IN   BINARY_INTEGER     DEFAULT FOREVER,
  msgid          OUT  RAW);
```

```
DBMS_STREAMS_MESSAGING.DEQUEUE (
  queue_name      IN   VARCHAR2,
  streams_name    IN   VARCHAR2,
  payload         OUT  ANYDATA,
  dequeue_mode    IN   VARCHAR2          DEFAULT 'REMOVE',
  navigation      IN   VARCHAR2          DEFAULT 'NEXT MESSAGE',
  wait           IN   BINARY_INTEGER     DEFAULT FOREVER);
```

### Parameters

**Table 148–2 DEQUEUE Procedure Parameters**

Parameter	Description
<code>queue_name</code>	<p>The name of the local queue from which messages will be dequeued, specified as [<i>schema_name</i>.] <i>queue_name</i>. The current database must contain the queue, and the queue must be a secure queue of ANYDATA type.</p> <p>For example, to specify a queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.</p>
<code>streams_name</code>	<p>The name of the Oracle Streams messaging client. For example, if the user <code>strmadmin</code> is the messaging client, then specify <code>strmadmin</code>.</p> <p>If NULL and a relevant messaging client for the queue exists, then the procedure uses the relevant messaging client. If NULL and multiple relevant messaging clients for the queue exist, then the procedure raises an error.</p>
<code>payload</code>	The payload that is dequeued
<code>dequeue_mode</code>	<p>Specify one of the following settings:</p> <p>REMOVE: Read the message and delete it. This setting is the default. The message can be retained in the queue table based on the retention properties.</p> <p>LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This setting is equivalent to a select for update statement.</p> <p>BROWSE: Read the message without acquiring any lock on the message. This specification is equivalent to a select statement.</p>

**Table 148–2 (Cont.) DEQUEUE Procedure Parameters**

Parameter	Description
navigation	<p>The position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved.</p> <p>Specify one of the following settings:</p> <p><b>NEXT MESSAGE:</b> Retrieve the next message that is available and matches the search criteria. If the previous message belongs to a message group, then retrieve the next available message that matches the search criteria and belongs to the message group. This setting is the default.</p> <p><b>NEXT TRANSACTION:</b> Skip the remainder of the current message group (if any) and retrieve the first message of the next message group. This setting can only be used if message grouping is enabled for the current queue.</p> <p><b>FIRST MESSAGE:</b> Retrieves the first message which is available and matches the search criteria. This setting resets the position to the beginning of the queue.</p> <p><b>Note:</b> Each message group contains the messages in a single transaction.</p> <p><b>See Also:</b> <i>Oracle Streams Advanced Queuing User's Guide</i> for more information about dequeue options</p>
wait	<p>Either <code>FOREVER</code> or <code>NO_WAIT</code></p> <p>If <code>FOREVER</code>, then the dequeue call is blocked without a time out until a message is available in the queue.</p> <p>If <code>NO_WAIT</code>, then a wait time of zero seconds is used. In this case, the dequeue will return immediately even if there are no messages in the queue.</p>
msgid	The message identifier of the message that is dequeued

## Exceptions

**Table 148–3 DEQUEUE Procedure Exceptions**

Exception	Description
ENDOFCURTRANS	<p>Dequeue has reached the end of the messages in the current transaction. Specify this exception in the following way:</p> <pre>SYS.DBMS_STREAMS_MESSAGING.ENDOFCURTRANS</pre> <p>Every dequeue procedure should include an exception handler that handles this exception.</p>
NOMOREMSGS	<p>There are no more messages in the queue for the dequeue operation. Specify this exception in the following way:</p> <pre>SYS.DBMS_STREAMS_MESSAGING.NOMOREMSGS</pre> <p>A dequeue procedure that specifies <code>NO_WAIT</code> for the <code>wait</code> parameter should include an exception handler that handles this exception.</p>

## ENQUEUE Procedure

This procedure enables the current user to enqueue a message into the specified queue.

This procedure is overloaded. One version of this procedure contains the `msgid` OUT parameter, and the other does not.

### Syntax

```
DBMS_STREAMS_MESSAGING.ENQUEUE (
  queue_name IN  VARCHAR2,
  payload    IN  ANYDATA,
  msgid      OUT RAW);
```

```
DBMS_STREAMS_MESSAGING.ENQUEUE (
  queue_name IN  VARCHAR2,
  payload    IN  ANYDATA);
```

### Parameters

**Table 148–4 ENQUEUE Procedure Parameters**

Parameter	Description
<code>queue_name</code>	The name of the local queue into which messages will be enqueued, specified as <code>[schema_name.]queue_name</code> . The current database must contain the queue, and the queue must be a secure queue of ANYDATA type.  For example, to specify a queue named <code>streams_queue</code> in the <code>strmadmin</code> schema, enter <code>strmadmin.streams_queue</code> for this parameter. If the schema is not specified, then the current user is the default.
<code>payload</code>	The payload that is enqueued
<code>msgid</code>	The message identifier of the message that is enqueued

### Usage Notes

To successfully enqueue messages into a queue, the current user must be mapped to a unique Advanced Queuing agent with the same name as the current user. You can run the `DBMS_STREAMS_ADM.SET_UP_QUEUE` procedure and specify a user as the queue user to grant the necessary privileges to the user to perform enqueues. The Advanced Queuing agent is created automatically when you run `SET_UP_QUEUE` and specify a queue user.

**See Also:** [SET\\_UP\\_QUEUE Procedure](#) on page 144-172

---

---

## DBMS\_STREAMS\_TABLESPACE\_ADM

The DBMS\_STREAMS\_TABLESPACE\_ADM package, one of a set of Oracle Streams packages, provides administrative interfaces for copying tablespaces between databases and moving tablespaces from one database to another. This package uses transportable tablespaces, Data Pump, the DBMS\_FILE\_TRANSFER package, and the DBMS\_FILE\_GROUP package.

This chapter contains the following topics:

- [Using DBMS\\_STREAMS\\_TABLESPACE\\_ADM](#)
  - Overview
  - Security Model
- [Data Structures](#)
- [Summary of DBMS\\_STREAMS\\_TABLESPACE\\_ADM Subprograms](#)

**See Also:** *Oracle Streams Concepts and Administration* and *Oracle Streams Replication Administrator's Guide* for more information about this package and Oracle Streams

## Using DBMS\_STREAMS\_TABLESPACE\_ADM

This section contains topics which relate to using the DBMS\_STREAMS\_TABLESPACE\_ADM package.

- [Overview](#)
- [Security Model](#)

## Overview

Either a simple tablespace or a self-contained tablespace set must be specified in each procedure in this package.

A **self-contained tablespace** has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. A **simple tablespace** is a self-contained tablespace that uses only one datafile.

A simple tablespace must be specified in the following procedures:

- [ATTACH\\_SIMPLE\\_TABLESPACE Procedure](#)
- [CLONE\\_SIMPLE\\_TABLESPACE Procedure](#)
- [DETACH\\_SIMPLE\\_TABLESPACE Procedure](#)
- [PULL\\_SIMPLE\\_TABLESPACE Procedure](#)

A **self-contained tablespace set** has no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained.

A self-contained tablespace set must be specified in the following procedures:

- [ATTACH\\_TABLESPACES Procedure](#)
- [CLONE\\_TABLESPACES Procedure](#)
- [DETACH\\_TABLESPACES Procedure](#)
- [PULL\\_TABLESPACES Procedure](#)

To determine whether a set of tablespaces is self-contained, use the `TRANSPORT_SET_CHECK` procedure in the Oracle supplied package `DBMS_TTS`.

**See Also:** *Oracle Database Administrator's Guide* for more information about self-contained tablespaces and tablespace sets

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting `EXECUTE` on this package to selected users or roles.
- Granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.



## Data Structures

The DBMS\_STREAMS\_TABLESPACE\_ADM package defines RECORD types and TABLE types.

### RECORD Types

- [FILE Record Type](#)

### TABLE Types

- [DIRECTORY\\_OBJECT\\_SET Table Type](#)
- [FILE\\_SET Table Type](#)
- [TABLESPACE\\_SET Table Type](#)

## DIRECTORY\_OBJECT\_SET Table Type

Contains the names of one or more directory objects. Each name must be a directory object created using the SQL statement `CREATE DIRECTORY`.

### Syntax

```
TYPE DIRECTORY_OBJECT_SET IS TABLE OF VARCHAR2(32)
INDEX BY BINARY_INTEGER;
```

## FILE Record Type

Contains the directory object associated with a directory and the name of the file in the directory.

### Syntax

```
TYPE FILE IS RECORD(  
    directory_object VARCHAR2(32),  
    file_name        VARCHAR2(4000));
```

### Fields

**Table 149-1 FILE Fields**

Field	Description
directory_object	The name of a directory object. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.
file_name	The name of the file in the corresponding directory associated with the directory object

## FILE\_SET Table Type

Contains one or more files.

### Syntax

```
TYPE FILE_SET IS TABLE OF FILE  
INDEX BY BINARY_INTEGER;
```

## TABLESPACE\_SET Table Type

Contains the names of one or more tablespaces.

### Syntax

```
TYPE TABLESPACE_SET IS TABLE OF VARCHAR2(32)  
INDEX BY BINARY_INTEGER;
```

---

## Summary of DBMS\_STREAMS\_TABLESPACE\_ADM Subprograms

**Table 149–2 DBMS\_STREAMS\_TABLESPACE\_ADM Package Subprograms**

Subprogram	Description
<a href="#">ATTACH_SIMPLE_TABLESPACE Procedure</a> on page 149-11	Uses Data Pump to import a simple tablespace previously exported using the DBMS_STREAMS_TABLESPACE_ADM package or Data Pump export
<a href="#">ATTACH_TABLESPACES Procedure</a> on page 149-13	Uses Data Pump to import a self-contained tablespace set previously exported using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the Recovery Manager (RMAN) <code>TRANSPORT TABLESPACE</code> command
<a href="#">CLONE_SIMPLE_TABLESPACE Procedure</a> on page 149-18	Clones a simple tablespace. The tablespace can later be attached to a database.
<a href="#">CLONE_TABLESPACES Procedure</a> on page 149-20	Clones a set of self-contained tablespaces. The tablespaces can later be attached to a database.
<a href="#">DETACH_SIMPLE_TABLESPACE Procedure</a> on page 149-24	Detaches a simple tablespace. The tablespace can later be attached to a database.
<a href="#">DETACH_TABLESPACES Procedure</a> on page 149-26	Detaches a set of self-contained tablespaces. The tablespaces can later be attached to a database.
<a href="#">PULL_SIMPLE_TABLESPACE Procedure</a> on page 149-30	Copies a simple tablespace from a remote database and attaches it to the current database
<a href="#">PULL_TABLESPACES Procedure</a> on page 149-32	Copies a set of self-contained tablespaces from a remote database and attaches the tablespaces to the current database

---

**Note:** All subprograms commit unless specified otherwise.

---

## ATTACH\_SIMPLE\_TABLESPACE Procedure

This procedure uses Data Pump to import a simple tablespace previously exported using the DBMS\_STREAMS\_TABLESPACE\_ADM package, Data Pump export, or the Recovery Manager (RMAN) TRANSPORT TABLESPACE command.

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_SIMPLE_TABLESPACE (
  directory_object      IN  VARCHAR2,
  tablespace_file_name IN  VARCHAR2,
  converted_file_name   IN  VARCHAR2  DEFAULT NULL,
  datafile_platform    IN  VARCHAR2  DEFAULT NULL,
  tablespace_name      OUT VARCHAR2);
```

### Parameters

**Table 149–3** ATTACH\_SIMPLE\_TABLESPACE Procedure Parameters

Parameter	Description
directory_object	<p>The directory that contains the Data Pump dump file and the datafile for the tablespace. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.</p> <p>The name of the Data Pump export dump file must be the same as the data file name for the tablespace, except with a .dmp extension. If the converted_file_name is non-NULL, specify the dump file produced by the export database, not the file name after conversion.</p> <p>The Data Pump import log file is written to this directory. The name of the log file is the same as the data file name for the tablespace, except with an .alg extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</p> <p>If NULL, then the procedure raises an error.</p>
tablespace_file_name	<p>The name of the datafile for the tablespace being imported.</p> <p>If NULL, then the procedure raises an error.</p>
converted_file_name	<p>If the datafile_platform parameter is non-NULL and is different from the platform of the local import database, then specify a file name for the converted datafile. The datafile is converted to the platform of the local import database and copied to the new file name. The existing datafile is not modified nor deleted.</p> <p>If non-NULL and the datafile_platform parameter is NULL, then the procedure ignores this parameter.</p> <p>If non-NULL and the datafile_platform parameter specifies the same platform as the local import database, then the procedure ignores this parameter.</p> <p>If NULL and the datafile_platform parameter is non-NULL, then the procedure raises an error.</p>

**Table 149–3 (Cont.) ATTACH\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
datafile_platform	<p>Specify NULL if the platform is the same for the export database and the current import database.</p> <p>Specify the platform for the export database if the platform is different for the export database and the import database.</p> <p>You can determine the platform of a database by querying the PLATFORM_NAME column in the V\$DATABASE dynamic performance view. The V\$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.</p>
tablespace_name	<p>Contains the name of the attached tablespace. The attached tablespace is read-only. Use an ALTER TABLESPACE statement to make the tablespace read/write if necessary.</p>

## Usage Notes

To run this procedure, a user must meet the following requirements:

- Have IMP\_FULL\_DATABASE role
- Have READ and WRITE privilege on the directory object that contains the Data Pump export dump file and the datafiles for the tablespaces in the set, specified by the directory\_object parameter

Automatic Storage Management (ASM) directories cannot be used with this procedure.

**See Also:** [Overview](#) on page 149-3



## ATTACH\_TABLESPACES Procedure

This procedure uses Data Pump to import a self-contained tablespace set previously exported using the DBMS\_STREAMS\_TABLESPACE\_ADM package, Data Pump export, or the Recovery Manager (RMAN) `TRANSPORT TABLESPACE` command.

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the `datapump_job_name` parameter. This job performs the Data Pump import to complete the attach operation. In addition, if the platform at the export database is different than the local database platform, then this procedure optionally can create datafiles for the tablespace set that can be used with the local platform.
- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump import is performed. This version of the procedure uses the files in a file group version and can copy the export dump file, export log file, and the datafiles that comprise the tablespace set into the specified directories. The file group and version are specified using the `file_group_name` and `version_name` parameters, respectively. This version of the procedure does not require a datafiles platform specification if the platform at the export database is different than the local database platform. Instead, the tablespace set is migrated automatically to the correct platform when it is attached.

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES (
  datapump_job_name      IN OUT VARCHAR2,
  dump_file              IN      FILE,
  tablespace_files       IN      FILE_SET,
  converted_files        IN      FILE_SET,
  datafiles_platform     IN      VARCHAR2  DEFAULT NULL,
  log_file               IN      FILE      DEFAULT NULL,
  tablespace_names       OUT     TABLESPACE_SET);
```

```
DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES (
  file_group_name        IN      VARCHAR2,
  version_name           IN      VARCHAR2  DEFAULT NULL,
  datafiles_directory_object IN  VARCHAR2  DEFAULT NULL,
  logfile_directory_object IN  VARCHAR2  DEFAULT NULL,
  repository_db_link     IN      VARCHAR2  DEFAULT NULL,
  tablespace_names       OUT     TABLESPACE_SET);
```

### Parameters

**Table 149–4 ATTACH\_TABLESPACES Procedure Parameters**

Parameter	Description
<code>data_pump_job_name</code>	The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily.  If <code>NULL</code> , then the system generates a Data Pump job name.

**Table 149–4 (Cont.) ATTACH\_TABLESPACES Procedure Parameters**

Parameter	Description
dump_file	The file name of the Data Pump dump file to import. If NULL or if a file attribute is NULL, then the procedure raises an error.
tablespace_files	The file set that contains the datafiles for the tablespace set being imported. If NULL, then the procedure raises an error.
converted_files	If the datafiles_platform parameter is non-NULL and is different from the platform for the local import database, then specify a file set with the names of the converted datafiles. The datafiles are converted to the platform of the local import database and copied to the new file names. In this case, the number of files in the specified file set must match the number of files in the file set specified for the tablespace_files parameter. The existing datafiles are not modified nor deleted.  If non-NULL and the datafiles_platform parameter is NULL, then the procedure ignores this parameter.  If non-NULL and the datafiles_platform parameter specifies the same platform as the local import database, then the procedure ignores this parameter.  If NULL and the datafiles_platform parameter is non-NULL, then the procedure raises an error.
datafiles_platform	Specify NULL if the platform is the same for the export database and the current import database.  Specify the platform for the export database if the platform is different for the export database and the import database.  You can determine the platform of a database by querying the PLATFORM_NAME column in the V\$DATABASE dynamic performance view. The V\$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.
log_file	Specify the log file name for the Data Pump import. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .alg and places it in the Data Pump export dump file directory.  If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.
file_group_name	The name of the file group, specified as [ <i>schema_name</i> .] <i>file_group_name</i> . For example, if the schema is hq_dba and the file group name is sales, then specify hq_dba.sales. If the schema is not specified, then the current user is the default.
version_name	The name of the file group version to attach. If NULL, then the procedure uses the version with the latest creation time for the file group.

**Table 149–4 (Cont.) ATTACH\_TABLESPACES Procedure Parameters**

Parameter	Description
<code>datafiles_directory_object</code>	<p>The directory object into which the datafiles and Data Pump export dump file are copied. The files are copied from the tablespace repository directories to this directory.</p> <p>If non-NULL, the attached tablespaces use the files in specified directory. However, the file group version specified in the <code>version_name</code> parameter consists of the files in the original directory, not in the directory specified by this <code>datafiles_directory_object</code> parameter.</p> <p>If NULL, then the procedure does not copy the datafiles and dump file.</p>
<code>logfile_directory_object</code>	<p>The directory object into which the Data Pump import log file is placed. The system generates a log file name with the extension <code>.alg</code>.</p> <p>If NULL, then the procedure places the import log file in the same directory as the dump file.</p>
<code>repository_db_link</code>	<p>If the file group is in a different database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure.</p> <p>If this parameter is non-NULL, then meet the following requirements:</p> <ul style="list-style-type: none"> <li>■ Each directory object that contains files in the version being attached must exist on both databases.</li> <li>■ The corresponding directory objects must have the same names on both databases.</li> </ul> <p>If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.</p>
<code>tablespace_names</code>	<p>Contains the names of the attached tablespaces. The attached tablespaces are read-only. Use <code>ALTER TABLESPACE</code> statements to make the tablespaces read/write if necessary.</p>

## Usage Notes

The following sections contain usage notes for this procedure:

- [User Requirements](#)
- [Procedures Used to Clone or Detach a Tablespace Set](#)
- [When the Attach Database Is Different Than the Clone or Detach Database](#)
- [Automatic Storage Management Directories](#)

### See Also:

- [Overview](#) on page 149-3
- *Oracle Streams Concepts and Administration*

### User Requirements

To run either version of this procedure, a user must meet the following requirements:

- Have `IMP_FULL_DATABASE` role

- Have `READ` and `WRITE` privilege on the directory objects that contain the Data Pump export dump file and the datafiles for the tablespaces in the set, specified by the `dump_file` and `tablespace_files` parameters, or by the `datafiles_directory_object` parameter
- Have `WRITE` privilege on the directory object that will hold the Data Pump import log file, specified by the `log_file` parameter or `logfile_directory_object` parameter if it is non-NULL

If the Data Pump job version of the procedure is run, then the user must have `WRITE` privilege on the directory objects that will hold the converted datafiles for the tablespaces in the set if platform conversion is necessary. These directory objects are specified by the `converted_files` parameter if it is non-NULL.

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

### Procedures Used to Clone or Detach a Tablespace Set

After a tablespace set is cloned or detached using the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure, respectively, the tablespace set can be attached to a database using the `ATTACH_TABLESPACES` procedure. If the Data Pump job version of the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure was used, then use the Data Pump job version of the `ATTACH_TABLESPACES` procedure. If the file group version of the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure was used, then use the file group version of the `ATTACH_TABLESPACES` procedure.

#### See Also:

- [CLONE\\_TABLESPACES Procedure](#) on page 149-20
- [DETACH\\_TABLESPACES Procedure](#) on page 149-26

### When the Attach Database Is Different Than the Clone or Detach Database

You can attach a tablespace set to a different database than the database from which the tablespace set was cloned or detached. The two databases might or might not share a file system. If the two databases do not share a file system, then you must transfer the dump file and datafiles to the remote system using the `DBMS_FILE_TRANSFER` package, FTP, or some other method. You can attach the tablespace set in one of the following ways depending on the version of the `ATTACH_TABLESPACES` procedure you use:

- If you use the Data Pump job version of the procedure, then specify the relevant files on the file system. The directory object names can be different in the databases.
- If you use the file group version of the procedure, then you can use the `repository_db_link` parameter to specify the database where tablespace repository resides. The directory objects for the files must exist and must match in the databases.

#### See Also:

- [CLONE\\_TABLESPACES Procedure](#) on page 149-20
- [DETACH\\_TABLESPACES Procedure](#) on page 149-26
- [Chapter 67, "DBMS\\_FILE\\_GROUP"](#) for more information about file groups

### **Automatic Storage Management Directories**

Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

**See Also:** *Oracle Database Utilities* for information about specifying ASM directories for directory objects

## CLONE\_SIMPLE\_TABLESPACE Procedure

This procedure clones a simple tablespace. The specified tablespace must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace and places the dump file in the specified directory
3. Places the datafile for the specified tablespace in the specified directory
4. If this procedure made the tablespace read-only, then makes the tablespace read/write

In addition, this procedure optionally can create a datafile for the tablespace that can be used with a platform that is different than the local database platform.

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.CLONE_SIMPLE_TABLESPACE (
    tablespace_name      IN  VARCHAR2,
    directory_object     IN  VARCHAR2,
    destination_platform IN  VARCHAR2 DEFAULT NULL,
    tablespace_file_name OUT VARCHAR2);
```

### Parameters

**Table 149–5 CLONE\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
tablespace_name	The tablespace to be cloned. If NULL, then the procedure raises an error.
directory_object	The directory where the Data Pump export dump file, the Data Pump export log file, and the datafile for the tablespace are placed. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.  The name of the Data Pump export dump file is the same as the data file name for the tablespace, except with a .dmp extension. If a file exists with the same name as the dump file in the directory, then the procedure raises an error.  The name of the log file is the same as the data file name for the tablespace, except with a .clg extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.  If NULL, then the procedure raises an error.
destination_platform	Specify NULL if the platform is the same for the current export database and the intended import database.  Specify the platform for the intended import database if the platform is different for the export database and the import database.  You can determine the platform of a database by querying the PLATFORM_NAME column in the V\$DATABASE dynamic performance view. The V\$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.

**Table 149-5 (Cont.) CLONE\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
tablespace_file_name	Contains the name of the cloned tablespace datafile. This datafile is placed in the directory specified by the parameter <code>directory_object</code> .

## Usage Notes

To run this procedure, a user must meet the following requirements:

- Have `EXP_FULL_DATABASE` role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include `DBA_TABLESPACES` and `USER_TABLESPACES`.
- Have `MANAGE TABLESPACE` or `ALTER TABLESPACE` on a tablespace if the tablespace must be made read-only
- Have `READ` privilege on the directory object for the directory that contains the datafile for the tablespace. The name of this tablespace is specified by the `tablespace_name` parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure.
- Have `READ` and `WRITE` privilege on the directory object that will contain the Data Pump export dump file, specified by the `directory_object` parameter
- If the file group version of the procedure is run, then the user must have the necessary privileges to manage file group.

After cloning a tablespace using this procedure, you can add the tablespace to a different database using the `ATTACH_SIMPLE_TABLESPACE` procedure. If the database is a remote database and you want to use the `ATTACH_SIMPLE_TABLESPACE` procedure, then you can transfer the dump file and datafile to the remote system using the `DBMS_FILE_TRANSFER` package, FTP, or some other method.

Automatic Storage Management (ASM) directories cannot be used with this procedure.

### See Also:

- [Overview](#) on page 149-3
- [ATTACH\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-11 and [PULL\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-30

## CLONE\_TABLESPACES Procedure

This procedure clones a set of self-contained tablespaces. All of the tablespaces in the specified tablespace set must be online.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set read-only
2. Uses Data Pump to export the metadata for the tablespaces in the tablespace set and places the dump file in the specified directory
3. Places the datafiles that comprise the specified tablespace set in the specified directory
4. If this procedure made a tablespace read-only, then makes the tablespace read/write

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the `datapump_job_name` parameter. This job performs the Data Pump export. This version of the procedure completes the clone operation by placing the export dump file, export log file, and the datafiles that comprise the tablespace set in the specified directories, but the files are not added to a file group version. In addition, this version of the procedure optionally can create datafiles for the tablespace set that can be used with a platform that is different than the local database platform.
- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump export is performed, and this version of the procedure completes the clone operation by placing the export dump file, export log file, and the datafiles that comprise the tablespace set in the appropriate file group version. The file group and version are specified using the `file_group_name` and `version_name` parameters, respectively. This version of the procedure does not require a destination platform specification if the destination platform is different. Instead, the tablespace set is migrated automatically to the correct platform when it is attached at the destination database using the file group version of the `ATTACH_TABLESPACES` procedure.

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES (
  datapump_job_name          IN OUT VARCHAR2,
  tablespace_names           IN     TABLESPACE_SET,
  dump_file                  IN     FILE,
  tablespace_directory_objects IN  DIRECTORY_OBJECT_SET,
  destination_platform       IN  VARCHAR2  DEFAULT NULL,
  log_file                   IN     FILE    DEFAULT NULL,
  tablespace_files           OUT    FILE_SET);
```

```
DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES (
  tablespace_names           IN  TABLESPACE_SET,
  tablespace_directory_object IN  VARCHAR2  DEFAULT NULL,
  log_file_directory_object  IN  VARCHAR2  DEFAULT NULL,
  file_group_name            IN  VARCHAR2,
  version_name               IN  VARCHAR2  DEFAULT NULL,
  repository_db_link         IN  VARCHAR2  DEFAULT NULL);
```



## Parameters

**Table 149–6 CLONE\_TABLESPACES Procedure Parameters**

Parameter	Description
data_pump_job_name	<p>The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily.</p> <p>If NULL, then the system generates a Data Pump job name.</p>
tablespace_names	<p>The tablespace set to be cloned.</p> <p>If NULL, then the procedure raises an error.</p>
dump_file	<p>The file name of the Data Pump dump file that is exported.</p> <p>If NULL or if a file attribute is NULL, then the procedure raises an error.</p> <p>If the specified file exists, then the procedure raises an error.</p>
tablespace_directory_objects	<p>The set of directory objects into which the datafiles for the tablespaces are copied. If multiple directory objects are in the set, then the procedure copies a datafile to each directory object in the set in sequence. In this case, if the end of the directory object set is reached, then datafile copying starts again with the first directory object in the set.</p> <p>If NULL, then the procedure copies datafiles for the tablespace set to the dump file directory.</p>
destination_platform	<p>Specify NULL if the platform is the same for the current export database and the intended import database.</p> <p>Specify the platform for the intended import database if the platform is different for the export database and the import database.</p> <p>You can determine the platform of a database by querying the PLATFORM_NAME column in the V\$DATABASE dynamic performance view. The V\$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.</p>
log_file	<p>Specify the log file name for the Data Pump export.</p> <p>If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .clg and places it in the dump file directory.</p> <p>If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</p>

**Table 149–6 (Cont.) CLONE\_TABLESPACES Procedure Parameters**

Parameter	Description
tablespace_directory_object	<p>The directory object into which the data files are copied and Data Pump export dump file is placed. The system generates a dump file name with the extension <code>.dmp</code>.</p> <p>If <code>NULL</code>, then the procedure copies the datafiles to and places the dump file in the default directory object for the version. If the version does not have a default directory object, then the procedure uses the default directory object for the file group.</p> <p>If <code>NULL</code> and no default directory object exists for the version or file group, then the procedure raises an error.</p>
log_file_directory_object	<p>The directory object into which the Data Pump export log file is placed. The system generates a log file name with the extension <code>.clg</code>.</p> <p>If <code>NULL</code>, then the procedure uses the directory object specified in <code>tablespace_directory_object</code>.</p>
file_group_name	<p>The name of the file group, specified as <code>[schema_name.]file_group_name</code>. For example, if the schema is <code>hq_dba</code> and the file group name is <code>sales</code>, then specify <code>hq_dba.sales</code>. If the schema is not specified, then the current user is the default.</p> <p>If the specified file group does not exist, then the procedure creates it.</p>
version_name	<p>The name of the version into which the cloned tablespace set is placed. The specified version name cannot be a positive integer.</p> <p>If the specified version does not exist, then the procedure creates it.</p> <p>If the specified version exists, then the procedure adds the tablespace set to the version. Only one Data Pump export dump file can exist in a version.</p> <p>If <code>NULL</code>, then the procedure creates a new version, and the version number can be used to manage the version.</p>
repository_db_link	<p>If the file group is in a remote database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure.</p> <p>If this parameter is non-<code>NULL</code>, then the directory object specified in <code>tablespace_directory_object</code> must exist on the local database and on the remote database. If <code>tablespace_directory_object</code> is <code>NULL</code>, then the default directory object must exist on both databases. The directory object must have the same name on each database and must correspond to the same directory on a shared file system.</p> <p>If <code>NULL</code>, then the procedure does not use a database link, and the procedure uses the file group in the local database.</p>
tablespace_files	<p>Contains the datafiles for the cloned tablespace set. These datafiles are placed in the directories specified by the directory objects in the parameter <code>tablespace_directory_objects</code>.</p>

## Usage Notes

To run either version of this procedure, a user must meet the following requirements:

- Have `EXP_FULL_DATABASE` role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include `DBA_TABLESPACES` and `USER_TABLESPACES`.
- Have `MANAGE TABLESPACE` or `ALTER TABLESPACE` on a tablespace if the tablespace must be made read-only
- Have `READ` privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the `tablespace_names` parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.
- Have `READ` and `WRITE` privilege on the directory object that will contain the Data Pump export dump file, specified by the `dump_file` parameter or the `tablespace_directory_object` parameter
- Have `WRITE` privilege on the directory objects that will contain the copied datafiles for the tablespaces in the set, specified by the `tablespace_directory_objects` parameter if non-NULL or the `tablespace_directory_object` parameter
- Have `WRITE` privilege on the directory object that will contain the Data Pump export log file, specified by the `log_file` parameter if non-NULL or the `log_file_directory_object` parameter if non-NULL

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.

Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

After cloning a tablespace set using this procedure, you can attach the tablespaces to a different database using the `ATTACH_TABLESPACES` procedure.

### See Also:

- [Overview](#) on page 149-3
- [ATTACH\\_TABLESPACES Procedure](#) on page 149-13
- [Chapter 67, "DBMS\\_FILE\\_GROUP"](#) for more information about file groups
- *Oracle Streams Concepts and Administration*

## DETACH\_SIMPLE\_TABLESPACE Procedure

This procedure detaches a simple tablespace. The specified tablespace must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace and places the dump file in the directory that contains the tablespace datafile
3. Drops the tablespace and its contents from the database

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.DETACH_SIMPLE_TABLESPACE (
  tablespace_name      IN VARCHAR2,
  directory_object     OUT VARCHAR2,
  tablespace_file_name OUT VARCHAR2);
```

### Parameters

**Table 149–7 DETACH\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
data_pump_job_name	The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily. If NULL, then the system generates a Data Pump job name.
directory_object	Contains the directory where the Data Pump export dump file and the Data Pump export log file are placed. The procedure uses the directory of the datafile for the tablespace. Therefore, make sure a directory object created using the SQL statement CREATE DIRECTORY exists for this directory.  The name of the Data Pump export dump file is the same as the data file name for the tablespace, except with a .dmp extension. If a file exists with the same name as the dump file in the directory, then the procedure raises an error.  The name of the log file is the same as the data file name for the tablespace, except with a .dlg extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.
tablespace_file_name	Contains the name of the detached tablespace datafile.

### Usage Notes

To run this procedure, a user must meet the following requirements:

- Have EXP\_FULL\_DATABASE role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA\_TABLESPACES and USER\_TABLESPACES.
- Have DROP TABLESPACE privilege
- Have MANAGE TABLESPACE or ALTER TABLESPACE on a tablespace if the tablespace must be made read-only
- Have READ and WRITE privilege on the directory object for the directory that contains the tablespace datafile. The name of this tablespace is specified by the

tablespace\_name parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure. This directory also will contain the Data Pump export dump file generated by this procedure.

After detaching a tablespace using this procedure, you can add the tablespace to a different database using the ATTACH\_SIMPLE\_TABLESPACE procedure. If the database is a remote database and you want to use the ATTACH\_SIMPLE\_TABLESPACE procedure, then you can transfer the dump file and datafile to the remote system using the DBMS\_FILE\_TRANSFER package, FTP, or some other method. You can use the two OUT parameters in this procedure to accomplish the attach or pull operation.

Automatic Storage Management (ASM) directories cannot be used with this procedure.

---

---

**Note:** Do not use the DETACH\_SIMPLE\_TABLESPACE procedure on a tablespace if the tablespace is using the Oracle-managed files feature. If you do, then the datafile for the tablespace is dropped automatically when the tablespace is dropped.

---

---

**See Also:**

- [Overview](#) on page 149-3
- [ATTACH\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-11 and [PULL\\_SIMPLE\\_TABLESPACE Procedure](#) on page 149-30
- *Oracle Database Administrator's Guide* for more information about the Oracle-managed files feature

## DETACH\_TABLESPACES Procedure

This procedure detaches a set of self-contained tablespaces. All of the tablespaces in the specified tablespace set must be online and any table partitions must not span tablespaces in the tablespace set.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set read-only
2. Uses Data Pump to export the metadata for the tablespace set and places the dump file in the specified directory
3. Drops the tablespaces in the specified tablespace set and their contents from the database

This procedure does not move or copy the datafiles that comprise the specified tablespace set.

This procedure is overloaded and consists of the following versions:

- One version of the procedure uses a Data Pump job name in the `datapump_job_name` parameter. This job performs the Data Pump export. This version of the procedure completes the detach operation by placing the export dump file and export log file in the specified directories, but the files are not added to a file group version.
- The other version of the procedure uses a file group that can consist of multiple versions of the tablespace set in a tablespace repository. A tablespace repository is a collection of tablespace sets in a file group repository. When this version of the procedure is run, a Data Pump export is performed, and this version of the procedure completes the detach operation by placing the export dump file and export log file in the appropriate file group version. The datafiles that comprise the tablespace set are not moved or copied, but they are referenced in the version that is detached. The file group and version are specified using the `file_group_name` and `version_name` parameters, respectively. Also, if the destination platform is different, then the tablespace set is migrated automatically to the correct platform when it is attached at the destination database using the file group version of the `ATTACH_TABLESPACES` procedure.

---



---

**Note:** Do not use the `DETACH_TABLESPACES` procedure if any of the tablespaces in the tablespace set are using the Oracle-managed files feature. If you do, then the datafiles for these tablespaces are dropped automatically when the tablespaces are dropped.

---



---

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES (
    datapump_job_name      IN OUT VARCHAR2,
    tablespace_names       IN     TABLESPACE_SET,
    dump_file              IN     FILE,
    log_file               IN     FILE DEFAULT NULL,
    tablespace_files       OUT    FILE_SET);
```

```
DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES (
    tablespace_names       IN TABLESPACE_SET,
    export_directory_object IN VARCHAR2 DEFAULT NULL,
    log_file_directory_object IN VARCHAR2 DEFAULT NULL,
    file_group_name        IN VARCHAR2,
```

```

version_name          IN VARCHAR2  DEFAULT NULL,
repository_db_link    IN VARCHAR2  DEFAULT NULL);

```

## Parameters

**Table 149–8 DETACH\_TABLESPACES Procedure Parameters**

Parameter	Description
data_pump_job_name	<p>The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily.</p> <p>If NULL, then the system generates a Data Pump job name.</p>
tablespace_names	<p>The tablespace set to be detached.</p> <p>If NULL, then the procedure raises an error.</p>
dump_file	<p>The file name of the Data Pump dump file that is exported.</p> <p>If NULL or if a file attribute is NULL, then the procedure raises an error.</p> <p>If the specified file exists, then the procedure raises an error.</p>
log_file	<p>Specify the log file name for the Data Pump export.</p> <p>If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .dlg and places it in the dump file directory.</p> <p>If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.</p>
tablespace_files	<p>Contains the names of the datafiles for the detached tablespace set.</p>
export_directory_object	<p>The directory object into which the Data Pump export dump file is placed. The system generates a dump file name with the extension .dmp.</p> <p>If NULL, then procedure places the dump file in the default directory object for the version. If the version does not have a default directory object, then the procedure uses the default directory object for the file group.</p> <p>If NULL and no default directory object exists for the version or file group, then the procedure raises an error.</p>
log_file_directory_object	<p>The directory object into which the Data Pump export log file is placed. The system generates a log file name with the extension .dlg.</p> <p>If NULL, then the procedure places the export log file in the same directory as the export dump file.</p>
file_group_name	<p>The name of the file group, specified as <i>[schema_name.]file_group_name</i>. For example, if the schema is hq_dba and the file group name is sales, then specify hq_dba.sales. If the schema is not specified, then the current user is the default.</p> <p>If the specified file group does not exist, then the procedure creates it.</p>

**Table 149–8 (Cont.) DETACH\_TABLESPACES Procedure Parameters**

Parameter	Description
<code>version_name</code>	<p>The name of the version into which the detached tablespace set is placed. The specified version name cannot be a positive integer.</p> <p>If the specified version does not exist, then the procedure creates it.</p> <p>If the specified version exists, then procedure adds the tablespace set to the version. Only one Data Pump export dump file can exist in a version.</p> <p>If NULL, then the procedure creates a new version, and the version number can be used to manage the version.</p>
<code>repository_db_link</code>	<p>If the file group is in a remote database, then specify the name of the database link to the database that contains the file group. The database link must be accessible to the user who runs the procedure.</p> <p>If this parameter is non-NULL, then the directory object specified in <code>export_directory_object</code> must exist on the local database and on the remote database. If <code>export_directory_object</code> is NULL, then the default directory object must exist on both databases. The directory object must have the same name on each database and must correspond to the same directory on a shared file system.</p> <p>If NULL, then the procedure does not use a database link, and the procedure uses the file group in the local database.</p>

## Usage Notes

To run this either version of this procedure, a user must meet the following requirements:

- Have `EXP_FULL_DATABASE` role
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include `DBA_TABLESPACES` and `USER_TABLESPACES`.
- Have `DROP TABLESPACE` privilege
- Have `MANAGE TABLESPACE` or `ALTER TABLESPACE` on a tablespace if the tablespace must be made read-only
- Have `READ` privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the `tablespace_names` parameter. If a directory object does not exist for one or more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.
- Have `READ` and `WRITE` privilege on the directory object that will contain the Data Pump export dump file, specified by the `dump_file` parameter or the `export_directory_object` parameter
- Have `WRITE` privilege on the directory object that will contain the Data Pump export the log file, specified by the `log_file` parameter if non-NULL or by the `log_file_directory_object` parameter if non-NULL

If the file group version of the procedure is run, then the user must have the necessary privileges to manage the file group.



Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

After detaching a tablespace set using this procedure, you can attach the tablespaces to a different database using the `ATTACH_TABLESPACES` procedure.

**See Also:**

- [Overview](#) on page 149-3
- [ATTACH\\_TABLESPACES Procedure](#) on page 149-13
- [Chapter 67, "DBMS\\_FILE\\_GROUP"](#) for more information about file groups
- *Oracle Streams Concepts and Administration*
- *Oracle Database Administrator's Guide* for more information about the Oracle-managed files feature

## PULL\_SIMPLE\_TABLESPACE Procedure

This procedure copies a simple tablespace from a remote database and attaches it to the current database. The specified tablespace at the remote database must be online.

Specifically, this procedure performs the following actions:

1. Makes the specified tablespace read-only at the remote database if it is not read-only
2. Uses Data Pump to export the metadata for the tablespace
3. Uses a database link and the `DBMS_FILE_TRANSFER` package to transfer the datafile for the tablespace and the log file for the Data Pump export to the current database
4. Places the datafile for the specified tablespace and the log file for the Data Pump export in the specified directory at the local database
5. If this procedure made the tablespace read-only, then makes the tablespace read/write
6. Uses Data Pump to import the metadata for the tablespace in the local database

In addition, this procedure optionally can create a datafile for the tablespace that can be used with the local platform, if the platform at the remote database is different than the local database platform.

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.PULL_SIMPLE_TABLESPACE (
    tablespace_name      IN VARCHAR2,
    database_link        IN VARCHAR2,
    directory_object     IN VARCHAR2  DEFAULT NULL,
    conversion_extension IN VARCHAR2  DEFAULT NULL,
    convert_directory_object IN VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 149–9 PULL\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
<code>tablespace_name</code>	The tablespace to be pulled. If <code>NULL</code> , then the procedure raises an error.
<code>database_link</code>	The name of the database link to the database that contains the tablespace to pull. The database link must be accessible to the user who runs the procedure. If <code>NULL</code> , then the procedure raises an error.
<code>directory_object</code>	The directory object to which the datafile for the tablespace is copied on the local database. You must specify the name of a directory object created using the SQL statement <code>CREATE DIRECTORY</code> . The Data Pump import log file is written to this directory. The name of the log file is the same as the data file name for the tablespace, except with a <code>.plg</code> extension. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file. If <code>NULL</code> , then the procedure raises an error.

**Table 149–9 (Cont.) PULL\_SIMPLE\_TABLESPACE Procedure Parameters**

Parameter	Description
<code>conversion_extension</code>	Specify NULL if the platform is the same for the remote export database and the current import database.  If the platform is different for the export database and the import database, then specify an extension for the tablespace datafile that is different than the extension for the tablespace datafile at the remote database. In this case, the procedure transfers the datafile to the import database and converts it to be compatible with the current import database platform automatically. After conversion is complete, the original datafile is deleted at the import database.
<code>convert_directory_object</code>	Specify NULL if the platform is the same for the remote export database and the current import database.  If the platform is different for the export database and the import database, then specify a directory object in the local export database. The procedure uses the directory object for platform conversion before it transfers the files to the remote database. You must specify the name of a directory object created using the SQL statement <code>CREATE DIRECTORY</code> .

## Usage Notes

To run this procedure, a user must meet the following requirements on the remote database:

- Have the `EXP_FULL_DATABASE` role
- Have `EXECUTE` privilege on the `DBMS_STREAMS_TABLESPACE_ADM` package
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include `DBA_TABLESPACES` and `USER_TABLESPACES`.
- Have `MANAGE TABLESPACE` or `ALTER TABLESPACE` privilege on a tablespace if the tablespace must be made read-only
- Have `READ` privilege on the directory object for the directory that contains the datafile for the tablespace. The name of this tablespace is specified by the `tablespace_name` parameter. If a directory object does not exist for this directory, then create the directory object and grant the necessary privileges before you run this procedure.

To run this procedure, a user must meet the following requirements on the local database:

- Have the roles `IMP_FULL_DATABASE` and `EXECUTE_CATALOG_ROLE`
- Have `WRITE` privilege on the directory object that will contain the Data Pump export the log file, specified by the `log_file` parameter if non-NULL
- Have `WRITE` privilege on the directory object that will hold the datafile for the tablespace, specified by the `directory_object` parameter

Automatic Storage Management (ASM) directories cannot be used with this procedure.

**See Also:** [Overview](#) on page 149-3

## PULL\_TABLESPACES Procedure

This procedure copies a set of self-contained tablespaces from a remote database and attaches the tablespaces to the current database. All of the tablespaces in the specified tablespace set at the remote database must be online.

Specifically, this procedure performs the following actions:

1. Makes any read/write tablespace in the specified tablespace set at the remote database read-only
2. Uses Data Pump to export the metadata for the tablespaces in the tablespace set
3. Uses a database link and the `DBMS_FILE_TRANSFER` package to transfer the datafiles for the tablespace set and the log file for the Data Pump export to the current database
4. Places the datafiles that comprise the specified tablespace set in the specified directories at the local database
5. Places the log file for the Data Pump export in the specified directory at the local database
6. If this procedure made a tablespace read-only, then makes the tablespace read/write
7. Uses Data Pump to import the metadata for the tablespaces in the tablespace set at the local database

In addition, this procedure optionally can create datafiles for the tablespace set that can be used with the local platform, if the platform at the remote database is different than the local database platform.

### Syntax

```
DBMS_STREAMS_TABLESPACE_ADM.PULL_TABLESPACES (
  datapump_job_name          IN OUT VARCHAR2,
  database_link              IN      VARCHAR2,
  tablespace_names          IN      TABLESPACE_SET,
  tablespace_directory_objects IN  DIRECTORY_OBJECT_SET,
  log_file                  IN      FILE,
  conversion_extension      IN      VARCHAR2  DEFAULT NULL,
  convert_directory_object  IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 149–10 PULL\_TABLESPACES Procedure Parameters**

Parameter	Description
<code>data_pump_job_name</code>	The Data Pump job name. Specify a Data Pump job name to adhere to naming conventions or to track the job more easily.  If <code>NULL</code> , then the system generates a Data Pump job name.
<code>database_link</code>	The name of the database link to the database that contains the tablespace set to pull. The database link must be accessible to the user who runs the procedure.  If <code>NULL</code> , then the procedure raises an error.

**Table 149–10 (Cont.) PULL\_TABLESPACES Procedure Parameters**

Parameter	Description
tablespace_names	The tablespace set to be pulled. If NULL, then the procedure raises an error.
tablespace_directory_objects	The set of directory objects to which the datafiles for the tablespaces are copied. If multiple directory objects are in the set, then the procedure copies a datafile to each directory object in the set in sequence. In this case, if the end of the directory object set is reached, then datafile copying starts again with the first directory object in the set. If NULL, then the procedure raises an error.
log_file	Specify the log file name for the Data Pump export. If NULL or if at least one file parameter is NULL, then the system generates a log file name with the extension .plg and places it in one of the data file directories. If a file exists with the same name as the log file in the directory, then the procedure overwrites the file.
conversion_extension	Specify NULL if the platform is the same for the remote export database and the current import database. If the platform is different for the export database and the import database, then specify an extension for the tablespace datafiles that is different than the extension for the tablespace datafiles at the remote database. In this case, the procedure transfers the datafiles to the import database and converts them to be compatible with the current import database platform automatically. After conversion is complete, the original datafiles are deleted at the import database.
convert_directory_object	Specify NULL if the platform is the same for the remote export database and the current import database. If the platform is different for the export database and the import database, then specify a directory object in the local export database. The procedure uses the directory object for platform conversion before it transfers the files to the remote database. You must specify the name of a directory object created using the SQL statement CREATE DIRECTORY.

## Usage Notes

To run this procedure, a user must meet the following requirements on the remote database:

- Have the EXP\_FULL\_DATABASE role
- Have EXECUTE privilege on the DBMS\_STREAMS\_TABLESPACE\_ADM package
- Have access to at least one data dictionary view that contains information about the tablespaces. These views include DBA\_TABLESPACES and USER\_TABLESPACES.
- Have MANAGE TABLESPACE or ALTER TABLESPACE privilege on a tablespace if the tablespace must be made read-only
- Have READ privilege on the directory objects for the directories that contain the datafiles for the tablespace set. The names of these tablespaces are specified by the tablespace\_names parameter. If a directory object does not exist for one or

more of these directories, then create the directory objects and grant the necessary privileges before you run this procedure.

To run this procedure, a user must meet the following requirements on the local database:

- Have the roles `IMP_FULL_DATABASE` and `EXECUTE_CATALOG_ROLE`
- Have `WRITE` privilege on the directory object that will contain the Data Pump export the log file, specified by the `log_file` parameter if non-NULL
- Have `WRITE` privilege on the directory objects that will hold the datafiles for the tablespaces in the set, specified by the `tablespace_directory_objects` parameter

Automatic Storage Management (ASM) directories can be specified for the directory objects that store datafiles and export dump files, but ASM directories cannot be specified for directory objects that store log files.

**See Also:** [Overview](#) on page 149-3

The DBMS\_TDB package reports whether a database can be transported between platforms using the RMAN CONVERT DATABASE command. The package verifies that databases on the current host platform are of the same endian format as the destination platform, and that the state of the current database does not prevent transport of the database.

**See Also:** *Oracle Database Backup and Recovery User's Guide* regarding database transport using CONVERT DATABASE

This chapter contains the following topics:

- [Using DBMS\\_TDB](#)
  - Overview
  - Security Model
  - Constants
  - Views
  - Operational Notes
- [Summary of DBMS\\_TDB Subprograms](#)

## Using DBMS\_TDB

This section contains topics which relate to using DBMS\_TDB.

- [Overview](#)
- [Constants](#)
- [Views](#)
- [Operational Notes](#)



## Overview

In many cases, Oracle supports transporting databases between platforms which have the same endian format. However, even when the endian formats are the same, a database must undergo a conversion process to move from one platform to another. There are also preconditions required for the process of transporting a database, such as having the database to be transported open read-only.

The DBMS\_TDB package serves two purposes:

- Confirming that Oracle supports transporting a database from a given source platform to a given target platform
- Determining whether a database to be transported has been properly prepared for transport, and if not, identifying the condition that prevents database transport

The actual conversion is performed using the Recovery Manager `CONVERT DATABASE` command. For a complete discussion of the requirements for transporting a database, the process of converting a database for transport across platforms, and examples of the use of the DBMS\_TDB subprograms in the conversion process, see *Oracle Database Backup and Recovery User's Guide*.

## Security Model

Use of this package requires the DBA privilege.

## Constants

The DBMS\_TDB package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS\_TDB.SKIP\_NONE.

The DBMS\_TDB package uses the constants shown in [Table 150-1](#).

**Table 150-1 DBMS\_TDB Constants**

Name	Type	Value	Description
SKIP_NONE	NUMBER	0	Check all files when checking whether a database is ready for transport.
SKIP_OFFLINE	NUMBER	2	Skip files in offline tablespaces when checking whether a database is ready for transport.
SKIP_READONLY	NUMBER	3	Skip files in readonly tablespaces when checking whether a database is ready for transport.

## Views

The DBMS\_TDB package uses the following view listed in *Oracle Database Reference*:

- V\$DB\_TRANSPORTABLE\_PLATFORM, which specifies which combinations of source and target platforms support database transport

## Operational Notes

- The subprograms in this package are useful both in determining whether the desired cross-platform database conversion is possible, and in checking whether the database is ready for conversion. See *Oracle Database Backup and Recovery User's Guide* for details on the different uses of these subprograms are used in the conversion process.
- The subprograms in this package return simple `TRUE` or `FALSE` results to indicate whether database transport is possible. Use the subprograms with `SERVEROUTPUT ON` for informative messages about why transport is not possible.

## Summary of DBMS\_TDB Subprograms

**Table 150–2 DBMS\_TDB Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CHECK_DB Function</a> on page 150-9	Checks whether a database can be transported to a target platform
<a href="#">CHECK_EXTERNAL Function</a> on page 150-9	Checks whether a database has external tables, directory or BFILEs

## CHECK\_DB Function

This function checks whether a database can be transported to a target platform. It tests whether transport is supported at all for a given source and destination platform, and whether the database is currently in the correct state for transport.

You can specify whether to skip checking parts of the database that are read-only or offline, if you do not plan to transport them.

The function is overloaded. The different functionality of each form of syntax is presented along with the definition.

### Syntax

```
DBMS_TDB.CHECK_DB (
    target_platform_name  IN VARCHAR2,
    skip_option           IN NUMBER)
RETURN BOOLEAN;

DBMS_TDB.CHECK_DB (
    target_platform_name  IN VARCHAR2)
RETURN BOOLEAN;

DBMS_TDB.CHECK_DB
RETURN BOOLEAN;
```

### Parameters

**Table 150–3 CHECK\_DB Function Parameters**

Parameter	Description
target_platform_name	The name of the destination platform, as it appears in V\$DB_TRANSPORTABLE_PLATFORM.
skip_option	Specifies which, if any, parts of the database to skip when checking whether the database can be transported. Supported values are listed in <a href="#">Table 150–1, "DBMS_TDB Constants"</a> on page 150-5.

### Return Values

If the database cannot be transported to the target platform or is not ready to be transported, returns FALSE. If the database is ready for transport, returns TRUE.

### Usage Notes

- If SERVEROUTPUT is ON, then the output will contain the reasons why the database cannot be transported and how to fix the problems. For details on possible reasons and fixes, see [Table 150–4, "Reasons for CHECK\\_DB Function to Return FALSE"](#) on page 150-9.

**Table 150–4 Reasons for CHECK\_DB Function to Return FALSE**

Cause	Action
Unrecognized target platform name.	Check V\$DB_TRANSPORTABLE_PLATFORM for recognized platform names.

**Table 150–4 (Cont.) Reasons for CHECK\_DB Function to Return FALSE**

<b>Cause</b>	<b>Action</b>
Target platform has a different endian format.	Conversion is not supported.
Database is not open read-only.	Open database read-only and retry.
There are active or in-doubt transactions in the database.	Open the database read-write. After the active transactions are rolled back, open the database read-only and retry the operation. This situation can occur if users flash back the database and open it read only. The active transactions will be rolled back when the database is opened read-write.
Deferred transaction rollback needs to be done.	Open the database read-write and bring online the necessary tablespaces. Once the deferred transaction rollback is complete, open the database read-only and retry the operation.
Database compatibility version is below 10.0.0.	Change the COMPATIBLE initialization parameter to 10.0.0 or higher, open the database read-only, and retry the operation.
Some tablespaces have not been open read-write with compatibility version is 10.0.0 or higher.	Change the COMPATIBLE initialization parameter to 10.0.0 or higher, then open the affected tablespaces read-write. Shut down the database, open it read-only, and retry the operation.

## Examples

This example illustrates the use of CHECK\_DB with a database that is open read-write:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
    db_ready BOOLEAN;
BEGIN
    db_ready := DBMS_TDB.CHECK_DB('Microsoft Windows IA (32-bit)');
END;
/
```

Database is not open READ ONLY. Please open database READ ONLY and retry.

PL/SQL procedure successfully completed.



## CHECK\_EXTERNAL Function

This function determines whether a database has external tables, directories, or BFILEs.

### Syntax

```
DBMS_TDB.CHECK_EXTERNAL  
RETURN BOOLEAN;
```

### Return Values

If the database has external tables, directories, or BFILEs, return `TRUE`. Otherwise, return `FALSE`.

### Usage Notes

- If `SERVEROUTPUT` is `ON`, then the function will output the names of the external tables, directories, and BFILEs in the database.
- The database must be open read-write.

### Examples

This example illustrates the use of `CHECK_EXTERNAL` with a database that has several external tables, directories, and BFILEs:

```
SQL> SET SERVEROUTPUT ON  
SQL> DECLARE  
    external BOOLEAN;  
BEGIN  
    external := DBMS_TDB.CHECK_EXTERNAL;  
END;  
/
```

The following external tables exist in the database:

```
SH.SALES_TRANSACTIONS_EXT
```

The following directories exist in the database:

```
SYS.MEDIA_DIR, SYS.DATA_FILE_DIR, SYS.LOG_FILE_DIR, SYS.DATA_PUMP_DIR
```

The following BFILEs exist in the database:

```
PM.PRINT_MEDIA
```

PL/SQL procedure successfully completed.



The DBMS\_TRACE package contains the interface to trace PL/SQL functions, procedures, and exceptions.

This chapter contains the following topics:

- [Using DBMS\\_TRACE](#)
  - Overview
  - Security Model
  - Constants
  - Restrictions
  - Operational Notes
- [Summary of DBMS\\_TRACE Subprograms](#)

---

## Using DBMS\_TRACE

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Restrictions](#)
- [Operational Notes](#)

## Overview

DBMS\_TRACE provides subprograms to start and stop PL/SQL tracing in a session. Oracle collects the trace data as the program executes and writes it to database tables.

A typical session involves:

- Starting PL/SQL tracing in session (DBMS\_TRACE.SET\_PLSQL\_TRACE).
- Running an application to be traced.
- Stopping PL/SQL tracing in session (DBMS\_TRACE.CLEAR\_PLSQL\_TRACE).

## Security Model

This package must be created under SYS.

## Constants

The DBMS\_TRACE package uses the constants shown in [Table 151-1, "DBMS\\_TRACE Constants"](#):

**Table 151-1 DBMS\_TRACE Constants**

Name	Type	Value	Description
TRACE_MINOR_VERSION	INTEGER	0	
TRACE_MAJOR_VERSION	INTEGER	1	
TRACE_ALL_CALLS	INTEGER	1	Traces calls or returns
TRACE_ENABLED_CALLS	INTEGER	2	
TRACE_ALL_EXCEPTIONS	INTEGER	4	Traces exceptions
TRACE_ENABLED_EXCEPTIONS	INTEGER	8	Traces exceptions and handlers
TRACE_LIMIT	INTEGER	16	Save only the last few records. This allows tracing up to a problem area, without filling the database up with masses of irrelevant information. If event 10940 is set, the limit is 1023*(the value of event 10940). This can be overridden by the use of "TRACE_LIMIT" flag.
TRACE_ALL_SQL	INTEGER	32	Traces SQL statements
TRACE_ENABLED_SQL	INTEGER	64	Traces SQL statements at PL/SQL level. This does not invoke SQL Trace
TRACE_ALL_LINES	INTEGER	128	Traces each line
TRACE_ENABLED_LINES	INTEGER	256	
TRACE_PAUSE	INTEGER	4096	Pauses tracing
TRACE_RESUME	INTEGER	8192	Resume tracing
TRACE_STOP	INTEGER	16384	Stops tracing
NO_TRACE_ADMINISTRATIVE	INTEGER	32768	Prevents tracing of 'administrative events such as <ul style="list-style-type: none"> <li>■ PL/SQL Trace Tool started</li> <li>■ Trace flags changed</li> <li>■ PL/SQL Virtual Machine started</li> <li>■ PL/SQL Virtual Machine stopped</li> </ul>
NO_TRACE_HANDLED_EXCEPTIONS	INTEGER	65536	Prevents tracing of handled exceptions

Oracle recommends using the symbolic form for all these constants.

## Restrictions

You cannot use PL/SQL tracing in a shared server environment.



## Operational Notes

- [Controlling Data Volume](#)
- [Creating Database Tables to Collect DBMS\\_TRACE Output](#)
- [Collecting Trace Data](#)
- [Collected Data](#)
- [Trace Control](#)

### Controlling Data Volume

Profiling large applications may produce a large volume of data. You can control the volume of data collected by *enabling* specific program units for trace data collection.

You can enable a program unit by compiling it debug. This can be done in one of two ways:

```
alter session set plsql_debug=true;
create or replace ... /* create the library units - debug information will be
generated */
```

or:

```
/* recompile specific library unit with debug option */
alter [PROCEDURE | FUNCTION | PACKAGE BODY] <libunit-name> compile debug;
```

---

**Note:** You cannot use the second method for anonymous blocks.

---

You can limit the amount of storage used in the database by retaining only the most recent 8,192 records (approximately) by including `TRACE_LIMIT` in the `TRACE_LEVEL` parameter of the `SET_PLSQL_TRACE` procedure.

### Creating Database Tables to Collect DBMS\_TRACE Output

You must create database tables into which the `DBMS_TRACE` package writes output. Otherwise, the data is not collected. To create these tables, run the script `TRACETAB.SQL`. The tables this script creates are owned by `SYS`.

### Collecting Trace Data

The PL/SQL features you can trace are described in the script `DBMSPBT.SQL`. Some of the key tracing features are:

- [Tracing Calls](#)
- [Tracing Exceptions](#)
- [Tracing SQL](#)
- [Tracing Lines](#)

Additional features of `DBMS_TRACE` also allow pausing and resuming trace, and limiting the output.

#### Tracing Calls

Two levels of call tracing are available:

- Level 1: Trace all calls. This corresponds to the constant `TRACE_ALL_CALLS`.

- Level 2: Trace calls to enabled program units only. This corresponds to the constant `TRACE_ENABLED_CALLS`.

Enabling cannot be detected for remote procedure calls (RPCs); hence, RPCs are only traced with level 1.

### Tracing Exceptions

Two levels of exception tracing are available:

- Level 1: Trace all exceptions. This corresponds to `TRACE_ALL_EXCEPTIONS`.
- Level 2: Trace exceptions raised in enabled program units only. This corresponds to `TRACE_ENABLED_EXCEPTIONS`.

### Tracing SQL

Two levels of SQL tracing are available:

- Level 1: Trace all SQL. This corresponds to the constant `TRACE_ALL_SQL`.
- Level 2: Trace SQL in enabled program units only. This corresponds to the constant `TRACE_ENABLED_SQL`.

### Tracing Lines

Two levels of line tracing are available:

- Level 1: Trace all lines. This corresponds to the constant `TRACE_ALL_LINES`.
- Level 2: Trace lines in enabled program units only. This corresponds to the constant `TRACE_ENABLED_LINES`.

When tracing lines, Oracle adds a record to the database each time the line number changes. This includes line number changes due to procedure calls and returns.

---

---

**Note:** For all types of tracing, level 1 overrides level 2. For example, if both level 1 and level 2 are enabled, then level 1 takes precedence.

---

---

## Collected Data

If tracing is requested only for enabled program units, and if the current program unit is not enabled, then no trace data is written.

When tracing calls, both the call and return are traced. The check for whether tracing is "enabled" passes if either the called routine or the calling routine is "enabled".

Call tracing will always output the program unit type, program unit name, and line number for both the caller and the callee. It will output the caller's stack depth. If the caller's unit is enabled, the calling procedure name will also be output. If the callee's unit is enabled, the called procedure name will be output.

Exception tracing writes out the line number. Raising the exception shows information on whether the exception is user-defined or pre-defined. It also shows the exception number in the case of pre-defined exceptions. Both the place where the exceptions are raised and their handler is traced. The check for tracing being "enabled" is done independently for the place where the exception is raised and the place where the exception is handled.

All calls to `DBMS_TRACE.SET_PLSQL_TRACE` and `DBMS_TRACE.CLEAR_PLSQL_TRACE` place a special trace record in the database. Therefore, it is always possible to determine when trace settings were changed.

## Trace Control

As well as determining which items are collected, you can pause and resume the trace process. No information is gathered between the time that tracing is paused and the time that it is resumed. The constants `TRACE_PAUSE` and `TRACE_RESUME` are used to accomplish this. Trace records are generated to indicate that the trace was paused/resumed.

It is also possible to retain only the last 8,192 trace events of a run by using the constant `TRACE_LIMIT`. This allows tracing to be turned on without filling up the database. When tracing stops, the last 8,192 records are saved. The limit is approximate, since it is not checked on every trace record. At least the requested number of trace records will be generated; up to 1,000 additional records may be generated. At least the requested number of trace records will be generated; up to 1,000 additional records may be generated. The 8,192 record limit can be changed. Setting event 10940 to level  $n$  changes the record limit to  $1024 * n$ .

## Summary of DBMS\_TRACE Subprograms

**Table 151–2 DBMS\_TRACE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CLEAR_PLSQL_TRACE Procedure</a> on page 151-11	Stops trace data dumping in session
<a href="#">GET_PLSQL_TRACE_LEVEL Function</a> on page 151-12	Gets the trace level
<a href="#">PLSQL_TRACE_VERSION Procedure</a> on page 151-13	Gets the version number of the trace package
<a href="#">SET_PLSQL_TRACE Procedure</a> on page 151-14	Starts tracing in the current session

## **CLEAR\_PLSQL\_TRACE Procedure**

This procedure disables trace data collection.

### **Syntax**

```
DBMS_TRACE.CLEAR_PLSQL_TRACE;
```

## GET\_PLSQL\_TRACE\_LEVEL Function

This procedure returns the current trace level as a list of the enabled constants.

### Syntax

```
DBMS_TRACE.GET_PLSQL_TRACE_LEVEL  
RETURN BINARY_INTEGER;
```

## PLSQL\_TRACE\_VERSION Procedure

This procedure gets the version number of the trace package. It returns the major and minor version number of the DBMS\_TRACE package.

### Syntax

```
DBMS_TRACE.PLSQL_TRACE_VERSION (  
    major OUT BINARY_INTEGER,  
    minor OUT BINARY_INTEGER);
```

### Parameters

**Table 151-3 PLSQL\_TRACE\_VERSION Procedure Parameters**

Parameter	Description
major	Major version number of DBMS_TRACE.
minor	Minor version number of DBMS_TRACE.

## SET\_PLSQL\_TRACE Procedure

This procedure enables PL/SQL trace data collection.

### Syntax

```
DBMS_TRACE.SET_PLSQL_TRACE (  
    trace_level INTEGER);
```

### Parameters

**Table 151–4** SET\_PLSQL\_TRACE Procedure Parameters

Parameter	Description
trace_level	You must supply one or more of the constants as listed in <a href="#">Table 151–1</a> , "DBMS_TRACE Constants". By summing the constants, you can enable tracing of multiple PL/SQL language features simultaneously. The control constants "TRACE_PAUSE", "TRACE_RESUME" and "TRACE_STOP" should not be used in combination with other constants  Also see " <a href="#">Collecting Trace Data</a> " on page 151-7 for more information.



---

---

## DBMS\_TRANSACTION

The DBMS\_TRANSACTION package provides access to SQL transaction statements from stored procedures.

**See Also:** *Oracle Database SQL Language Reference*

This chapter contains the following topics:

- [Using DBMS\\_TRANSACTION](#)
  - Security Model
- [Summary of DBMS\\_TRANSACTION Subprograms](#)

## Using DBMS\_TRANSACTION

- [Security Model](#)

## Security Model

This package runs with the privileges of calling user, rather than the package owner SYS.

---

## Summary of DBMS\_TRANSACTION Subprograms

**Table 152–1 DBMS\_TRANSACTION Package Subprograms**

Subprogram	Description
<a href="#">ADVISE_COMMIT Procedure</a> on page 152-5	Equivalent to the SQL statement: ALTER SESSION ADVISE COMMIT
<a href="#">ADVISE_NOTHING Procedure</a> on page 152-6	Equivalent to the SQL statement: ALTER SESSION ADVISE NOTHING
<a href="#">ADVISE_ROLLBACK Procedure</a> on page 152-7	Equivalent to the SQL statement: ALTER SESSION ADVISE ROLLBACK
<a href="#">COMMIT Procedure</a> on page 152-8	Equivalent to the SQL statement: COMMIT
<a href="#">COMMIT_COMMENT Procedure</a> on page 152-9	Equivalent to the SQL statement: COMMIT COMMENT <text>
<a href="#">COMMIT_FORCE Procedure</a> on page 152-10	Equivalent to the SQL statement: COMMIT FORCE <text>, <number>"
<a href="#">LOCAL_TRANSACTION_ID Function</a> on page 152-11	Returns the local (to instance) unique identifier for the current transaction
<a href="#">PURGE_LOST_DB_ENTRY Procedure</a> on page 152-12	Enables removal of incomplete transactions from the local site when the remote database is destroyed or re-created before recovery completes
<a href="#">PURGE_MIXED Procedure</a> on page 152-14	Deletes information about a given mixed outcome transaction
<a href="#">READ_ONLY Procedure</a> on page 152-15	Equivalent to the SQL statement: SET TRANSACTION READ ONLY
<a href="#">READ_WRITE Procedure</a> on page 152-16	equivalent to the SQL statement: SET TRANSACTION READ WRITE
<a href="#">ROLLBACK Procedure</a> on page 152-17	Equivalent to the SQL statement: ROLLBACK
<a href="#">ROLLBACK_FORCE Procedure</a> on page 152-18	Equivalent to the SQL statement: ROLLBACK FORCE <text>
<a href="#">ROLLBACK_SAVEPOINT Procedure</a> on page 152-19	Equivalent to the SQL statement: ROLLBACK TO SAVEPOINT <savepoint_name>
<a href="#">SAVEPOINT Procedure</a> on page 152-20	Equivalent to the SQL statement: SAVEPOINT <savepoint_name>
<a href="#">STEP_ID Function</a> on page 152-21	Returns local (to local transaction) unique positive integer that orders the DML operations of a transaction
<a href="#">USE_ROLLBACK_SEGMENT Procedure</a> on page 152-22	Equivalent to the SQL statement: SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>

## **ADVISE\_COMMIT Procedure**

This procedure is equivalent to the SQL statement:

```
ALTER SESSION ADVISE COMMIT
```

### **Syntax**

```
DBMS_TRANSACTION.ADVISE_COMMIT;
```

## ADVISE\_NOTHING Procedure

This procedure is equivalent to the SQL statement:

```
ALTER SESSION ADVISE NOTHING
```

### Syntax

```
DBMS_TRANSACTION.ADVISE_NOTHING;
```

## **ADVISE\_ROLLBACK Procedure**

This procedure is equivalent to the SQL statement:

```
ALTER SESSION ADVISE ROLLBACK
```

### **Syntax**

```
DBMS_TRANSACTION.ADVISE_ROLLBACK;
```

## COMMIT Procedure

This procedure is equivalent to the SQL statement:

```
COMMIT
```

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

### Syntax

```
DBMS_TRANSACTION.COMMIT;
```



## COMMIT\_COMMENT Procedure

This procedure is equivalent to the SQL statement:

```
COMMIT COMMENT <text>
```

### Syntax

```
DBMS_TRANSACTION.COMMIT_COMMENT (  
    cmnt VARCHAR2);
```

### Parameters

**Table 152–2** COMMIT\_COMMENT Procedure Parameters

Parameter	Description
cmnt	Comment to associate with this commit.

## COMMIT\_FORCE Procedure

This procedure is equivalent to the SQL statement:

```
COMMIT FORCE <text>, <number>"
```

### Syntax

```
DBMS_TRANSACTION.COMMIT_FORCE (  
    xid VARCHAR2,  
    scn VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 152–3 COMMIT\_FORCE Procedure Parameters**

Parameter	Description
xid	Local or global transaction ID.
scn	System change number.

## LOCAL\_TRANSACTION\_ID Function

This function returns the local (to instance) unique identifier for the current transaction. It returns null if there is no current transaction.

### Syntax

```
DBMS_TRANSACTION.LOCAL_TRANSACTION_ID (  
    create_transaction BOOLEAN := FALSE)  
RETURN VARCHAR2;
```

### Parameters

**Table 152-4 LOCAL\_TRANSACTION\_ID Function Parameters**

Parameter	Description
create_transaction	If true, then start a transaction if one is not currently active.

## PURGE\_LOST\_DB\_ENTRY Procedure

When a failure occurs during commit processing, automatic recovery consistently resolves the results at all sites involved in the transaction. However, if the remote database is destroyed or re-created before recovery completes, then the entries used to control recovery in `DBA_2PC_PENDING` and associated tables are never removed, and recovery will periodically retry. Procedure `PURGE_LOST_DB_ENTRY` enables removal of such transactions from the local site.

### Syntax

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY (
    xid VARCHAR2);
```

### Parameters

**Table 152–5** *PURGE\_LOST\_DB\_ENTRY Procedure Parameters*

Parameter	Description
<code>xid</code>	Must be set to the value of the <code>LOCAL_TRAN_ID</code> column in the <code>DBA_2PC_PENDING</code> table.

### Usage Notes

---



---

**WARNING:** `PURGE_LOST_DB_ENTRY` should *only* be used when the other database is lost or has been re-created. Any other use may leave the other database in an unrecoverable or inconsistent state.

---



---

Before automatic recovery runs, the transaction may show up in `DBA_2PC_PENDING` as state "collecting", "committed", or "prepared". If the DBA has forced an in-doubt transaction to have a particular result by using "commit force" or "rollback force", then states "forced commit" or "forced rollback" may also appear. Automatic recovery normally deletes entries in any of these states. The only exception is when recovery finds a forced transaction which is in a state inconsistent with other sites in the transaction; in this case, the entry is left in the table and the `MIXED` column has the value 'yes'.

However, under certain conditions, it may not be possible for automatic recovery to run. For example, a remote database may have been permanently lost. Even if it is re-created, it gets a new database ID, so that recovery cannot identify it (a possible symptom is `ORA-02062`). In this case, the DBA may use the procedure `PURGE_LOST_DB_ENTRY` to clean up the entries in any state other than "prepared". The DBA does not need to be in any particular hurry to resolve these entries, because they are not holding any database resources.

The following table indicates what the various states indicate about the transaction and what the DBA actions should be:

**Table 152–6 PURGE\_LOST\_DB\_ENTRY Procedure States**

<b>State of Column</b>	<b>State of Global Transaction</b>	<b>State of Local Transaction</b>	<b>Normal DBA Action</b>	<b>Alternative DBA Action</b>
Collecting	Rolled back	Rolled back	None	PURGE_LOST_DB_ENTRY (See Note 1)
Committed	Committed	Committed	None	PURGE_LOST_DB_ENTRY (See Note 1)
Prepared	Unknown	Prepared	None	FORCE COMMIT or ROLLBACK
Forced commit	Unknown	Committed	None	PURGE_LOST_DB_ENTRY (See Note 1)
Forced rollback	Unknown	Rolled back	None	PURGE_LOST_DB_ENTRY (See Note 1)
Forced commit (mixed)	Mixed	Committed	(See Note 2)	
Forced rollback (mixed)	Mixed	Rolled back	(See Note 2)	

---

**NOTE 1:** Use only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples are total loss of the remote database, reconfiguration in software resulting in loss of two-phase commit capability, or loss of information from an external transaction coordinator such as a TP monitor.

---



---

**NOTE 2:** Examine and take any manual action to remove inconsistencies; then use the procedure PURGE\_MIXED.

---

## PURGE\_MIXED Procedure

When in-doubt transactions are forced to commit or rollback (instead of letting automatic recovery resolve their outcomes), there is a possibility that a transaction can have a mixed outcome: Some sites commit, and others rollback. Such inconsistency cannot be resolved automatically by Oracle; however, Oracle flags entries in `DBA_2PC_PENDING` by setting the `MIXED` column to a value of 'yes'.

Oracle never automatically deletes information about a mixed outcome transaction. When the application or DBA is certain that all inconsistencies that might have arisen as a result of the mixed transaction have been resolved, this procedure can be used to delete the information about a given mixed outcome transaction.

### Syntax

```
DBMS_TRANSACTION.PURGE_MIXED (  
    xid VARCHAR2);
```

### Parameters

**Table 152–7** *PURGE\_MIXED Procedure Parameters*

Parameter	Description
<code>xid</code>	Must be set to the value of the <code>LOCAL_TRAN_ID</code> column in the <code>DBA_2PC_PENDING</code> table.

## **READ\_ONLY Procedure**

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION READ ONLY
```

### **Syntax**

```
DBMS_TRANSACTION.READ_ONLY;
```

## **READ\_WRITE Procedure**

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION READ WRITE
```

### **Syntax**

```
DBMS_TRANSACTION.READ_WRITE;
```



## ROLLBACK Procedure

This procedure is equivalent to the SQL statement:

```
ROLLBACK
```

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

### Syntax

```
DBMS_TRANSACTION.ROLLBACK;
```

## ROLLBACK\_FORCE Procedure

This procedure is equivalent to the SQL statement:

```
ROLLBACK FORCE <text>
```

### Syntax

```
DBMS_TRANSACTION.ROLLBACK_FORCE (  
  xid VARCHAR2);
```

### Parameters

**Table 152–8** *ROLLBACK\_FORCE Procedure Parameters*

Parameter	Description
<code>xid</code>	Local or global transaction ID.

## ROLLBACK\_SAVEPOINT Procedure

This procedure is equivalent to the SQL statement:

```
ROLLBACK TO SAVEPOINT <savepoint_name>
```

This procedure is included for completeness, the functionality being already implemented as part of PL/SQL.

### Syntax

```
DBMS_TRANSACTION.ROLLBACK_SAVEPOINT (  
    savept VARCHAR2);
```

### Parameters

**Table 152–9 ROLLBACK\_SAVEPOINT Procedure Parameters**

Parameter	Description
savept	Savepoint identifier.

## SAVEPOINT Procedure

This procedure is equivalent to the SQL statement:

```
SAVEPOINT <savepoint_name>
```

This procedure is included for completeness, the feature being already implemented as part of PL/SQL.

### Syntax

```
DBMS_TRANSACTION.SAVEPOINT (  
    savept VARCHAR2);
```

### Parameters

**Table 152–10** *SAVEPOINT Procedure Parameters*

Parameter	Description
savept	Savepoint identifier.

## STEP\_ID Function

This function returns local (to local transaction) unique positive integer that orders the DML operations of a transaction.

### Syntax

```
DBMS_TRANSACTION.STEP_ID  
RETURN NUMBER;
```

## USE\_ROLLBACK\_SEGMENT Procedure

This procedure is equivalent to the SQL statement:

```
SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>
```

### Syntax

```
DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT (  
    rb_name VARCHAR2);
```

### Parameters

**Table 152–11** *USE\_ROLLBACK\_SEGMENT Procedure Parameters*

Parameter	Description
rb_name	Name of rollback segment to use.

---

---

## DBMS\_TRANSFORM

The DBMS\_TRANSFORM package provides an interface to the message format transformation features of Oracle Advanced Queuing.

**See Also:** *Oracle Streams Advanced Queuing User's Guide* for more on message format transformations.

This chapter contains the following topic:

- [Summary of DBMS\\_TRANSFORM Subprograms](#)

## Summary of DBMS\_TRANSFORM Subprograms

**Table 153–1 DBMS\_TRANSFORM Package Subprograms**

<b>Subprograms</b>	<b>Description</b>
<a href="#">CREATE_TRANSFORMATION Procedure</a> on page 153-3	Creates a transformation that maps an object of the source type to an object of the destination type
<a href="#">DROP_TRANSFORMATION Procedure</a> on page 153-5	Drops the given transformation
<a href="#">MODIFY_TRANSFORMATION Procedure</a> on page 153-6	Modifies an existing transformation



## CREATE\_TRANSFORMATION Procedure

This procedure creates a transformation that maps an object of the source type to an object of the target type. The transformation expression can be a SQL expression or a PL/SQL function. It must return an object of the target type.

### Syntax

```
DBMS_TRANSFORM.CREATE_TRANSFORMATION (
  schema          VARCHAR2 (30) ,
  name            VARCHAR2 (30) ,
  from_schema     VARCHAR2 (30) ,
  from_type       VARCHAR2 (30) ,
  to_schema       VARCHAR2 (30) ,
  to_type         VARCHAR2 (30) ,
  transformation   VARCHAR2 (4000)) ;
```

### Parameters

**Table 153–2 CREATE\_TRANSFORMATION Procedure Parameters**

Parameter	Description
schema	Specifies the schema of the transformation.
name	Specifies the name of the transformation.
from_schema	Specifies the schema of the source type.
from_type	Specifies the source type.
to_schema	Specifies the target type schema.
to_type	Specifies the target type.
transformation	Specifies the transformation expression, returning an object of the target type. The expression must be a function returning an object of the target type or a constructor expression for the target type. You can choose not to specify a transformation expression and instead specify transformations for attributes of the target type using MODIFY_TRANSFORMATION.

### Usage Notes

- The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type.
- To create, modify or drop transformations, a user must be granted execute privileges on DBMS\_TRANSFORM. The user must also have execute privileges on the user defined types that are the source and destination types of the transformation. In addition, the user must also have execute privileges on any PLSQL function being used in the transformation function.
- The transformation cannot write database state (perform DML) or commit or rollback the current transaction.
- The transformation must be a SQL function with source type as input type, returning an object of the target type. It could also be a SQL expression of target type, referring to a source type. All references to the source type must be of the form *source.user\_data*.

- Both source and target types must be non-scalar database types. A null transformation expression maps to a null target object.

For using the transformation at enqueue and dequeue time, the login user invoking the operation must have execute privileges on the PLSQL functions used by the transformation. For propagation, the owning schema of the queue must have these privileges.

## DROP\_TRANSFORMATION Procedure

This procedure drops the given transformation.

### Syntax

```
DBMS_TRANSFORM.DROP_TRANSFORMATION (
  schema  VARCHAR2(30),
  name    VARCHAR2(30) );
```

### Parameters

**Table 153–3** *DROP\_TRANSFORMATION Procedure Parameters*

Parameter	Description
schema	Specifies the schema of the transformation.
name	Specifies the name of the transformation.

## MODIFY\_TRANSFORMATION Procedure

This procedure modifies the transformation expression for the given transformation.

### Syntax

```
DBMS_TRANSFORM.MODIFY_TRANSFORMATION (
  schema          VARCHAR2(30),
  name            VARCHAR2(30),
  attribute_number INTEGER,
  transformation  VARCHAR2(4000));
```

### Parameters

**Table 153–4** MODIFY\_TRANSFORMATION Procedure Parameters

Parameter	Description
schema	Specifies the schema of the transformation.
name	Specifies the name of the transformation.
attribute_number	The attribute of the target type for which the new transformation expression is being specified. When specifying the new transformation as a single expression of the target type, specify a value of 0.
transformation	The transformation expression must be a SQL expression or a PL/SQL function returning the type of the specified attribute of the target type. If the attribute_number is 0, then the expression must be a PL/SQL function returning an object of the target type or a constructor expression for the target type.

### Usage Notes

- If the new transformation is a single expression of the target type, it may be specified with an `attribute_number` of 0. The new transformation may also be specified for each attribute of the target type.
- You can use this procedure to define the transformation as a separate expression for each attribute of the target type. For large transformations, this representation may be more readable and allow the application of fine grain control over the transformation. If the transformation expression was left unspecified for some of the attributes of the target type, they are evaluated to null when the transformation is applied.

The DBMS\_TTS package checks if the transportable set is self-contained. All violations are inserted into a temporary table that can be selected from the view `TRANSPORT_SET_VIOLATIONS`.

**See Also:**

- *Oracle Database Administrator's Guide*
- *Oracle Database Upgrade Guide*

This chapter contains the following topics:

- [Using DBMS\\_TTS](#)
  - Security Model
  - Exceptions
  - Operational Notes
- [Summary of DBMS\\_TTS Subprograms](#)

## Using DBMS\_TTS

- [Security Model](#)
- [Exceptions](#)
- [Operational Notes](#)

## Security Model

Only users having the `execute_catalog_role` can execute this procedure. This role is initially only assigned to user `SYS`.

## Exceptions

```
ts_not_found EXCEPTION;  
PRAGMA exception_init(ts_not_found, -29304);  
ts_not_found_num NUMBER := -29304;  
  
invalid_ts_list EXCEPTION;  
PRAGMA exception_init(invalid_ts_list, -29346);  
invalid_ts_list_num NUMBER := -29346;  
  
sys_or_tmp_ts EXCEPTION;  
PRAGMA exception_init(sys_or_tmp_ts, -29351);  
sys_or_tmp_ts_num NUMBER := -29351;
```



## Operational Notes

With respect to transportable tablespaces, disabled and enabled referential integrity constraints are handled differently:

- A disabled referential integrity constraint does not violate the transportability rules and is dropped during the import phase.
- An enabled referential integrity constraint violates the transportability rules if it references a table in a tablespace outside the transportable set.

## Summary of DBMS\_TTS Subprograms

These two procedures are designed to be called by database administrators.

**Table 154–1 DBMS\_TTS Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DOWNGRADE Procedure</a> on page 154-7	Downgrades transportable tablespace related data
<a href="#">TRANSPORT_SET_CHECK Procedure</a> on page 154-8	Checks if a set of tablespaces (to be transported) is self-contained

## **DOWNGRADE Procedure**

This procedure downgrades transportable tablespace related data.

### **Syntax**

```
DBMS_TTS.DOWNGRADE;
```

## TRANSPORT\_SET\_CHECK Procedure

This procedure checks if a set of tablespaces (to be transported) is self-contained. After calling this procedure, the user may select from a view to see a list of violations, if there are any.

### Syntax

```
DBMS_TTS.TRANSPORT_SET_CHECK (
    ts_list          IN CLOB,
    incl_constraints IN BOOLEAN DEFAULT FALSE,
    full_check       IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 154–2 TRANSPORT\_SET\_CHECK Procedure Parameters**

Parameter	Description
ts_list	List of tablespace, separated by comma.
incl_constraints	TRUE if you want to count in referential integrity constraints when examining if the set of tablespaces is self-contained. (The incl_constraints parameter is a default so that TRANSPORT_SET_CHECK will work if it is called with only the ts_list argument.)
full_check	Indicates whether a full or partial dependency check is required. If TRUE, treats all IN and OUT pointers (dependencies) and captures them as violations if they are not self-contained in the transportable set. The parameter should be set to TRUE for TSPITR or if a strict version of transportable is desired. By default the parameter is set to false. It will only consider OUT pointers as violations.

### Examples

If the view does not return any rows, then the set of tablespaces is self-contained. For example,

```
SQLPLUS> EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('foo,bar', TRUE);
SQLPLUS> SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```

The `DBMS_TYPES` package consists of constants, which represent the built-in and user-defined types.

This chapter contains the following topics:

- [Using DBMS\\_TYPES](#)
  - Constants
  - Exceptions

## Using DBMS\_TYPES

- [Constants](#)
- [Exceptions](#)

## Constants

The following table lists the constants in the DBMS\_TYPES package.

**Table 155–1 DBMS\_TYPES Constants**

Constant	Description
NO_DATA	Is only relevant if PieceWise is called, for a collection or anydataset. Denotes the end of collection/anydataset when all the elements have been accessed
SUCCESS	The operation succeeded
TYPECODE_BDOUBLE	A NUMBER type
TYPECODE_BFILE	A BFILE type
TYPECODE_BFLOAT	A NUMBER type
TYPECODE_BLOB	A BLOB type
TYPECODE_CFILE	A CFILE type
TYPECODE_CHAR	A CHAR type
TYPECODE_CLOB	A CLOB type
TYPECODE_DATE	A DATE type
TYPECODE_INTERVAL_DS	An INTERVAL_DS type
TYPECODE_INTERVAL_YM	A INTERVAL_YM type
TYPECODE_MLSLABEL	An MLSLABEL type
TYPECODE_NAMEDCOLLECTION	A named collection (VARRAY/nested table) type
TYPECODE_NCHAR	A NCHAR type
TYPECODE_NCLOB	A NCLOB type
TYPECODE_NUMBER	A NUMBER type
TYPECODE_NVARCHAR2	A NVARCHAR2 type
TYPECODE_OBJECT	An OBJECT type
TYPECODE_OPAQUE	An OPAQUE type
TYPECODE_RAW	A RAW type
TYPECODE_REF	A REF type
TYPECODE_TABLE	A nested table collection type
TYPECODE_TIMESTAMP	A TIMESTAMP type
TYPECODE_TIMESTAMP_LTZ	A TIMESTAMP_LTZ type
TYPECODE_TIMESTAMP_TZ	A TIMESTAMP_TZ type
TYPECODE_UROWID	A UROWID type
TYPECODE_VARCHAR2	A VARCHAR2 type
TYPECODE_VARCHAR	A VARCHAR type
TYPECODE_VARRAY	A VARRAY collection type

## Exceptions

- INVALID\_PARAMETERS
- INCORRECT\_USAGE
- TYPE\_MISMATCH



The DBMS\_UTILITY package provides various utility subprograms.

This chapter contains the following topics:

- [Using DBMS\\_UTILITY](#)
  - Security Model
  - Constants
  - Exceptions
- [Data Structures](#)
  - Record Types
  - Table Types
- [Summary of DBMS\\_UTILITY Subprograms](#)

---

## Using DBMS\_UTILITY

- [Security Model](#)
- [Constants](#)
- [Exceptions](#)

## Security Model

DBMS\_UTILITY runs with the privileges of the calling user for the [NAME\\_RESOLVE Procedure](#) and the [COMPILE\\_SCHEMA Procedure](#). This is necessary so that the SQL works correctly.

The package does not run as SYS. The privileges are checked using DBMS\_DDL.

## Constants

The `DBMS_UTILITY` package uses the constants shown in [Table 156-1, "DBMS\\_UTILITY Constants"](#).

**Table 156-1** *DBMS\_UTILITY Constants*

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
<code>INV_ERROR_ON_RESTRICTIONS</code>	<code>PLS_INTEGER</code>	1	This constant is the only legal value for the <code>p_option_flags</code> parameter of the <code>INVALIDATE</code> subprogram

## Exceptions

The following table lists the exceptions raised by DBMS\_UTILITY.

**Table 156–2 Exceptions Raised by DBMS\_UTILITY**

Exception	Error Code	Description
INV_NOT_EXIST_OR_NO_PRIV	-24237	Raised by the INVALIDATE subprogram when the <code>object_id</code> argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated
INV_MALFORMED_SETTINGS	-24238	Raised by the INVALIDATE subprogram if a compiler setting is specified more than once in the <code>p_plsql_object_settings</code> parameter
INV_RESTRICTED_OBJECT	-24239	Raised by the INVALIDATE subprogram when different combinations of conditions pertaining to the <code>p_object_id</code> parameter are contravened

## Data Structures

The `DBMS_UTILITY` package defines a single `RECORD` type and `TABLE` types.

### Record Types

- `INSTANCE_RECORD` Record Type

### Table Types

- `DBLINK_ARRAY` TABLE Type
- `INDEX_TABLE_TYPE` Table Type
- `INSTANCE_TABLE` Table Type
- `LNAME_ARRAY` Table Type
- `NAME_ARRAY` Table Type
- `NUMBER_ARRAY` Table Type
- `UNCL_ARRAY` Table Type

## INSTANCE\_RECORD Record Type

This type describes a list of active instance number-name pairs.

### Syntax

```
TYPE INSTANCE_RECORD IS RECORD (  
    inst_number    NUMBER,  
    inst_name      VARCHAR2(60));
```

### Fields

**Table 156-3** *INSTANCE\_RECORD Record Type Fields*

Field	Description
inst_number	Active instance number
inst_name	Instance name

## DBLINK\_ARRAY TABLE Type

This type stores a list of database links.

### Syntax

```
TYPE DBLINK_ARRAY IS TABLE OF VARCHAR2(128) INDEX BY BINARY_INTEGER;
```



## **INDEX\_TABLE\_TYPE Table Type**

This type describes the order in which generated objects are returned to a user.

### **Syntax**

```
TYPE INDEX_TABLE_TYPE IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

## INSTANCE\_TABLE Table Type

This type describes a table of [INSTANCE\\_RECORD Record Type](#).

### Syntax

```
TYPE INSTANCE_TABLE IS TABLE OF INSTANCE_RECORD INDEX BY BINARY_INTEGER;
```

### Usage Notes

The starting index of `INSTANCE_TABLE` is 1; `INSTANCE_TABLE` is dense.

## LNAME\_ARRAY Table Type

This type stores lists of LONG NAME including fully qualified attribute names.

### Syntax

```
TYPE LNAME_ARRAY IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

## NAME\_ARRAY Table Type

This type stores lists of NAME.

### Syntax

```
TYPE NAME_ARRAY IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

## NUMBER\_ARRAY Table Type

This type describes the order in which generated objects are returned to users.

### Syntax

```
TYPE NUMBER_ARRAY IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

## UNCL\_ARRAY Table Type

This type stores lists of "user"."name"."column"@link

### Syntax

```
TYPE UNCL_ARRAY IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```

## Summary of DBMS\_UTILITY Subprograms

**Table 156–4 DBMS\_UTILITY Package Subprograms**

Subprogram	Description
<a href="#">ACTIVE_INSTANCES Procedure</a> on page 156-17	Returns the active instance
<a href="#">ANALYZE_DATABASE Procedure</a> on page 156-18	Analyzes all the tables, clusters and indexes in a database
<a href="#">ANALYZE_PART_OBJECT Procedure</a> on page 156-19	Analyzes the given tables and indexes
<a href="#">ANALYZE_SCHEMA Procedure</a> on page 156-20	Analyzes all the tables, clusters and indexes in a schema
<a href="#">CANONICALIZE Procedure</a> on page 156-21	Canonicalizes a given string
<a href="#">COMMA_TO_TABLE Procedures</a> on page 156-22	Converts a comma-delimited list of names into a PL/SQL table of names
<a href="#">COMPILE_SCHEMA Procedure</a> on page 156-23	Compiles all procedures, functions, packages, views and triggers in the specified schema
<a href="#">CREATE_ALTER_TYPE_ERROR_TABLE Procedure</a> on page 156-24	Creates an error table to be used in the EXCEPTION clause of the ALTER TYPE statement
<a href="#">CURRENT_INSTANCE Function</a> on page 156-25	Returns the current connected instance number
<a href="#">DATA_BLOCK_ADDRESS_BLOCK Function</a> on page 156-26	Gets the block number part of a data block address
<a href="#">DATA_BLOCK_ADDRESS_FILE Function</a> on page 156-27	Gets the file number part of a data block address
<a href="#">DB_VERSION Procedure</a> on page 156-28	Returns version information for the database
<a href="#">EXEC_DDL_STATEMENT Procedure</a> on page 156-29	Executes the DDL statement in <code>parse_string</code>
<a href="#">FORMAT_CALL_STACK Function</a> on page 156-30	Formats the current call stack
<a href="#">FORMAT_ERROR_BACKTRACE Function</a> on page 156-31	Formats the backtrace from the point of the current error to the exception handler where the error has been caught
<a href="#">FORMAT_ERROR_STACK Function</a> on page 156-34	Formats the current error stack
<a href="#">GET_CPU_TIME Function</a> on page 156-35	Returns the current CPU time in 100th's of a second
<a href="#">GET_DEPENDENCY Procedure</a> on page 156-36	Shows the dependencies on the object passed in.
<a href="#">GET_ENDIANNESNESS Function</a> on page 156-36	Gets the endianness of the database platform
<a href="#">GET_HASH_VALUE Function</a> on page 156-38	Computes a hash value for the given string
<a href="#">GET_PARAMETER_VALUE Function</a> on page 156-39	Gets the value of specified init.ora parameter

**Table 156–4 (Cont.) DBMS\_UTILITY Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_SQL_HASH Function</a> on page 156-41	Computes a hash value for the given string using MD5 algorithm
<a href="#">GET_TIME Function</a> on page 156-42	Returns the current time in 100th's of a second
<a href="#">GET_TZ_TRANSITIONS Procedure</a> on page 156-43	Returns timezeone transitions by regionid from the <code>timezone.dat</code> file
<a href="#">INVALIDATE Procedure</a> on page 156-44	Invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings
<a href="#">IS_BIT_SET Function</a> on page 156-47	Checks the bit setting for the given bit in the given RAW value
<a href="#">IS_CLUSTER_DATABASE Function</a> on page 156-48	Checks if the database is running in cluster database mode
<a href="#">MAKE_DATA_BLOCK_ADDRESS Function</a> on page 156-49	Creates a data block address given a file number and a block number
<a href="#">NAME_RESOLVE Procedure</a> on page 156-50	Resolves the given name
<a href="#">NAME_TOKENIZE Procedure</a> on page 156-52	Calls the parser to parse the given name
<a href="#">OLD_CURRENT_SCHEMA Function</a> on page 156-53	Returns the session value from SYS_CONTEXT ('USERENV', 'CURRENT_SCHEMA')
<a href="#">OLD_CURRENT_USER Function</a> on page 156-54	Returns the session value from SYS_CONTEXT ('USERENV', 'CURRENT_USER')
<a href="#">PORT_STRING Function</a> on page 156-55	Returns a string that uniquely identifies the version of Oracle and the operating system
<a href="#">SQLID_TO_SQLHASH Function</a> on page 156-56	Converts a SQL ID into a hash value
<a href="#">TABLE_TO_COMMA Procedures</a> on page 156-57	Converts a PL/SQL table of names into a comma-delimited list of names
<a href="#">VALIDATE Procedure</a> on page 156-58	Makes invalid database objects valid
<a href="#">WAIT_ON_PENDING_DML Function</a> on page 156-60	Waits until all transactions (other than the caller's own) that have locks on the listed tables and began prior to the specified SCN have either committed or been rolled back



## ACTIVE\_INSTANCES Procedure

This procedure returns the active instance.

### Syntax

```
DBMS_UTILITY.ACTIVE_INSTANCES (  
    instance_table  OUT INSTANCE_TABLE,  
    instance_count  OUT NUMBER);
```

### Parameters

**Table 156-5 ACTIVE\_INSTANCES Procedure Parameters**

Procedure	Description
instance_table	Contains a list of the active instance numbers and names. When no instance is up, the list is empty.
instance_count	Number of active instances

## ANALYZE\_DATABASE Procedure

This procedure analyzes all the tables, clusters and indexes in a database.

### Syntax

```
DBMS_UTILITY.ANALYZE_DATABASE (
  method          IN  VARCHAR2,
  estimate_rows   IN  NUMBER DEFAULT NULL,
  estimate_percent IN  NUMBER DEFAULT NULL,
  method_opt      IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 156–6 ANALYZE\_DATABASE Procedure Parameters**

Parameter	Description
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified ignore this parameter.
method_opt	Method options of the following format: [ FOR TABLE ] [ FOR ALL [INDEXED] COLUMNS] [SIZE n] [ FOR ALL INDEXES ]

### Exceptions

Ora-20000: Insufficient privileges for some object in this database

## ANALYZE\_PART\_OBJECT Procedure

This procedure is equivalent to SQL:

```
"ANALYZE TABLE|INDEX [<schema>.]<object_name> PARTITION <pname> [<command_type>]
 [<command_opt>] [<sample_clause>]"
```

### Syntax

```
DBMS_UTILITY.ANALYZE_PART_OBJECT (
  schema          IN VARCHAR2 DEFAULT NULL,
  object_name     IN VARCHAR2 DEFAULT NULL,
  object_type     IN CHAR      DEFAULT 'T',
  command_type    IN CHAR      DEFAULT 'E',
  command_opt     IN VARCHAR2 DEFAULT NULL,
  sample_clause   IN VARCHAR2 DEFAULT 'sample 5 percent ');
```

### Parameters

**Table 156–7 ANALYZE\_PART\_OBJECT Procedure Parameters**

Parameter	Description
schema	Schema of the object_name
object_name	Name of object to be analyzed, must be partitioned
object_type	Type of object, must be T (table) or I (index)
command_type	Must be V (validate structure)
command_opt	Other options for the command type. For C, E it can be FOR table, FOR all LOCAL indexes, FOR all columns or combination of some of the 'for' options of analyze statistics (table). For V, it can be CASCADE when object_type is T.
sample_clause	Sample clause to use when command_type is 'E'

### Usage Notes

For each partition of the object, run in parallel using job queues.

## ANALYZE\_SCHEMA Procedure

This procedure analyzes all the tables, clusters and indexes in a schema.

### Syntax

```
DBMS_UTILITY.ANALYZE_SCHEMA (
  schema          IN VARCHAR2,
  method          IN VARCHAR2,
  estimate_rows   IN NUMBER DEFAULT NULL,
  estimate_percent IN NUMBER DEFAULT NULL,
  method_opt      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 156–8 ANALYZE\_SCHEMA Procedure Parameters**

Parameter	Description
schema	Name of the schema
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE then either estimate_rows or estimate_percent must be nonzero.
estimate_rows	Number of rows to estimate
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified ignore this parameter.
method_opt	Method options of the following format: [ FOR TABLE ] [ FOR ALL [INDEXED] COLUMNS] [SIZE n] [ FOR ALL INDEXES ]

### Exceptions

Ora-20000: Insufficient privileges for some object in this schema

## CANONICALIZE Procedure

This procedure canonicalizes the given string. The procedure handles a single reserved or key word (such as 'table'), and strips off white spaces for a single identifier so that ' table ' becomes TABLE.

### Syntax

```
DBMS_UTILITY.CANONICALIZE(
  name          IN   VARCHAR2,
  canon_name    OUT  VARCHAR2,
  canon_len     IN   BINARY_INTEGER);
```

### Parameters

**Table 156–9 CANONICALIZE Procedure Parameters**

Parameter	Description
name	String to be canonicalized
canon_name	Canonicalized string
canon_len	Length of the string (in bytes) to canonicalize

### Return Values

Returns the first `canon_len` bytes in `canon_name`.

### Usage Notes

- If `name` is NULL, `canon_name` becomes NULL.
- If `name` is not a dotted name, and if `name` begins and ends with a double quote, remove both quotes. Alternatively, convert to upper case with `NLS_UPPER`. Note that this case does not include a name with special characters, such as a space, but is not doubly quoted.
- If `name` is a dotted name (such as `a."b".c`), for each component in the dotted name in the case in which the component begins and ends with a double quote, no transformation will be performed on this component. Alternatively, convert to upper case with `NLS_UPPER` and apply begin and end double quotes to the capitalized form of this component. In such a case, each canonicalized component will be concatenated together in the input position, separated by ".".
- Any other character after `a[b]*` will be ignored.
- The procedure does not handle cases like 'A B.'

### Examples

- `a` becomes `A`
- `"a"` becomes `a`
- `"a".b` becomes `"a"."B"`
- `"a".b,c.f` becomes `"a"."B"` with `",c.f"` ignored.

## COMMA\_TO\_TABLE Procedures

These procedures convert a comma-delimited list of names into a PL/SQL table of names. The second version supports fully-qualified attribute names.

### Syntax

```
DBMS_UTILITY.COMMA_TO_TABLE (
  list IN VARCHAR2,
  tablen OUT BINARY_INTEGER,
  tab OUT uncl_array);
```

```
DBMS_UTILITY.COMMA_TO_TABLE (
  list IN VARCHAR2,
  tablen OUT BINARY_INTEGER,
  tab OUT lname_array);
```

### Parameters

**Table 156–10** *COMMA\_TO\_TABLE Procedure Parameters*

Parameter	Description
list	Comma separated list of list of 'names', where a name should have the following format for the first overloading: a [. b [. c ]][ @ d ] and the following format for the second overloading: a [. b]* where a, b, c, d are simple identifiers (quoted or unquoted).
tablen	Number of tables in the PL/SQL table
tab	PL/SQL table which contains list of names

### Return Values

A PL/SQL table is returned, with values 1..n and n+1 is null.

### Usage Notes

- The list must be a non-empty comma-delimited list: Anything other than a comma-delimited list is rejected. Commas inside double quotes do not count.
- Entries in the comma-delimited list cannot include multibyte characters.
- The values in tab are copied from the original list, with no transformations.
- The procedure fails if the string between separators is longer than 30 bytes.

## COMPILE\_SCHEMA Procedure

This procedure compiles all procedures, functions, packages, views and triggers in the specified schema.

### Syntax

```
DBMS_UTILITY.COMPILE_SCHEMA (
  schema          IN VARCHAR2,
  compile_all     IN BOOLEAN DEFAULT TRUE,
  reuse_settings  IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 156–11** *COMPILE\_SCHEMA Procedure Parameters*

Parameter	Description
schema	Name of the schema
compile_all	If TRUE, will compile everything within the schema regardless of whether it is VALID If FALSE, will compile only INVALID objects
reuse_settings	Indicates whether the session settings in the objects should be reused, or whether the current session settings should be adopted instead

### Exceptions

**Table 156–12** *COMPILE\_SCHEMA Procedure Exceptions*

Exception	Description
ORA-20000	Insufficient privileges for some object in this schema
ORA-20001	Cannot recompile SYS objects
ORA-20002	Maximum iterations exceeded. Some objects may not have been recompiled.

### Usage Notes

- Note that this subprogram is a wrapper for the [RECOMP\\_SERIAL Procedure](#) included with the [UTL\\_RECOMP](#) package.
- After calling this procedure, you should select from view `ALL_OBJECTS` for items with status of `INVALID` to see if all objects were successfully compiled.
- To see the errors associated with `INVALID` objects, you may use the Enterprise Manager command:
 

```
SHOW ERRORS <type> <schema>.<name>
```

## CREATE\_ALTER\_TYPE\_ERROR\_TABLE Procedure

This procedure creates an error table to be used in the `EXCEPTION` clause of the `ALTER TYPE` statement.

### Syntax

```
DBMS_UTILITY.CREATE_ALTER_TYPE_ERROR_TABLE(  
    schema_name    IN    VARCHAR2,  
    table_name     IN    VARCHAR2);
```

### Parameters

**Table 156–13** *CREATE\_ALTER\_TYPE\_ERROR\_TABLE Procedure Parameters*

Parameter	Description
<code>schema_name</code>	Name of the schema
<code>table_name</code>	Name of the table created

### Exceptions

An error is returned if the table already exists.



## CURRENT\_INSTANCE Function

This function returns the current connected instance number. It returns NULL when connected instance is down.

### Syntax

```
DBMS_UTILITY.CURRENT_INSTANCE  
RETURN NUMBER;
```

## DATA\_BLOCK\_ADDRESS\_BLOCK Function

This function gets the block number part of a data block address.

### Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK (  
    dba NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 156–14 DATA\_BLOCK\_ADDRESS\_BLOCK Function Parameters**

Parameter	Description
dba	Data block address

### Pragmas

```
pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);
```

### Return Values

Block offset of the block.

### Usage Notes

This function should not be used with datablocks which belong to bigfile tablespaces.

## DATA\_BLOCK\_ADDRESS\_FILE Function

This function gets the file number part of a data block address.

### Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE (  
    dba NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 156–15 DATA\_BLOCK\_ADDRESS\_FILE Function Parameters**

Parameter	Description
dba	Data block address

### Pragmas

```
pragma restrict_references (data_block_address_file, WNDS, RNDS, WNPS, RNPS);
```

### Return Values

File that contains the block.

### Usage Notes

This function should not be used with datablocks which belong to bigfile tablespaces.

## DB\_VERSION Procedure

This procedure returns version information for the database.

### Syntax

```
DBMS_UTILITY.DB_VERSION (  
    version      OUT VARCHAR2,  
    compatibility OUT VARCHAR2);
```

### Parameters

**Table 156–16** *DB\_VERSION Procedure Parameters*

Parameter	Description
version	A string which represents the internal software version of the database (for example, 7.1.0.0.0). The length of this string is variable and is determined by the database version.
compatibility	The compatibility setting of the database determined by the "compatible" <code>init.ora</code> parameter. If the parameter is not specified in the <code>init.ora</code> file, then NULL is returned.

## EXEC\_DDL\_STATEMENT Procedure

This procedure executes the DDL statement in `parse_string`.

### Syntax

```
DBMS_UTILITY.EXEC_DDL_STATEMENT (  
    parse_string IN VARCHAR2);
```

### Parameters

**Table 156–17 EXEC\_DDL\_STATEMENT Procedure Parameters**

Parameter	Description
<code>parse_string</code>	DDL statement to be executed

## FORMAT\_CALL\_STACK Function

This function formats the current call stack. This can be used on any stored procedure or trigger to access the call stack. This can be useful for debugging.

### Syntax

```
DBMS_UTILITY.FORMAT_CALL_STACK  
RETURN VARCHAR2;
```

### Pragmas

```
pragma restrict_references(format_call_stack,WNDS);
```

### Return Values

This returns the call stack, up to 2000 bytes.

## FORMAT\_ERROR\_BACKTRACE Function

This procedure displays the call stack at the point where an exception was raised, even if the procedure is called from an exception handler in an outer scope. The output is similar to the output of the SQLERRM function, but not subject to the same size limitation.

### Syntax

```
DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
RETURN VARCHAR2;
```

### Return Values

The backtrace string. A NULL string is returned if no error is currently being handled.

### Examples

```
CREATE OR REPLACE PROCEDURE Log_Errors ( i_buff in varchar2 ) IS
  g_start_pos integer := 1;
  g_end_pos integer;

  FUNCTION Output_One_Line RETURN BOOLEAN IS
  BEGIN
    g_end_pos := Instr ( i_buff, Chr(10), g_start_pos );

    CASE g_end_pos > 0
      WHEN true THEN
        DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
          g_end_pos-g_start_pos ) );
        g_start_pos := g_end_pos+1;
        RETURN TRUE;

      WHEN FALSE THEN
        DBMS_OUTPUT.PUT_LINE ( Substr ( i_buff, g_start_pos,
          (Length(i_buff)-g_start_pos)+1 ) );
        RETURN FALSE;
    END CASE;
  END Output_One_Line;

BEGIN
  WHILE Output_One_Line() LOOP NULL;
  END LOOP;
END Log_Errors;
/

Set Doc Off
Set Feedback off
Set Echo Off

CREATE OR REPLACE PROCEDURE P0 IS
  e_01476 EXCEPTION; pragma exception_init ( e_01476, -1476 );
BEGIN
  RAISE e_01476;
END P0;
/
Show Errors

CREATE OR REPLACE PROCEDURE P1 IS
```

```

BEGIN
  P0();
END P1;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P2 IS
BEGIN
  P1();
END P2;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P3 IS
BEGIN
  P2();
END P3;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE P4 IS
  BEGIN P3(); END P4;
/
CREATE OR REPLACE PROCEDURE P5 IS
  BEGIN P4(); END P5;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_Naive IS
BEGIN
  P5();
END Top_Naive;
/
SHOW ERRORS

CREATE OR REPLACE PROCEDURE Top_With_Logging IS
  -- NOTE: SqlErrm in principle gives the same info as Format_Error_Stack.
  -- But SqlErrm is subject to some length limits,
  -- while Format_Error_Stack is not.
BEGIN
  P5();
EXCEPTION
  WHEN OTHERS THEN
    Log_Errors ( 'Error_Stack...' || Chr(10) ||
      DBMS_UTILITY.FORMAT_ERROR_STACK() );
    Log_Errors ( 'Error_Backtrace...' || Chr(10) ||
      DBMS_UTILITY.FORMAT_ERROR_BACKTRACE() );
    DBMS_OUTPUT.PUT_LINE ( '-----' );
END Top_With_Logging;
/
SHOW ERRORS

-----

Set ServerOutput On
call Top_Naive()
/*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at "U.P0", line 4

```



```
ORA-06512: at "U.P1", line 3
ORA-06512: at "U.P2", line 3
ORA-06512: at "U.P3", line 3
ORA-06512: at "U.P4", line 2
ORA-06512: at "U.P5", line 2
ORA-06512: at "U.TOP_NAIVE", line 3
*/
;

Set ServerOutput On
call Top_With_Logging()
/*
  Error_Stack...
ORA-01476: divisor is equal to zero
  Error_Backtrace...
ORA-06512: at "U.P0", line 4
ORA-06512: at "U.P1", line 3
ORA-06512: at "U.P2", line 3
ORA-06512: at "U.P3", line 3
ORA-06512: at "U.P4", line 2
ORA-06512: at "U.P5", line 2
ORA-06512: at "U.TOP_WITH_LOGGING", line 6
-----
*/
;

/*
ORA-06512:
Cause:
  Backtrace message as the stack is
  unwound by unhandled exceptions.
Action:
  Fix the problem causing the exception
  or write an exception handler for this condition.
  Or you may need to contact your application administrator
  or database administrator.
*/
```

## FORMAT\_ERROR\_STACK Function

This function formats the current error stack. This can be used in exception handlers to look at the full error stack.

### Syntax

```
DBMS_UTILITY.FORMAT_ERROR_STACK  
RETURN VARCHAR2;
```

### Return Values

This returns the error stack, up to 2000 bytes.

## GET\_CPU\_TIME Function

This function returns a measure of current CPU processing time in hundredths of a second. The difference between the times returned from two calls measures the CPU processing time (not the total elapsed time) between those two points.

### Syntax

```
DBMS_UTILITY.GET_CPU_TIME  
RETURN NUMBER;
```

### Return Values

Time is the number of 100th's of a second from some arbitrary epoch.

### Usage Notes

The amount of work performed is calculated by measuring the difference between a start point and end point for a particular operation.

## GET\_DEPENDENCY Procedure

This procedure shows the dependencies on the object passed in.

### Syntax

```
DBMS_UTILITY.GET_DEPENDENCY  
type          IN      VARCHAR2,  
schema        IN      VARCHAR2,  
name          IN      VARCHAR2);
```

### Parameters

**Table 156–18** GET\_DEPENDENCY Procedure Parameters

Parameter	Description
type	Type of the object, for example if the object is a table give the type as 'TABLE'
schema	Schema name of the object
name	Name of the object

### Usage Notes

This procedure uses the [DBMS\\_OUTPUT](#) package to display results, and so you must declare `SET SERVEROUTPUT ON` if you wish to view dependencies. Alternatively, any application that checks the `DBMS_OUTPUT` output buffers can invoke this subprogram and then retrieve the output by means of `DBMS_OUTPUT` subprograms such as `GET_LINES`.

## GET\_ENDIANNES Function

This Function Gets The Endianness Of The Database Platform.

### Syntax

```
DBMS_UTILITY.GET_ENDIANNES  
Return Number;
```

### Return Values

The endianness of the database platform.

## GET\_HASH\_VALUE Function

This function computes a hash value for the given string.

### Syntax

```
DBMS_UTILITY.GET_HASH_VALUE (  
    name      VARCHAR2,  
    base      NUMBER,  
    hash_size NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 156–19** GET\_HASH\_VALUE Function Parameters

Parameter	Description
name	String to be hashed.
base	Base value for the returned hash value at which to start
hash_size	Desired size of the hash table

### Pragmas

```
pragma restrict_references(get_hash_value, WNDS, RNDS, WNPS, RNPS);
```

### Return Values

A hash value based on the input string. For example, to get a hash value on a string where the hash value should be between 1000 and 3047, use 1000 as the base value and 2048 as the hash\_size value. Using a power of 2 for the hash\_size parameter works best.

## GET\_PARAMETER\_VALUE Function

This function gets the value of specified `init.ora` parameter.

### Syntax

```
DBMS_UTILITY.GET_PARAMETER_VALUE (
    parnam      IN      VARCHAR2,
    intval      IN OUT  BINARY_INTEGER,
    strval      IN OUT  VARCHAR2,
    listno      IN      BINARY_INTEGER DEFAULT 1)
RETURN BINARY_INTEGER;
```

### Parameters

**Table 156–20 GET\_PARAMETER\_VALUE Function Parameters**

Parameter	Description
parnam	Parameter name
intval	Value of an integer parameter or the value length of a string parameter
strval	Value of a string parameter
listno	List item number. If retrieving parameter values for a parameter that can be specified multiple times to accumulate values, use this parameter to get each individual parameter.

### Return Values

Parameter type:

- 0 if parameter is an `INTEGER/BOOLEAN` parameter
- 1 if parameter is a string/file parameter

### Usage Notes

When using `DBMS_UTILITY.GET_PARAMETER_VALUE`, only the first parameter setting of `/dir1` is returned when `init.ora` is set as follows:

```
utl_file_dir = /dir1
utl_file_dir = /dir2
```

However, the full comma-delimited string is returned if you are using:

```
utl_file_dir = /dir1, /dir2
```

### Examples

```
DECLARE
    parnam VARCHAR2(256);
    intval BINARY_INTEGER;
    strval VARCHAR2(256);
    partyp BINARY_INTEGER;
BEGIN
    partyp := dbms_utility.get_parameter_value('max_dump_file_size',
                                              intval, strval);

    dbms_output.put('parameter value is: ');
    IF partyp = 1 THEN
        dbms_output.put_line(strval);
```

```
ELSE
    dbms_output.put_line(intval);
END IF;
IF partyp = 1 THEN
    dbms_output.put('parameter value length is: ');
    dbms_output.put_line(intval);
END IF;
dbms_output.put('parameter type is: ');
IF partyp = 1 THEN
    dbms_output.put_line('string');
ELSE
    dbms_output.put_line('integer');
END IF;
END;
```



## GET\_SQL\_HASH Function

This function computes a hash value for the given string using MD5 algorithm.

### Syntax

```
Dbms_utility.get_sql_hash (
    name          IN   VARCHAR2,
    hash          OUT  RAW,
    pre10ishash  OUT  NUMBER)
RETURN NUMBER;
```

### Pragmas

```
Pragma Restrict_references(Get_sql_hash, Wnds, Rnds, Wnps, Rnps);
```

### Parameters

**Table 156–21 GET\_SQL\_HASH Procedure Parameters**

Parameter	Description
name	String to be hashed
hash	Optional field to store all 16 bytes of returned hash value
pre10ishash	Optional field to store the pre 10i database version hash value

### Return Values

A hash value (last 4 bytes) based on the input string. the MD5 hash algorithm computes a 16 byte hash value, but we only return the last 4 bytes so that we can return an actual number. one could use an optional raw parameter to get all 16 bytes and to store the pre 10i hash value of 4 bytes in the pre10i hash optional parameter.

## GET\_TIME Function

This function returns the current time in 100th's of a second. This subprogram is primarily used for determining elapsed time. The subprogram is called twice – at the beginning and end of some process – and then the first (earlier) number is subtracted from the second (later) number to determine the time elapsed.

### Syntax

```
DBMS_UTILITY.GET_TIME  
RETURN NUMBER;
```

### Return Values

Time is the number of 100th's of a second from the point in time at which the subprogram is invoked.

### Usage Notes

Numbers are returned in the range -2147483648 to 2147483647 depending on platform and machine, and your application must take the sign of the number into account in determining the interval. For instance, in the case of two negative numbers, application logic must allow that the first (earlier) number will be larger than the second (later) number which is closer to zero. By the same token, your application should also allow that the first (earlier) number be negative and the second (later) number be positive.

## GET\_TZ\_TRANSITIONS Procedure

This procedure returns timezeone transitions by `regionid` from the `timezone.dat` file.

### Syntax

```
DBMS_UTILITY.GET_TZ_TRANSITIONS
  regionid      IN      NUMBER,
  transitions   OUT    MAXRAW);
```

### Parameters

**Table 156–22 GET\_TZ\_TRANSITIONS Procedure Parameters**

Parameter	Description
<code>regionid</code>	Number corresponding to the region
<code>transitions</code>	Raw bytes from the <code>timezone.dat</code> file

### Exceptions

**Table 156–23 GET\_TZ\_TRANSITIONS Procedure Exceptions**

Exception	Description
ORA-6502: PL/SQL: NUMERIC OR VALUE ERROR	For an invalid <code>regionid</code>

## INVALIDATE Procedure

This procedure invalidates a database object and (optionally) modifies its PL/SQL compiler parameter settings. It also invalidates any objects that (directly or indirectly) depend on the object being invalidated.

### Syntax

```
DBMS_UTILITY.INVALIDATE (
    p_object_id          NUMBER,
    p_plsql_object_settings VARCHAR2 DEFAULT NULL,
    p_option_flags       PLS_INTEGER DEFAULT 0);
```

### Parameters

**Table 156–24** *INVALIDATE Procedure Parameters*

Parameter	Description
p_object_id	ID number of object to be invalidated. This is the same as the value of the OBJECT_ID column from ALL_OBJECTS. If the object_id argument is NULL or invalid then the exception inv_not_exist_or_no_priv is raised. The caller of this procedure must have create privileges on the object being invalidated else the inv_not_exist_or_no_priv exception is raised.
p_plsql_object_settings	Optional parameter that ignored if the object specified by p_object_id is not a PL/SQL object. If no value is specified for this parameter then the PL/SQL compiler settings are left unchanged, that is, equivalent to REUSE SETTINGS. If a value is provided, it must specify the values of the PL/SQL compiler settings separated by one or more spaces. Each setting can be specified only once else inv_malformed_settings exception will be raised. The setting values are changed only for the object specified by p_object_id and do not affect dependent objects that may be invalidated. The setting names and values are case insensitive. If a setting is omitted and REUSE SETTINGS is specified, then if a value was specified for the compiler setting in an earlier compilation of this library unit, Oracle Database uses that earlier value. If a setting is omitted and REUSE SETTINGS was not specified or no value has been specified for the parameter in an earlier compilation, then the database will obtain the value for that setting from the session environment.
p_option_flags	Optional parameter defaults to zero (no flags). Option flags supported by invalidate. <ul style="list-style-type: none"> <li>inv_error_on_restrictions (see <a href="#">Constants</a> on page 156-4): The subprogram imposes various restrictions on the objects that can be invalidated. For example, the object specified by p_object_id cannot be a table. By default, invalidate quietly returns on these conditions (and does not raise an exception). If the caller sets this flag, the exception inv_restricted_object is raised.</li> </ul>

## Exceptions

**Table 156–25 INVALIDATE Procedure Exceptions**

Exception	Description
INV_NOT_EXIST_OR_NO_PRIV	Raised when the <code>object_id</code> argument is NULL or invalid, or when the caller does not have CREATE privileges on the object being invalidated
INV_MALFORMED_SETTINGS	Raised if a compiler setting is specified more than once in the <code>p_plsql_object_settings</code> parameter
INV_RESTRICTED_OBJECT	Raised when different combinations of conditions pertaining to the <code>p_object_id</code> parameter are contravened

## Usage Notes

The object type (`object_type` column from ALL\_OBJECTS) of the object specified by `p_object_id` must be a PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE, TYPE BODY, LIBRARY, VIEW, OPERATOR, SYNONYM, or JAVA CLASS. If the object is not one of these types and the flag `inv_error_on_restrictions` is specified in `p_option_flags` then the exception `inv_restricted_object` is raised, else no action is taken.

If the object specified by `p_object_id` is the package specification of STANDARD, DBMS\_STANDARD, or specification or body of DBMS\_UTILITY and the flag `inv_error_on_restrictions` is specified in `p_option_flags` then the exception `inv_restricted_object` is raised, else no action is taken.

If the object specified by `p_object_id` is an object type specification and there exist tables which depend on the type and the flag `inv_error_on_restrictions` is specified in `p_option_flags` then the exception `inv_restricted_object` is raised, else no action is taken.

## Examples

### Example 1

```
DBMS_UTILITY.INVALIDATE (1232, 'PLSQL_OPTIMIZE_LEVEL = 2 REUSE SETTINGS');
```

Assume that the `object_id` 1232 refers to the procedure `remove_emp` in the HR schema. Then the above call will mark the `remove_emp` procedure invalid and change its `PLSQL_OPTIMIZE_LEVEL` compiler setting to 2. The values of other compiler settings will remain unchanged since `REUSE SETTINGS` is specified.

Objects that depend on `hr.remove_emp` will also get marked invalid. Their compiler parameters will not be changed.

### Example 2

```
DBMS_UTILITY.INVALIDATE (40775, 'plsql_code_type = native');
```

Assume that the `object_id` 40775 refers to the type body `leaf_category_typ` in the OE schema. Then the above call will mark the type body invalid and change its `PLSQL_CODE_TYPE` compiler setting to NATIVE. The values of other compiler settings will be picked up from the current session environment since `REUSE SETTINGS` has not been specified.

Since no objects can depend on bodies, there are no cascaded invalidations.

**Example 3**

```
DBMS_UTILITY.INVALIDATE (40796);
```

Assume that the `object_id` 40796 refers to the view `oc_orders` in the OE schema. Then the above call will mark the `oc_orders` view invalid.

Objects that depend on `oe.oc_orders` will also get marked invalid.

## IS\_BIT\_SET Function

This function checks the bit setting for the given bit in the given RAW value.

### Syntax

```
DBMS_UTILITY.IS_BIT_SET (  
    r      IN    RAW,  
    n      IN    NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 156–26 IS\_BET\_SET Function Parameters**

Parameter	Description
r	RAW source
n	Bit in r to check

### Return Values

This function returns 1 if bit n in raw r is set, zero otherwise. Bits are numbered high to low with the lowest bit being bit number 1.

## IS\_CLUSTER\_DATABASE Function

This function finds out if this database is running in cluster database mode.

### Syntax

```
DBMS_UTILITY.IS_CLUSTER_DATABASE  
RETURN BOOLEAN;
```

### Return Values

This function returns `TRUE` if this instance was started in cluster database mode;  
`FALSE` otherwise.



## MAKE\_DATA\_BLOCK\_ADDRESS Function

This function creates a data block address given a file number and a block number. A data block address is the internal structure used to identify a block in the database. This function is useful when accessing certain fixed tables that contain data block addresses.

### Syntax

```
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (  
    file NUMBER,  
    block NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 156–27 MAKE\_DATA\_BLOCK\_ADDRESS Function Parameters**

Parameter	Description
file	File that contains the block
block	Offset of the block within the file in terms of block increments

### Pragmas

```
pragma restrict_references (make_data_block_address, WNDS, RNDS, WNPS, RNPS);
```

### Return Values

Data block address.

## NAME\_RESOLVE Procedure

This procedure resolves the given name, including synonym translation and authorization checking as necessary.

### Syntax

```
DBMS_UTILITY.NAME_RESOLVE (
  name          IN  VARCHAR2,
  context       IN  NUMBER,
  schema        OUT VARCHAR2,
  part1         OUT VARCHAR2,
  part2         OUT VARCHAR2,
  dblink        OUT VARCHAR2,
  part1_type    OUT NUMBER,
  object_number OUT NUMBER);
```

### Parameters

**Table 156–28 NAME\_RESOLVE Procedure Parameters**

Parameter	Description
name	<p>Name of the object.</p> <p>This can be of the form <code>[[a.]b.]c[@d]</code>, where a, b, c are SQL identifier and d is a dblink. No syntax checking is performed on the dblink. If a dblink is specified, or if the name resolves to something with a dblink, then object is not resolved, but the <code>schema</code>, <code>part1</code>, <code>part2</code> and <code>dblink</code> OUT parameters are filled in.</p> <p>a, b and c may be delimited identifiers, and may contain Globalization Support (NLS) characters (single and multibyte).</p>
context	<p>Must be an integer between 0 and 9.</p> <ul style="list-style-type: none"> <li>■ 0 - table</li> <li>■ 1 - PL/SQL (for 2 part names)</li> <li>■ 2 - sequences</li> <li>■ 3 - trigger</li> <li>■ 4 - Java Source</li> <li>■ 5 - Java resource</li> <li>■ 6 - Java class</li> <li>■ 7 - type</li> <li>■ 8 - Java shared data</li> <li>■ 9 - index</li> </ul>
schema	<p>Schema of the object: c. If no schema is specified in name, then the schema is determined by resolving the name.</p>
part1	<p>First part of the name. The type of this name is specified <code>part1_type</code> (synonym or package).</p>
part2	<p>If this is non-NULL, then this is a subprogram name. If <code>part1</code> is non-NULL, then the subprogram is within the package indicated by <code>part1</code>. If <code>part1</code> is NULL, then the subprogram is a top-level subprogram.</p>

**Table 156–28 (Cont.) NAME\_RESOLVE Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>dblink</code>	If this is non-NULL, then a database link was either specified as part of <code>name</code> or <code>name</code> was a synonym which resolved to something with a database link. In this case, if further name translation is desired, then you must call the <code>DBMS_UTILITY.NAME_RESOLVE</code> procedure on this remote node.
<code>part1_type</code>	Type of <code>part1</code> is: <ul style="list-style-type: none"><li>■ 5 - synonym</li><li>■ 7 - procedure (top level)</li><li>■ 8 - function (top level)</li><li>■ 9 - package</li></ul>
<code>object_number</code>	Object identifier

## Exceptions

All errors are handled by raising exceptions. A wide variety of exceptions are possible, based on the various syntax error that are possible when specifying object names.

## NAME\_TOKENIZE Procedure

This procedure calls the parser to parse the given name as "a [. b [. c ]][@ dblink ]". It strips double quotes, or converts to uppercase if there are no quotes. It ignores comments of all sorts, and does no semantic analysis. Missing values are left as NULL.

### Syntax

```
DBMS_UTILITY.NAME_TOKENIZE (  
    name      IN  VARCHAR2,  
    a         OUT VARCHAR2,  
    b         OUT VARCHAR2,  
    c         OUT VARCHAR2,  
    dblink   OUT VARCHAR2,  
    nextpos  OUT BINARY_INTEGER);
```

### Parameters

**Table 156–29** *NAME\_RESOLVE Procedure Parameters*

Parameter	Description
name	Input name, consisting of SQL identifiers (for example, scott.foo@dblink)
a	Output for the first token of the name
b	Output for the second token of the name (if applicable)
c	Output for the third token of the name (if applicable)
dblink	Output for the dblink of the name
nextpos	Next position after parsing the input name

## OLD\_CURRENT\_SCHEMA Function

This function returns the session value from `sys_context ('userenv', 'current_schema')`.

### Syntax

```
DBMS_UTILITY.OLD_CURRENT_SCHEMA  
RETURN VARCHAR2;
```

## OLD\_CURRENT\_USER Function

This function returns the session value from `sys_context ('userenv', 'current_user')`.

### Syntax

```
DBMS_UTILITY.OLD_CURRENT_USER  
RETURN VARCHAR2;
```

## PORT\_STRING Function

This function returns a string that identifies the operating system and the TWO TASK PROTOCOL version of the database. For example, "VAX/VMX-7.1.0.0"

The maximum length is port-specific.

### Syntax

```
DBMS_UTILITY.PORT_STRING  
RETURN VARCHAR2;
```

### Pragmas

```
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```

## SQLID\_TO\_SQLHASH Function

This function converts a SQL ID into a hash value.

### Syntax

```
DBMS_UTILITY.SQLID_TO_SQLHASH (  
    sql_id    IN    VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 156–30** *SQLID\_TO\_SQLHASH Function Parameters*

Parameter	Description
sql_id	SQL ID of a SQL statement. Must be VARCHAR2 (13).



## TABLE\_TO\_COMMA Procedures

These procedures converts a PL/SQL table of names into a comma-delimited list of names. This takes a PL/SQL table, 1 . . n, terminated with n+1 null. The second version supports fully-qualified attribute names.

### Syntax

```
DBMS_UTILITY.TABLE_TO_COMMA (
  tab    IN UNCL_ARRAY,
  tablen OUT BINARY_INTEGER,
  list   OUT VARCHAR2);
```

```
DBMS_UTILITY.TABLE_TO_COMMA (
  tab    IN lname_array,
  tablen OUT BINARY_INTEGER,
  list   OUT VARCHAR2);
```

### Parameters

**Table 156–31** *TABLE\_TO\_COMMA Procedure Parameters*

Parameter	Description
tab	PL/SQL table which contains list of table names
tablen	Number of tables in the PL/SQL table
list	Comma separated list of tables

### Return Values

A comma-delimited list and the number of elements found in the table.

## VALIDATE Procedure

This procedure makes invalid database objects valid.

### Syntax

```
DBMS_UTILITY.VALIDATE (
    object_id      NUMBER);
```

```
DBMS_UTILITY.VALIDATE (
    owner          VARCHAR2,
    objname        VARCHAR2,
    namespace      NUMBER,
    edition_name   := SYS_CONTEXT ('USERENV', 'CURRENT_EDITION'));
```

### Parameters

**Table 156–32 VALIDATE Procedure Parameters**

Parameter	Description
owner	Name of the user who owns the object. Same as the OWNER field in ALL_OBJECTS.
objname	Name of the object to be validated. Same as the OBJECT_NAME field in ALL_OBJECTS.
namespace	Namespace of the object. Same as the namespace field in obj\$. Equivalent numeric values are as follows: <ul style="list-style-type: none"> <li>■ 1 = TABLE/PROCEDURE/TYPE</li> <li>■ 2 = BODY</li> <li>■ 3 = TRIGGER</li> <li>■ 4 = INDEX</li> <li>■ 5 = CLUSTER</li> <li>■ 8 = LOB</li> <li>■ 9 = DIRECTORY</li> <li>■ 10 = QUEUE</li> <li>■ 11 = REPLICATION OBJECT GROUP</li> <li>■ 12 = REPLICATION PROPAGATOR</li> <li>■ 13 = JAVA SOURCE</li> <li>■ 14 = JAVA RESOURCE</li> <li>■ 58 = (Data Mining) MODEL</li> </ul>
edition_name	[Note: Currently not operable. Reserved for future use]

### Usage Notes

- No errors are raised if the object does not exist or is already valid or is an object that cannot be validated.
- If the object being validated is not actual in the specified edition, the subprogram automatically switches into the edition in which the object is actual prior to validation. That is, a call to VALIDATE will not actualize the object in the specified edition.

- The [INVALIDATE Procedure](#) invalidates a database object and optionally changes its PL/SQL compiler parameter settings. The object to be invalidated is specified by its `object_id`. The subprogram automatically switches to the edition in which the object is actual prior to invalidation. That is, a call to `INVALIDATE` will not actualize the object in the current edition.

## WAIT\_ON\_PENDING\_DML Function

This procedure waits until all transactions (other than the caller's own) that have locks on the listed tables and began prior to the specified scn have either committed or been rolled back.

### Syntax

```
DBMS_UTILITY.WAIT_ON_PENDING_DML (
  tables      IN      VARCHAR2,
  timeout     IN      BINARY_INTEGER,
  scn         IN OUT  NUMBER)
RETURN BOOLEAN;
```

### Parameters

**Table 156–33** WAIT\_ON\_PENDING\_DML Function Parameters

Parameter	Description
tables	Comma-separated list of one or more table names. The list must be valid for <a href="#">COMMA_TO_TABLE Procedures</a> , and each item valid to the <a href="#">NAME_RESOLVE Procedure</a> . Neither column specifiers nor DBLINK (database link) specifiers are allowed in the names, and each name must resolve to an existing table in the local database.
timeout	Maximum number of seconds to wait, totalled across all tables/transactions. A NULL or negative value will cause a very long wait.
scn	SCN prior to which transactions must have begun to be considered relevant to this request. If the value is NULL or not recognized as a meaningful scn on input, the most current SCN across all instances will be used and will be set into the passed argument as an output. If a meaningful value is passed in, its value will be preserved in the output.

### Return Values

TRUE if all relevant transactions have committed or been rolled back, FALSE if the timeout occurred prior to all relevant transactions committing or being rolled back

---

---

## DBMS\_WARNING

The `DBMS_WARNING` package provides a way to manipulate the behavior of PL/SQL warning messages, in particular by reading and changing the setting of the `PLSQL_WARNINGS` initialization parameter to control what kinds of warnings are suppressed, displayed, or treated as errors. This package provides the interface to query, modify and delete current system or session settings.

This chapter contains the following topics:

- [Using DBMS\\_WARNING](#)
  - Security Model
- [Summary of DBMS\\_WARNING Subprograms](#)

## Using DBMS\_WARNING

- [Security Model](#)

## Security Model

Note that for all the following interfaces, if value of the scope parameter is `SYSTEM`, then the user must have `ALTER SYSTEM` privilege.

## Summary of DBMS\_WARNING Subprograms

**Table 157–1 DBMS\_WARNING Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ADD_WARNING_SETTING_CAT Procedure</a> on page 157-5	Modifies the current session or system warning settings of the <code>warning_category</code> previously supplied
<a href="#">ADD_WARNING_SETTING_NUM Procedure</a> on page 157-6	Modifies the current session or system warning settings of the or <code>warning_number</code> previously supplied
<a href="#">GET_CATEGORY Function</a> on page 157-7	Returns the category name, given the message number
<a href="#">GET_WARNING_SETTING_CAT Function</a> on page 157-8	Returns the specific warning category in the session
<a href="#">GET_WARNING_SETTING_NUM Function</a> on page 157-9	Returns the specific warning number in the session
<a href="#">GET_WARNING_SETTING_STRING Function</a> on page 157-10	Returns the entire warning string for the current session
<a href="#">SET_WARNING_SETTING_STRING Procedure</a> on page 157-11	Replaces previous settings with the new value



## ADD\_WARNING\_SETTING\_CAT Procedure

You can modify the current session's or system's warning settings with the value supplied. The value will be added to the existing parameter setting if the value for the `warning_category` or `warning_value` has not been set, or override the existing value. The effect of calling this function is same as adding the qualifier (ENABLE/DISABLE/ERROR) on the category specified to the end of the current session or system setting.

### Syntax

```
DBMS_WARNING.ADD_WARNING_SETTING_CAT (
  warning_category  IN   VARCHAR2,
  warning_value     IN   VARCHAR2,
  scope             IN   VARCHAR2);
```

### Parameters

**Table 157-2 ADD\_WARNING\_SETTING\_CAT Procedure Parameters**

Parameter	Description
<code>warning_category</code>	Name of the category. Allowed values are ALL, INFORMATIONAL, SEVERE and PERFORMANCE.
<code>warning_value</code>	Value for the category. Allowed values are ENABLE, DISABLE, and ERROR.
<code>scope</code>	Specifies if the changes are being performed in the session context or the system context. Allowed values are SESSION or SYSTEM.

## ADD\_WARNING\_SETTING\_NUM Procedure

You can modify the current `session` or `system` warning settings with the value supplied. If the value was already set, you will override the existing value. The effect of calling this function is same as adding the qualifier (`ENABLE` / `DISABLE` / `ERROR`) on the category specified to the end of the current session or system setting.

### Syntax

```
DBMS_WARNING.ADD_WARNING_SETTING_NUM (  
    warning_number    IN    NUMBER,  
    warning_value     IN    VARCHAR2,  
    scope             IN    VARCHAR2);
```

### Parameters

**Table 157-3 ADD\_WARNING\_SETTING\_NUM Procedure Parameters**

Parameter	Description
<code>warning_number</code>	The warning number. Allowed values are all valid warning numbers.
<code>warning_value</code>	Value for the category. Allowed values are <code>ENABLE</code> , <code>DISABLE</code> , and <code>ERROR</code> .
<code>scope</code>	Specifies if the changes are being performed in the session context or the system context. Allowed values are <code>SESSION</code> or <code>SYSTEM</code> .

## GET\_CATEGORY Function

This function returns the category name, given the message number.

### Syntax

```
DBMS_WARNING.GET_CATEGORY (  
    warning_number IN pls_integer)  
RETURN VARCHAR2;
```

### Parameters

**Table 157-4** GET\_CATEGORY Function Parameters

Parameter	Description
warning_number	The warning message number.

## GET\_WARNING\_SETTING\_CAT Function

This function returns the specific warning category setting for the current session.

### Syntax

```
DBMS_WARNING.GET_WARNING_SETTING_CAT (  
    warning_category IN VARCHAR2)  
RETURN warning_value;
```

### Parameters

**Table 157-5** *GET\_WARNING\_SETTING\_CAT Function Parameters*

Parameter	Description
warning_category	Name of the category. Allowed values are all valid category names (ALL, INFORMATIONAL, SEVERE and PERFORMANCE).

## GET\_WARNING\_SETTING\_NUM Function

This function returns the specific warning number setting for the current session.

### Syntax

```
DBMS_WARNING.GET_WARNING_SETTING_NUM (  
    warning_number    IN    NUMBER)  
RETURN warning_value;
```

### Parameters

**Table 157–6** *GET\_WARNING\_SETTING\_NUM Function Parameters*

Parameter	Description
warning_number	Warning number. Allowed values are all valid warning numbers.

## GET\_WARNING\_SETTING\_STRING Function

This function returns the entire warning string for the current session.

### Syntax

```
DBMS_WARNING.GET_WARNING_SETTING_STRING  
RETURN pls_integer;
```

### Usage Notes

Use this function when you do not have `SELECT` privilege on `v$parameter` or `v$parameter2` fixed tables, or if you want to parse the warning string yourself and then modify and set the new value using `SET_WARNING_SETTING_STRING`.

## SET\_WARNING\_SETTING\_STRING Procedure

This procedure replaces previous settings with the new value. The warning string may contain mix of category and warning numbers using the same syntax as used on the right hand side of '=' when issuing an ALTER SESSION or SYSTEM SET PLSQL\_WARNINGS command. This will have same effect as ALTER SESSION OR ALTER SYSTEM command.

### Syntax

```
DBMS_WARNING.SET_WARNING_SETTING_STRING (
    warning_value IN VARCHAR2,
    scope        IN VARCHAR2);
```

### Parameters

**Table 157-7 SET\_WARNING\_SETTING\_STRING Procedure Parameters**

Parameter	Description
warning_value	The new string that will constitute the new value.
scope	This will specify if the changes are being done in the session context, or system context. Allowed values are SESSION or SYSTEM.





The DBMS\_WM package provides an interface to Oracle Database Workspace Manager (often referred to as Workspace Manager).

- [Documentation of DBMS\\_WM](#)

---

## Documentation of DBMS\_WM

For a complete description of this package, see DBMS\_WM in *Oracle Database Workspace Manager Developer's Guide*.

---

---

## DBMS\_WORKLOAD\_CAPTURE

The `DBMS_WORKLOAD_CAPTURE` package configures the Workload Capture system and produce the workload capture data. Replay of this capture is implemented by way of the [DBMS\\_WORKLOAD\\_REPLAY](#) package.

**See Also:** *Oracle Database Real Application Testing User's Guide* for more information about Database Replay

This chapter contains the following topics:

- [Using DBMS\\_WORKLOAD\\_CAPTURE](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_WORKLOAD\\_CAPTURE Subprograms](#)

## Using DBMS\_WORKLOAD\_CAPTURE

- [Overview](#)
- [Security Model](#)

## Overview

Since the capture infrastructure is instance wide (and also within an Oracle Real Application Clusters (Oracle RAC)), only one workload capture is being produced at any point in time. Thus capture interfaces do not need a state object passed in as a parameter since there is one single state at any point in time. This means that all subprograms cannot be methods of an object but are package wide PL/SQL subprograms.

## Security Model

The following code describes the minimal set of privileges required to:

- Create directory objects
- Operate the interface provided by the DBMS\_WORKLOAD\_CAPTURE and DBMS\_WORKLOAD\_REPLAY packages
- Act as a replay client user (wrc someuser/somepassword or wrc USER=someuser PASSWORD=somepassword)

```
DROP USER rom1 CASCADE;
CREATE USER rom1 IDENTIFIED BY rom1;

GRANT EXECUTE ON DBMS_WORKLOAD_CAPTURE TO rom1;
GRANT EXECUTE ON DBMS_WORKLOAD_REPLAY TO rom1;

GRANT CREATE SESSION TO rom1;
GRANT CREATE ANY DIRECTORY TO rom1;
GRANT SELECT_CATALOG_ROLE TO rom1;
GRANT BECOME USER TO rom1;
```

Appropriate OS permissions are required to access and manipulate files and directories on both the capture and replay system. This means that the Oracle process(es) and the OS user performing the capture or replay must be able to access and manipulate at least one common directory accessible from the host where the instance is running. Additionally, the OS user performing the replay should be able to execute wrc on hosts that are used for the replay clients and be able to access the file system appropriately to copy the capture to the replay clients' hosts if required.

---

## Summary of DBMS\_WORKLOAD\_CAPTURE Subprograms

This table lists the package subprograms in alphabetical order.

**Table 159–1 DBMS\_WORKLOAD\_CAPTURE Package Subprograms**

Subprogram	Description
<a href="#">ADD_FILTER Procedures</a> on page 159-6	Adds a specified filter
<a href="#">DELETE_CAPTURE_INFO Procedure</a> on page 159-7	Deletes the rows in the DBA_WORKLOAD_CAPTURES and DBA_WORKLOAD_FILTERS views that corresponds to the given workload capture ID
<a href="#">DELETE_FILTER Procedure</a> on page 159-8	Deletes a specified filter
<a href="#">EXPORT_AWR Procedure</a> on page 159-9	Exports the AWR snapshots associated with a given capture ID
<a href="#">FINISH_CAPTURE Procedure</a> on page 159-10	Finalizes the workload capture by signaling all connected sessions to stop capture, and stops future requests to the database from being captured
<a href="#">GET_CAPTURE_INFO Function</a> on page 159-11	Retrieves all the information regarding a workload capture present in the stipulated directory, imports the information into the DBA_WORKLOAD_CAPTURES and DBA_WORKLOAD_FILTERS views, and returns the appropriate DBA_WORKLOAD_CAPTURES.ID
<a href="#">IMPORT_AWR Function</a> on page 159-12	Imports the AWR snapshots associated with a given capture ID
<a href="#">REPORT Function</a> on page 159-13	Returns a report on the workload capture under consideration using one or more different sources
<a href="#">START_CAPTURE Procedure</a>	Initiates workload capture on all instances

## ADD\_FILTER Procedures

This procedure adds a filter to capture a subset of the workload.

### Syntax

```
DBMS_WORKLOAD_CAPTURE.ADD_FILTER (
  fname          IN  VARCHAR2 NOT NULL,
  fattribute     IN  VARCHAR2 NOT NULL,
  fvalue        IN  VARCHAR2 NOT NULL);
```

```
DBMS_WORKLOAD_CAPTURE.ADD_FILTER (
  fname          IN  VARCHAR2 NOT NULL,
  fattribute     IN  VARCHAR2 NOT NULL,
  fvalue        IN  NUMBER NOT NULL);
```

### Parameters

**Table 159–2 ADD\_FILTER Procedure Parameters**

Parameter	Description
fname	Name for the filter to be added. Can be used to delete the filter later if it is not required. (Mandatory)
fattribute	Specifies the attribute on which the filter needs to be applied (Mandatory). The possible values are: <ul style="list-style-type: none"> <li>▪ INSTANCE_NUMBER - type NUMBER</li> <li>▪ USER - type STRING</li> <li>▪ MODULE - type STRING</li> <li>▪ ACTION - type STRING</li> <li>▪ PROGRAM - type STRING</li> <li>▪ SERVICE - type STRING</li> </ul>
fvalue	Specifies the value to which the given attribute should be equal to for the filter to be considered active. Wildcards like '%' are acceptable for all attributes that are of type STRING. This means that the filter for a NUMBER attribute is parsed as "attribute = value", with the filter for a STRING attribute parsed as "attribute like value" (Mandatory).

### Usage Notes

- The workload capture filters work in either the `DEFAULT INCLUSION` or the `DEFAULT EXCLUSION` mode as determined by the `default_action` input to the [START\\_CAPTURE Procedure](#).
- `ADD_FILTER` adds a new filter that affects the next workload capture, and whether the filters are considered as `INCLUSION` filters or `EXCLUSION` filters depends on the value of the `default_action` input to [START\\_CAPTURE Procedure](#).
- Filters once specified are valid only for the next workload capture. If the same set of filters need to be used for subsequent capture, they need to be specified each time before the [START\\_CAPTURE Procedure](#) is executed.
- All the filters are listed in the `DBA_WORKLOAD_FILTERS` view.



## DELETE\_CAPTURE\_INFO Procedure

This procedure deletes the rows in the DBA\_WORKLOAD\_CAPTURES and DBA\_WORKLOAD\_FILTERS views that corresponds to the given workload capture ID.

### Syntax

```
DBMS_WORKLOAD_CAPTURE.DELETE_CAPTURE_INFO  
  (capture_id IN NUMBER);
```

### Parameters

**Table 159–3** DELETE\_CAPTURE\_INFO Procedure Parameters

Parameter	Description
capture_id	ID of the workload capture that needs to be deleted. Corresponds to DBA_WORKLOAD_CAPTURES.ID. (Mandatory)

### Usage Notes

Passing the ID of a capture that is in progress will first automatically stop that capture.

## DELETE\_FILTER Procedure

This procedure deletes a specified filter.

### Syntax

```
DBMS_WORKLOAD_CAPTURE.DELETE_FILTER (  
    filter_name          IN  VARCHAR2(40) NOT NULL);
```

### Parameters

**Table 159–4** *DELETE\_FILTER Procedure Parameters*

Parameter	Description
filter_name	Filter to be deleted

### Usage Notes

The `DELETE_FILTER` Procedure only affects filters that have not been used by any previous capture. Consequently, filters can be deleted only if they have been added using the [ADD\\_FILTER Procedures](#) after any capture has been completed. Filters that have been added using `ADD_FILTER` before a `START_CAPTURE` and `FINISH_CAPTURE` cannot be deleted anymore using this subprogram.

## EXPORT\_AWR Procedure

This procedure exports the AWR snapshots associated with a given capture ID.

### Syntax

```
DBMS_WORKLOAD_CAPTURE.EXPORT_AWR (  
    capture_id    IN NUMBER);
```

### Parameters

**Table 159–5** EXPORT\_AWR Procedure Parameters

Parameter	Description
capture_id	ID of the capture whose AWR snapshots are to be exported. (Mandatory)

### Usage Notes

This procedure works only if the corresponding workload capture was performed in the current database (meaning that the corresponding row in DBA\_WORKLOAD\_CAPTURES was not created by calling the [GET\\_CAPTURE\\_INFO Function](#)) and the AWR snapshots that correspond to the original capture time period are still available.

## FINISH\_CAPTURE Procedure

This procedure signals all connected sessions to stop the workload capture and stops future requests to the database from being captured.

### Syntax

```
DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE
  timeout      IN  NUMBER  DEFAULT 30
  reason       IN  VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 159–6 FINISH\_CAPTURE Procedure Parameters**

Parameter	Description
timeout	Specifies in seconds for how long the procedure should wait before it times out. Pass 0 if you want to cancel the current workload capture and not wait for any sessions to flush its capture buffers. Default value: 30 seconds
reason	Specifies a reason for calling the procedure. The reason appears in the column <code>ERROR_MESSAGE</code> of the view <code>DBA_WORKLOAD_CAPTURES</code> .

### Usage Notes

- By default, `FINISH_CAPTURE` waits for 30 seconds to receive a successful acknowledgement from all sessions in the database cluster before timing out.
- All sessions that either were in the middle of executing a user request or received a new user request, while `FINISH_CAPTURE` was waiting for acknowledgements, flush their buffers and send back their acknowledgement to `FINISH_CAPTURE`.
- If a database session remains idle (waiting for the next user request) throughout the duration of `FINISH_CAPTURE`, the session might have unflushed capture buffers and does not send its acknowledgement to `FINISH_CAPTURE`.

To avoid this, do not have sessions that remain idle (waiting for the next user request) while invoking `FINISH_CAPTURE`. Either close the database session(s) before running `FINISH_CAPTURE` or send new database requests to those sessions during `FINISH_CAPTURE`.

## GET\_CAPTURE\_INFO Function

This procedure retrieves all information regarding a workload capture present in the stipulated directory, imports the information into the DBA\_WORKLOAD\_CAPTURES and DBA\_WORKLOAD\_FILTERS views, and returns the appropriate DBA\_WORKLOAD\_CAPTURES.ID

### Syntax

```
DBMS_WORKLOAD_CAPTURE.GET_CAPTURE_INFO
  dir      IN  VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 159–7 GET\_CAPTURE\_INFO Function Parameters**

Parameter	Description
dir	Name of the DIRECTORY object (case sensitive) where all the workload capture files are located (Mandatory)

### Usage Notes

If an appropriate row describing the capture in the stipulated directory already exists in DBA\_WORKLOAD\_CAPTURES, the [GET\\_CAPTURE\\_INFO Function](#) simply returns that row's DBA\_WORKLOAD\_CAPTURES.ID. If no existing row matches the capture present in the stipulated directory a new row is inserted to DBA\_WORKLOAD\_CAPTURES and that row's ID is returned.

## IMPORT\_AWR Function

This procedure imports the AWR snapshots associated with a given capture ID provided those AWR snapshots were exported earlier from the original capture system using the [EXPORT\\_AWR Procedure](#).

### Syntax

```
DBMS_WORKLOAD_CAPTURE.IMPORT_AWR (
  capture_id      IN   NUMBER,
  staging_schema  IN   VARCHAR2,
  force_cleanup   IN   BOOLEAN DEFAULT FALSE)
RETURN NUMBER;
```

### Parameters

**Table 159–8** *IMPORT\_AWR Function Parameters*

Parameter	Description
capture_id	ID of the capture whose AWR snapshots should be imported. (Mandatory)
staging_schema	Name of a valid schema in the current database which can be used as a staging area while importing the AWR snapshots from the capture directory to the SYS AWR schema. The SYS schema is not a valid input. (Mandatory, Case sensitive).
force_cleanup	<p>Values:</p> <ul style="list-style-type: none"> <li>TRUE - any AWR data present in the given staging_schema are removed before the actual import operation. All tables with names that match any of the tables in AWR are dropped before the actual import. This typically is equivalent to dropping all tables returned by the following SQL: <pre>SELECT table_name FROM dba_tables WHERE owner = staging_schema AND table_name like 'WR_\$%';</pre>           Use this option only if you are sure that there are no important data in any such tables in the staging_schema.         </li> <li>FALSE - (default) no tables dropped from the staging_schema prior to the import operation</li> </ul>

### Return Values

Returns the new randomly generated database ID that was used to import the AWR snapshots. The same value can be found in the AWR\_DBID column in the DBA\_WORKLOAD\_CAPTURES view.

### Usage Notes

IMPORT\_AWR fails if the staging\_schema provided as input contains any tables with the same name as any of the AWR tables, such as WRM\$\_SNAPSHOT or WRH\$\_PARAMETER. Please drop any such tables in the staging\_schema before invoking IMPORT\_AWR.

## REPORT Function

This function generates a report on the stipulated workload capture.

### Syntax

```
DBMS_WORKLOAD_CAPTURE.REPORT (
  capture_id      IN  NUMBER,
  format         IN  VARCHAR2)
RETURN CLOB;
```

### Parameters

**Table 159–9** *REPORT Function Parameters*

Parameter	Description
capture_id	ID of the workload capture whose capture report is required. (Mandatory)  This relates to the directory that contains the workload capture on which the Report needs to be generated. Should be a valid DIRECTORY object that points to a valid directory in the host system that contains a workload capture.
format	Specifies the report format. Valid values are DBMS_WORKLOAD_CAPTURE.TYPE_TEXT and DBMS_WORKLOAD_CAPTURE.TYPE_HTML.(Mandatory)

### Return Values

The report body in the desired format returned as a CLOB.

**Table 159–10** *Constants Used by Report Function*

Constant	Type	Value	Description
TYPE_HTML	VARCHAR2 (4)	'HTML'	Generates the HTML version of the report
TYPE_TEXT	VARCHAR2 (4)	'TEXT'	Used as as input to the format argument to generate the text version of the report

## START\_CAPTURE Procedure

This procedure initiates workload capture on all instances.

---

**Note:** The functionality associated with the `capture_sts` and `sts_cap_interval` parameters is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```
DBMS_WORKLOAD_CAPTURE.START_CAPTURE (
  name          IN  VARCHAR2,
  dir           IN  VARCHAR2,
  duration      IN  NUMBER   DEFAULT NULL,
  default_action IN  VARCHAR2 DEFAULT 'INCLUDE',
  auto_unrestrict IN BOOLEAN DEFAULT TRUE,
  capture_sts   IN  BOOLEAN  DEFAULT FALSE,
  sts_cap_interval IN NUMBER  DEFAULT 300);
```

### Parameters

**Table 159–11** *START\_CAPTURE Procedure Parameters*

Parameter	Description
<code>name</code>	Name of the workload capture. Allows the workload capture to be given a label, such as "Thanksgiving weekend" or "Christmas peak workload" for future reference. The workload capture's name is preserved along with the captured workload actions. (Mandatory)
<code>dir</code>	Name of the <code>DIRECTORY</code> object (case sensitive) where all the workload capture files are stored. Should contain enough space to hold all the workload capture files. (Mandatory)
<code>duration</code>	Optional input to specify the duration (in seconds) for which the workload needs to be captured. <code>DEFAULT</code> is <code>NULL</code> which means that workload capture continues until the user executes <code>DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE</code> .
<code>default_action</code>	Can be either <code>INCLUDE</code> or <code>EXCLUDE</code> . Determines whether, by default, every user request should be captured or not. Also determines whether the workload filters specified should be considered as <code>INCLUSION</code> filters or <code>EXCLUSION</code> filters. <ul style="list-style-type: none"> <li>■ If <code>INCLUDE</code>, by default all user requests to the database are captured, except for the part of the workload defined by the filters. In this case, all the filters specified using the <a href="#">ADD_FILTER Procedures</a> are treated as <code>EXCLUSION</code> filters, determining the workload that is not captured. (<code>DEFAULT</code>, and so all the filters specified are assumed to be <code>EXCLUSION</code> filters.)</li> <li>■ If <code>EXCLUDE</code>, by default no user request to the database is captured, except for the part of the workload defined by the filters. In this case, all the filters specified using the <a href="#">ADD_FILTER Procedures</a> are treated as <code>INCLUSION</code> filters, determining the workload that is captured.</li> </ul>



**Table 159–11 (Cont.) START\_CAPTURE Procedure Parameters**

Parameter	Description
auto_unrestrict	<p>Can be either TRUE or FALSE.</p> <ul style="list-style-type: none"> <li>■ If TRUE, all instances started up in RESTRICTED mode using STARTUP RESTRICT are automatically unrestricted upon a successful START_CAPTURE. (DEFAULT)</li> <li>■ If FALSE, no database instance is automatically unrestricted.</li> </ul>
capture_sts	<p>If this parameter is TRUE, a SQL tuning set capture is also started in parallel with workload capture. The resulting SQL tuning set can be exported using the <a href="#">EXPORT_AWR Procedure</a> along with the AWR data.</p> <p>Currently, parallel STS capture is not supported in an Oracle RAC environment, so this parameter has no effect if used in that context.</p> <p>Capture filters defined using the DBMS_WORKLOAD_REPLAY interface do not apply to the SQL tuning set capture.</p> <p>The calling user must have the appropriate privileges ('ADMINISTER SQL TUNING SET').</p> <p>If starting SQL set capture fails, workload capture is stopped. The reason is stored in DBA_WORKLOAD_CAPTURES.ERROR_MESSAGE.</p> <p>The default value is FALSE.</p>
sts_cap_interval	<p>Specifies the capture interval of the SQL set capture from the cursor cache in seconds. The default value is 300.</p>

## Usage Notes

- All user requests sent to database after a successful invocation of START\_CAPTURE are recorded in the given dir directory for the given duration provided that one was specified. If no duration was specified, the capture lasts indefinitely until the [FINISH\\_CAPTURE Procedure](#) is executed.
- A workload capture once started continues to record user requests across database instance shutdowns and startups for the specified duration, or until FINISH\_CAPTURE is executed, whichever occurs first.
- One can use workload filters (as described with regard to the [ADD\\_FILTER Procedures](#)) to capture only a subset of the user requests sent to the database. By default, when no workload filters are defined, all user requests are captured.
- Workload that is initiated from Oracle Database background processes (such as SMON, PMON, MMON) and Oracle Database Scheduler Jobs (as detailed in the [DBMS\\_SCHEDULER](#) package) is not captured, no matter how the workload filters are defined. These activities should happen automatically on an appropriately configured replay system.
- By default, all database instances that were started up in RESTRICTED mode using STARTUP RESTRICT are UNRESTRICTED upon a successful invocation of START\_CAPTURE Use FALSE for the auto\_unrestrict input parameter, if you do not want this behavior.
- It is important to have a well-defined starting point for the workload so that the replay system can be restored to that point before initiating a replay of the captured workload. To have a well-defined starting point for the workload capture, it is preferable not to have any active user sessions when START\_CAPTURE is executed. If ongoing sessions have ongoing transactions, those

transactions are not replayed properly in subsequent database replays, since only that part of the transaction whose calls were executed after `START_CAPTURE` are replayed.

---

---

## DBMS\_WORKLOAD\_REPLAY

The DBMS\_WORKLOAD\_REPLAY package provides an interface to replay a workload capture that was originally created by way of the [DBMS\\_WORKLOAD\\_CAPTURE](#) package. Typically, the DBMS\_WORKLOAD\_CAPTURE package will be used in the production system to capture a production workload, and the DBMS\_WORKLOAD\_REPLAY package will be subsequently used in a test system to replay the captured production workload for testing purposes.

**See Also:** *Oracle Database Real Application Testing User's Guide* for more information about "Database Replay"

This chapter contains the following topics:

- [Using DBMS\\_WORKLOAD\\_REPLAY](#)
  - Security Model
- [Summary of DBMS\\_WORKLOAD\\_REPLAY Subprograms](#)

---

## Using DBMS\_WORKLOAD\_REPLAY

- [Security Model](#)

## Security Model

The following code describes the minimum set of privileges required to

- Create directory objects
- Operate the interface provided by the DBMS\_WORKLOAD\_CAPTURE and DBMS\_WORKLOAD\_REPLAY packages
- Act as a replay client user (wrc someuser/somepassword or wrc USER=someuser PASSWORD=somepassword)

```
DROP USER rom1 CASCADE;
CREATE USER rom1 IDENTIFIED BY rom1;

GRANT EXECUTE ON DBMS_WORKLOAD_CAPTURE TO rom1;
GRANT EXECUTE ON DBMS_WORKLOAD_REPLAY TO rom1;

GRANT CREATE SESSION TO rom1;
GRANT CREATE ANY DIRECTORY TO rom1;
GRANT SELECT_CATALOG_ROLE TO rom1;
GRANT BECOME USER TO rom1;
```

Appropriate OS permissions are required to access and manipulate files and directories on both the capture and replay system. This means that the Oracle process(es) and the OS user performing the capture or replay must be able to access and manipulate at least one common directory accessible from the host where the instance is running. Additionally, the OS user performing the replay must be able to execute wrc on hosts that will be used for the replay clients and be able to access the file system appropriately to be able to copy the capture to the replay clients' hosts if required.

## Summary of DBMS\_WORKLOAD\_REPLAY Subprograms

This table lists the package subprograms in alphabetical order.

**Table 160–1 DBMS\_WORKLOAD\_REPLAY Package Subprograms**

Subprogram	Description
<a href="#">ADD_FILTER Procedure</a> on page 160-6	Adds a filter to replay only a subset of the captured workload
<a href="#">CALIBRATE Function</a> on page 160-7	Operates on a processed workload capture directory to estimate the number of hosts and workload replay clients needed to faithfully replay the given workload
<a href="#">CANCEL_REPLAY Procedure</a> on page 160-8	Cancels the workload replay in progress
<a href="#">COMPARE_PERIOD_REPORT Procedure</a> on page 160-9	Generates a report comparing a replay to its capture or to another replay of the same capture
<a href="#">COMPARE_SQLSET_Report Function</a> on page 160-10	Generates a report comparing a sqlset captured during replay to one captured during workload capture or to one captured during another replay of the same capture
<a href="#">CREATE_FILTER_SET Procedure</a> on page 160-11	Uses the replay filters added to create a set of filters to use against the replay in <code>replay_dir</code>
<a href="#">DELETE_FILTER Procedure</a> on page 160-12	Deletes the named filter
<a href="#">DELETE_REPLAY_INFO Procedure</a> on page 160-13	Deletes the rows in <code>DBA_WORKLOAD_REPLAYS</code> that corresponds to the given workload replay ID
<a href="#">EXPORT_AWR Procedure</a> on page 160-14	Exports the AWR snapshots associated with a given replay ID
<a href="#">GET_DIVERGING_STATEMENT Function</a> on page 160-15	Exports the AWR snapshots associated with a given replay ID
<a href="#">GET_REPLAY_INFO Function</a> on page 160-16	Retrieves information about the workload capture and the history of all the workload replay attempts from the related directory
<a href="#">GET_REPLAY_TIMEOUT Procedure</a> on page 160-17	Retrieves the replay timeout setting
<a href="#">IMPORT_AWR Function</a> on page 160-18	Imports the AWR snapshots associated with a given replay ID
<a href="#">INITIALIZE_REPLAY Procedure</a> on page 160-19	Initializes replay, and loads specific data produced during processing into the database
<a href="#">IS_REPLAY_PAUSED Function</a> on page 160-20	Reports whether the replay is currently paused
<a href="#">PAUSE_REPLAY Procedure</a> on page 160-21	Pauses the in-progress workload replay
<a href="#">POPULATE_DIVERGENCE Procedure</a> on page 160-21	Precomputes the divergence information for the given call, stream, or the whole replay so that the <a href="#">GET_DIVERGING_STATEMENT Function</a> returns as quickly as possible for the precomputed calls
<a href="#">PREPARE_REPLAY Procedure</a> on page 160-23	Puts the RDBMS in a special "Prepare" mode

**Table 160-1 (Cont.) DBMS\_WORKLOAD\_REPLAY Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">PROCESS_CAPTURE Procedure</a> on page 160-27	Processes the workload capture found in <code>capture_dir</code> in place
<a href="#">REMAP_CONNECTION Procedure</a> on page 160-28	Remaps the captured connection to a new one so that the user sessions can connect to the database in a desired way during workload replay
<a href="#">REPORT Function</a> on page 160-29	Generates a report on the given workload replay
<a href="#">RESUME_REPLAY Procedure</a> on page 160-30	Resumes a paused workload replay
<a href="#">REUSE_REPLAY_FILTER_SET Procedure</a> on page 160-31	Reuses filters in the specified filter set as if each were added using the <a href="#">ADD_FILTER Procedure</a>
<a href="#">REUSE_REPLAY_FILTER_SET Procedure</a> on page 160-31	Sets an advanced parameter for replay besides the ones used with the <a href="#">PREPARE_REPLAY Procedure</a>
<a href="#">SET_REPLAY_TIMEOUT Procedure</a> on page 160-33	Sets the replay timeout setting
<a href="#">START_REPLAY Procedure</a> on page 160-34	Starts the workload replay
<a href="#">USE_FILTER_SET Procedure</a> on page 160-35	Uses the given filter set that has been created by calling the <a href="#">CREATE_FILTER_SET Procedure</a> to filter the current replay

## ADD\_FILTER Procedure

This procedure adds a filter to replay only a subset of the captured workload.

The procedure adds a new filter that is used in the next replay filter set created using the [CREATE\\_FILTER\\_SET Procedure](#). This filter will be considered an "INCLUSION" or "EXCLUSION" filter depending on the argument passed to CREATE\_FILTER\_SET when creating the filter set.

### Syntax

```
DBMS_WORKLOAD_REPLAY.ADD_FILTER (
    fname          IN VARCHAR2,
    fattribute     IN VARCHAR2,
    fvalue        IN VARCHAR2);
```

```
DBMS_WORKLOAD_REPLAY.ADD_FILTER (
    fname          IN VARCHAR2,
    fattribute     IN VARCHAR2,
    fvalue        IN NUMBER);
```

### Parameters

**Table 160–2 ADD\_FILTER Procedure Parameters**

Parameter	Description
fname	Name of the filter. Can be used to delete the filter later if it is not required. (Mandatory)
fattribute	(Mandatory) Specifies the attribute on which the filter is defined as one of the following values: <ul style="list-style-type: none"> <li>■ USER - type STRING</li> <li>■ MODULE - type STRING</li> <li>■ ACTION - type STRING</li> <li>■ PROGRAM - type STRING</li> <li>■ SERVICE - type STRING</li> <li>■ CONNECTION_STRING - type STRING</li> </ul>
fvalue	Specifies the value to which the given 'attribute' must be equal to for the filter to be considered active. Wildcards such as '%' are acceptable for all attributes that are of type STRING. (Mandatory)



## CALIBRATE Function

This function operates on a processed workload capture directory to estimate the number of hosts and workload replay clients needed to faithfully replay the given workload. This function returns the results as an XML CLOB.

### Syntax

```
DBMS_WORKLOAD_REPLAY.CALIBRATE (
  capture_dir          IN VARCHAR2,
  process_per_cpu      IN BINARY_INTEGER DEFAULT 4,
  threads_per_process  IN BINARY_INTEGER DEFAULT 50)
RETURN CLOB;
```

### Parameters

**Table 160–3 CALIBRATE Function Parameters**

Parameter	Description
capture_dir	Name of the directory object that points to the (case sensitive) OS directory that contains processed capture data
process_per_cpu	Maximum number of process allowed per CPU (default is 4)
threads_per_process	Maximum number of threads allowed per process (default is 50)

### Return Values

Returns a CLOB formatted as XML, that contains:

- Information about the capture
- Current database version
- Input parameters to this function
- Number of CPUs and replay clients needed to replay the given workload
- Information about the sessions captured (total number and maximum concurrency)

### Usage Notes

- Prerequisite: The input workload capture was already processed using the [PROCESS\\_CAPTURE Procedure](#) in the same database version.
- This procedure will return the same results as the workload replay client in calibrate mode, which can be run as follows.

```
$ wrc mode=calibrate replaydir=
```

## CANCEL\_REPLAY Procedure

This procedure cancels workload replay in progress. All the external replay clients (WRC) will automatically be notified to stop issuing the captured workload and exit.

### Syntax

```
DBMS_WORKLOAD_REPLAY.CANCEL_REPLAY (  
    error_msg    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 160–4 CANCEL\_REPLAY Procedure Parameters**

Parameter	Description
error_msg	An optional reason for cancelling the replay can be passed which will be recorded into DBA_WORKLOAD_REPLAYS.ERROR_MESSAGE. DEFAULT is NULL

### Usage Notes

Prerequisite: A call to the [INITIALIZE\\_REPLAY Procedure](#), or [PREPARE\\_REPLAY Procedure](#), or [START\\_REPLAY Procedure](#) was already issued.

## COMPARE\_PERIOD\_REPORT Procedure

This procedure generates a report comparing a replay to its capture or to another replay of the same capture.

### Syntax

```
DBMS_WORKLOAD_REPLAY.COMPARE_PERIOD_REPORT (
  replay_id1  IN  NUMBER,
  replay_id2  IN  NUMBER,
  format      IN  VARCHAR2,
  result      OUT CLOB );
```

### Parameters

**Table 160–5 COMPARE\_PERIOD\_REPORT Procedure Parameters**

Parameter	Description
replay_id1	First ID of the workload replay whose report is requested
replay_id2	Second ID of the workload replay whose report is requested. If this is NULL, then the comparison is done with the capture.
format	Specifies the report format. Valid values are DBMS_WORKLOAD_CAPTURE.TYPE_HTML and DBMS_WORKLOAD_CAPTURE.TYPE_XML.
result	Output of the report (CLOB)

## COMPARE\_SQLSET\_REPORT Function

This procedure generates a report comparing a sqlset captured during replay to one captured during workload capture or to one captured during another replay of the same capture.

---



---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```
DBMS_WORKLOAD_REPLAY.COMPARE_SQLSET_REPORT (
  replay_id1    IN NUMBER,
  replay_id2    IN NUMBER,
  format        IN VARCHAR2,
  r_level       IN VARCHAR2 := 'ALL',
  r_sections    IN VARCHAR2 := 'ALL',
  result        OUT CLOB )
RETURN VARCHAR2;
```

### Parameters

**Table 160–6 COMPARE\_SQLSET\_REPORT Function Parameters**

Parameter	Description
replay_id1	First ID of the workload replay after a change
replay_id2	Second ID of the workload replay before a change. If this is NULL, then the comparison is done with the capture.
format	Specifies the report format. Valid values are DBMS_WORKLOAD_CAPTURE.TYPE_HTML, DBMS_WORKLOAD_CAPTURE.TYPE_XML and DBMS_WORKLOAD_CAPTURE.TYPE_TEXT.
r_level	See level parameter in the <a href="#">REPORT_ANALYSIS_TASK Function</a> in the DBMS_SQLPA package
r_sections	See section parameter in the <a href="#">REPORT_ANALYSIS_TASK Function</a> in the DBMS_SQLPA package
result	Output of the report (CLOB)

## CREATE\_FILTER\_SET Procedure

This procedure uses the replay filters added since the previous successful call to create a set of filters to use against the replay in `replay_dir`. After the procedure has completed and replay initiated, the newly-created filter set can be used to filter the replay in `replay_dir` by calling the [USE\\_FILTER\\_SET Procedure](#).

### Syntax

```
DBMS_WORKLOAD_REPLAY.CREATE_FILTER_SET(
    replay_dir      IN  VARCHAR2,
    filter_set      IN  VARCHAR2,
    default_action  IN  VARCHAR2 DEFAULT 'INCLUDE');
```

### Parameters

**Table 160–7 CREATE\_FILTER\_SET Procedure Parameters**

Parameter	Description
<code>replay_dir</code>	Object directory of the replay to be filtered
<code>filter_set</code>	Name of the filter set to create (to use in <a href="#">USE_FILTER_SET Procedure</a> )
<code>default_action</code>	<p>Can be either <code>INCLUDE</code> or <code>EXCLUDE</code>. Determines whether, by default, every captured call must be replayed or not. Also determines, whether the workload filters specified must be considered as <code>INCLUSION</code> filters or <code>EXCLUSION</code> filters.)</p> <p>If it is <code>INCLUDE</code>, then by default all captured calls will be replayed, except for the part of the workload defined by the filters. In this case, all the filters that were specified using the <a href="#">ADD_FILTER Procedure</a> will be treated as <code>EXCLUSION</code> filters, and will determine the workload that will not be replayed.</p> <p>If it is <code>EXCLUDE</code>, then by default no captured call to the database will be replayed, except for the part of the workload defined by the filters. In this case, all the filters that were specified using the <a href="#">ADD_FILTER Procedure</a> will be treated as <code>INCLUSION</code> filters, and will determine the workload that will be replayed.</p> <p>Default: <code>INCLUDE</code> and all the filters specified will be assumed to be <code>EXCLUSION</code> filters</p>

### Usage Notes

This operation needs to be done when no replay is initialized, prepared, or in progress.

## DELETE\_FILTER Procedure

This procedure deletes the named filter.

### Syntax

```
DBMS_WORKLOAD_REPLAY.DELETE_FILTER(  
    fname    IN  VARCHAR2);
```

### Parameters

**Table 160–8** *DELETE\_FILTER Procedure Parameters*

Parameter	Description
fname	Name of the filter that must be deleted (Mandatory)

## DELETE\_REPLAY\_INFO Procedure

This procedure deletes the rows in DBA\_WORKLOAD\_REPLAYS that correspond to the given workload replay ID.

### Syntax

```
DBMS_WORKLOAD_REPLAY.DELETE_REPLAY_INFO (  
    replay_id    IN NUMBER);
```

### Parameters

**Table 160–9** *DELETE\_REPLAY\_INFO Procedure Parameters*

Parameter	Description
replay_id	ID of the workload replay that needs to be deleted. Corresponds to DBA_WORKLOAD_REPLAYS.ID (Mandatory)

## EXPORT\_AWR Procedure

This procedure exports the AWR snapshots associated with a stipulated replay ID.

### Syntax

```
DBMS_WORKLOAD_REPLAY.EXPORT_AWR (  
    replay_id    IN NUMBER);
```

### Parameters

**Table 160–10** EXPORT\_AWR Function Parameters

Parameter	Description
replay_id	ID of the replay whose AWR snapshots are to be exported (Mandatory)

### Usage Notes

- At the end of each replay, the corresponding AWR snapshots are automatically exported. Consequently, there is no need to do this manually after a workload replay is complete, unless the automatic EXPORT\_AWR invocation failed.
- This procedure will work only if the corresponding workload replay was performed in the current database (meaning that the corresponding row in DBA\_WORKLOAD\_REPLAYS was not created by calling the [GET\\_REPLAY\\_INFO Function](#)) and the AWR snapshots that correspond to that replay time period are still available.



## GET\_DIVERGING\_STATEMENT Function

This function retrieves information about a diverging call, including the statement text, the SQL ID, and the binds.

### Syntax

```
DBMS_WORKLOAD_REPLAY.GET_DIVERGING_STATEMENT (
    replay_id    IN NUMBER,
    stream_id    IN NUMBER,
    call_counter IN NUMBER)
RETURN CLOB;
```

### Parameters

**Table 160–11** GET\_DIVERGING\_STATEMENT Function Parameters

Parameter	Description
replay_id	ID of the replay in which that call diverged
stream_id	Stream ID of the diverging call
call_counter	Call counter of the diverging call

### Usage Notes

- Returns a CLOB formatted as XML that contains:
  - SQL ID
  - SQL Text
  - Bind information: position, name and value
- This function will silently invoke the [POPULATE\\_DIVERGENCE Procedure](#) to read the information from the capture files. Therefore, if divergence has not been populated, the first call to this function for a particular diverging call might take longer, especially in very large captures.

## GET\_REPLAY\_INFO Function

This function retrieves information about the workload capture and the history of all the workload replay attempts from the stipulated directory.

### Syntax

```
DBMS_WORKLOAD_REPLAY.GET_REPLAY_INFO (  
    dir    IN VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 160–12** GET\_REPLAY\_INFO Function Parameters

Parameter	Description
dir	Name of the workload replay directory object (case sensitive). (Mandatory)

### Return Values

The procedure returns the `CAPTURE_ID`, which can be associated with both `DBA_WORKLOAD_CAPTURE_ID` and `DBA_WORKLOAD_REPLAYS.CAPTURE.ID` to access the imported information.

### Usage Notes

- The procedure first imports a row into `DBA_WORKLOAD_CAPTURES` which will contain information about the capture. It then imports a row for every replay attempt retrieved from the given replay directory into `DBA_WORKLOAD_REPLAYS`.
- The procedure will not insert new rows to `DBA_WORKLOAD_CAPTURES` and `DBA_WORKLOAD_REPLAYS` if these views already contain rows describing the capture and replay history present in the given directory.

## GET\_REPLAY\_TIMEOUT Procedure

This procedure gets the replay timeout setting.

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```
DBMS_WORKLOAD_REPLAY.GET_REPLAY_TIMEOUT (
  enabled      OUT  BOOLEAN,
  min_delay    OUT  NUMBER,
  max_delay    OUT  NUMBER,
  delay_factor OUT  NUMBER);
```

### Parameters

**Table 160–13** GET\_REPLAY\_TIMEOUT Function Parameters

Parameter	Description
enabled	TRUE if the timeout action is enabled, FALSE otherwise.
min_delay	Lower bound of call delay in minutes. The replay action is activated only when the delay is equal to or more than min_delay.
max_delay	Upper bound of call delay in minutes. The timeout action throws ORA-15569 when the delay is more than max_delay.
delay_factor	Factor for the call delay that is between min_delay and max_delay. The timeout action throws ORA-15569 when the current replay elapsed time is more than the product of capture elapsed time and delay_factor.

### Usage Notes

This procedure can be called anytime during replay.

## IMPORT\_AWR Function

This procedure imports the AWR snapshots from a given replay.

### Syntax

```
DBMS_WORKLOAD_REPLAY.IMPORT_AWR (  
    replay_id      IN  NUMBER,  
    staging_schema IN  VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 160–14** *IMPORT\_AWR Function Parameters*

Parameter	Description
replay_id	ID of the replay whose AWR snapshots must be exported. (Mandatory)
staging_schema	Name of a valid schema in the current database which can be used as a staging area while importing the AWR snapshots from the replay directory to the SYS AWR schema. The SYS schema is not a valid input. (Mandatory)

### Return Values

Returns the new randomly generated database ID that was used to import the AWR snapshots. The same value can be found in the `AWR_DBID` column in the `DBA_WORKLOAD_REPLAYS` view.

### Usage Notes

- This procedure will work provided those AWR snapshots were exported earlier from the original replay system using the [EXPORT\\_AWR Procedure](#).
- `IMPORT_AWR` will fail if the `staging_schema` provided as input contains any tables with the same name as any of the AWR tables, such as `WRM$_SNAPSHOT` or `WRH$_PARAMETER`. Please drop any such tables in the `staging_schema` before invoking `IMPORT_AWR`.

## INITIALIZE\_REPLAY Procedure

This procedure puts the database state in INIT for REPLAY mode, and loads data into the replay system that is required before preparing for the replay (by executing the [PAUSE\\_REPLAY Procedure](#)).

### Syntax

```
DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY (
  replay_name      IN  VARCHAR2,
  replay_dir       IN  VARCHAR2);
```

### Parameters

**Table 160–15 INITIALIZE\_REPLAY Procedure Parameters**

Parameter	Description
replay_name	Name of the workload replay. Every replay of a processed workload capture can be given a name. (Mandatory)
replay_dir	Name of the directory object that points to the OS directory (case sensitive) that contains processed capture data

### Usage Notes

- Prerequisites:
  - Workload capture was already processed using the [PROCESS\\_CAPTURE Procedure](#) in the same database version.
  - Database state has been logically restored to what it was at the beginning of the original workload capture.
- The subprogram loads data into the replay system that is required before preparing for the replay by calling the [PAUSE\\_REPLAY Procedure](#).

For instance, during capture the user may record the connection string each session used to connect to the server. The [INITIALIZE\\_REPLAY Procedure](#) loads this data and allows the user to re-map the recorded connection string to new connection strings or service points.

Elaborating on the example described in the [PROCESS\\_CAPTURE Procedure](#), the user could invoke the following:

```
DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY('replay foo #1', 'rec_dir');
```

This command will load up the connection map and by default will set all replay time connection strings to be equal to NULL. A NULL replay time connection string means that the workload replay clients (WRCs) will connect to the default host as determined by the replay client's runtime environment settings. The user can change a particular connection string to a new one (or a new service point) for replay by using the [REMAP\\_CONNECTION Procedure](#).

## IS\_REPLAY\_PAUSED Function

This function reports whether the replay is currently paused.

### Syntax

```
DBMS_WORKLOAD_REPLAY.IS_REPLAY_PAUSED  
RETURN BOOLEAN;
```

### Return Values

Returns `TRUE` if the [PAUSE\\_REPLAY Procedure](#) has been called successfully and the [RESUME\\_REPLAY Procedure](#) has not been called yet.

### Usage Notes

A call to the [START\\_REPLAY Procedure](#) must have already been issued as a pre-requisite.

## PAUSE\_REPLAY Procedure

This procedure pauses the in-progress workload replay. All subsequent user calls from the replay clients will be stalled until either a call to the [RESUME\\_REPLAY Procedure](#) is issued or the replay is cancelled.

### Syntax

```
DBMS_WORKLOAD_REPLAY.PAUSE_REPLAY;
```

### Usage Notes

- Prerequisite: A call to the [START\\_REPLAY Procedure](#) must have already been issued.
- User calls that were already in-progress when PAUSE\_REPLAY was issued will be allowed to run to completion. Only subsequent user calls, when issued, will be paused.

## POPULATE\_DIVERGENCE Procedure

This procedure precomputes the divergence information for the given call, stream, or the whole replay so that the [GET\\_DIVERGING\\_STATEMENT Function](#) returns as quickly as possible for the precomputed calls.

### Syntax

```
DBMS_WORKLOAD_REPLAY.POPULATE_DIVERGENCE (  
    replay_id    IN    NUMBER,  
    stream_id    IN    NUMBER DEFAULT NULL,  
    call_counter IN    NUMBER DEFAULT NULL);
```

### Parameters

**Table 160–16** *POPULATE\_DIVERGENCE Procedure Parameters*

Parameter	Description
replay_id	ID of the replay
stream_id	Stream ID of the diverging call. If NULL is provided, then divergence information will be precomputed for all diverging calls in the given replay.
call_counter	Call counter of the diverging call. If NULL is provided, then divergence information will be precomputed for all diverging calls in the given stream.



## PREPARE\_REPLAY Procedure

This procedure puts the database state in PREPARE FOR REPLAY mode.

---

**Note:** The functionality associated with the `capture_sts` and `sts_cap_interval` parameters is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```
DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY (
    synchronization          IN BOOLEAN    DEFAULT TRUE,
    connect_time_scale      IN NUMBER     DEFAULT 100,
    think_time_scale        IN NUMBER     DEFAULT 100,
    think_time_auto_correct IN BOOLEAN    DEFAULT TRUE,
    scale_up_multiplier     IN NUMBER     DEFAULT 1,
    capture_sts             IN BOOLEAN    DEFAULT FALSE,
    sts_cap_interval        IN NUMBER     DEFAULT 300);
```

```
DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY (
    synchronization          IN VARCHAR2   DEFAULT 'SCN',,
    connect_time_scale      IN NUMBER     DEFAULT 100,
    think_time_scale        IN NUMBER     DEFAULT 100,
    think_time_auto_correct IN BOOLEAN    DEFAULT TRUE,
    scale_up_multiplier     IN NUMBER     DEFAULT 1,
    capture_sts             IN BOOLEAN    DEFAULT FALSE,
    sts_cap_interval        IN NUMBER     DEFAULT 300);
```

### Parameters

**Table 160–17 PREPARE\_REPLAY Procedure Parameters**

Parameter	Description
<code>synchronization</code>	<p>Turns synchronization ON or OFF during workload replay. When synchronization is ON, the COMMIT order observed during the original workload capture will be preserved during replay. Every action that is replayed will be executed ONLY AFTER all of its dependent COMMITs (all COMMITs that were issued before the given action in the original workload capture) have finished execution. DEFAULT is TRUE which preserves commit order.</p> <p>When synchronization is OBJECT_ID, a more advanced synchronization scheme is used. Every action that is replayed will be executed only after the relevant COMMITs have finished executing. The relevant COMMITs are the ones that were issued before the given action in the original workload capture and that had modified at least one of the database objects the given action is referencing (either implicitly or explicitly). This object_id scheme has the same logical property of making sure that any action will see the same data it saw during capture, but will allow more concurrency during replays for the actions that do not touch the same objects/tables. DEFAULT VALUE: SCN, preserve commit order. For legacy reasons, there is a boolean version of this procedure:</p> <ul style="list-style-type: none"> <li>▪ TRUE means 'SCN'</li> <li>▪ FALSE means 'OFF'</li> </ul>

**Table 160–17 (Cont.) PREPARE\_REPLAY Procedure Parameters**

Parameter	Description
<code>connect_time_scale</code>	Scales the time elapsed between the instant the workload capture was started and session connects with the given value. The input is interpreted as a % value. Can potentially be used to increase or decrease the number of concurrent users during the workload replay. <b>DEFAULT VALUE</b> is 100. See <a href="#">Application of the connect_time_scale Parameter</a> .
<code>think_time_scale</code>	Scales the time elapsed between two successive user calls from the same session. The input is interpreted as a % value. Can potentially be used to increase or decrease the number of concurrent users during the workload replay. <b>DEFAULT VALUE</b> is 100. See <a href="#">Application of the think_time_scale Parameter</a> .
<code>think_time_auto_correct</code>	Auto corrects the think time between calls appropriately when user calls takes longer time to complete during replay than how long the same user call took to complete during the original capture. <b>DEFAULT</b> is TRUE which is to reduce think time if replay goes slower than capture. See <a href="#">Application of the think_time_auto_correct Parameter</a>
<code>scale_up_multiplier</code>	Defines the number of times the query workload is scaled up during replay. Each captured session is replayed concurrently as many times as the value of the <code>scale_up_multiplier</code> . However, only one of the sessions in each set of identical replay sessions executes both queries and updates. The remaining sessions only execute queries.
<code>capture_sts</code>	If this parameter is TRUE, then a SQL tuning set capture is also started in parallel with workload replay. The resulting SQL tuning set can be exported using the <a href="#">EXPORT_AWR Procedure</a> along with the AWR data. Currently, parallel STS capture is not supported in an Oracle RAC environment. So, this parameter has no effect in that context. The calling user must have the appropriate privileges ('ADMINISTER SQL TUNING SET'). The default value is FALSE.
<code>sts_cap_interval</code>	Specifies the capture interval of the SQL set capture from the cursor cache in seconds. The default value is 300.

## Usage Notes

- Prerequisites:
  - The database has been initialized for replay using the [INITIALIZE\\_REPLAY Procedure](#).
  - Any capture time connection strings that require remapping have been already done using the [REMAP\\_CONNECTION Procedure](#).
- One or more external replay clients (WRC) can be started once the `PREPARE_REPLAY` procedure has been executed.
- With regard to `scale_up_multiplier`:
  - One replay session (base session) of each set of identical sessions will replay every call from the capture as usual
  - The remaining sessions (scale-up sessions) will only replay calls that are read-only. Thus, DDL, DML, and PL/SQL calls that modified the database will be skipped. `SELECT FOR UPDATE` statements are also skipped.
  - Readonly calls from the scale-up are synchronized appropriately and obey the timings defined by `think_time_scale`, `connect_time_scale`, and

think\_time\_auto\_correct. Also, the queries are made to wait for the appropriate commits.

- No replay data or error divergence records will be generated for the scale-up sessions.
- All base or scale-up sessions that replay the same capture file will connect from the same workload replay client.

## Examples

### Application of the connect\_time\_scale Parameter

If the following was observed during the original workload capture:

```
12:00 : Capture was started
12:10 : First session connect (10m after)
12:30 : Second session connect (30m after)
12:42 : Third session connect (42m after)
```

If the connect\_time\_scale is 50, then the session connects will happen as follows:

```
12:00 : Replay was started with 50% connect time scale
12:05 : First session connect ( 5m after)
12:15 : Second session connect (15m after)
12:21 : Third session connect (21m after)
```

If the connect\_time\_scale is 200, then the session connects will happen as follows:

```
12:00 : Replay was started with 200% connect time scale
12:20 : First session connect (20m after)
13:00 : Second session connect (60m after)
13:24 : Third session connect (84m after)
```

### Application of the think\_time\_scale Parameter

If the following was observed during the original workload capture:

```
12:00 : User SCOTT connects
12:10 : First user call issued (10m after completion of prevcall)
12:14 : First user call completes in 4mins
12:30 : Second user call issued (16m after completion of prevcall)
12:40 : Second user call completes in 10m
12:42 : Third user call issued ( 2m after completion of prevcall)
12:50 : Third user call completes in 8m
```

If the think\_time\_scale is 50 during the workload replay, then the user calls will look something like below:

```
12:00 : User SCOTT connects
12:05 : First user call issued 5 mins (50% of 10m) after the completion of
        previous call
12:10 : First user call completes in 5m (takes a minute longer)
12:18 : Second user call issued 8 mins (50% of 16m) after the completion of prev
        call
12:25 : Second user call completes in 7m (takes 3 minutes less)
12:26 : Third user call issued 1 min (50% of 2m) after the completion of prev
        call
12:35 : Third user call completes in 9m (takes a minute longer)
```

### Application of the think\_time\_auto\_correct Parameter

If the following was observed during the original workload capture:

```
12:00 : User SCOTT connects
12:10 : First user call issued (10m after completion of prevcall)
12:14 : First user call completes in 4m
12:30 : Second user call issued (16m after completion of prevcall)
12:40 : Second user call completes in 10m
12:42 : Third user call issued ( 2m after completion of prevcall)
12:50 : Third user call completes in 8m
```

If the `think_time_scale` is 100 and the `think_time_auto_correct` is TRUE during the workload replay, then the user calls will look something like below:

```
12:00 : User SCOTT connects
12:10 : First user call issued 10 mins after the completion of prev call
12:15 : First user call completes in 5m (takes 1 minute longer)
12:30 : Second user call issued 15 mins (16m minus the extra time of 1m the prev
       call took) after the completion of prev call
12:44 : Second user call completes in 14m (takes 4 minutes longer)
12:44 : Third user call issued immediately (2m minus the extra time of 4m the prev
       call took) after the completion of prev call
12:52 : Third user call completes in 8m
```

## PROCESS\_CAPTURE Procedure

This procedure processes the workload capture found in `capture_dir` in place.

### Syntax

```
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE (
    capture_dir          IN  VARCHAR2);
```

### Parameters

**Table 160–18** *PROCESS\_CAPTURE Procedure Parameters*

Parameter	Description
<code>capture_dir</code>	Name of the workload capture directory object (case sensitive). The directory object must point to a valid OS directory that has the appropriate permissions. New files will be added to this directory. (Mandatory)

### Usage Notes

- This subprogram analyzes the workload capture found in the `capture_dir` and creates new workload replay specific metadata files that are required to replay the given workload capture. It only creates new files and does not modify any files that were originally created during the workload capture. Therefore, this procedure can be run multiple times on the same capture directory, such as when the procedure encounters unexpected errors or is cancelled by the user.
- Once this procedure runs successfully, the `capture_dir` can be used as input to the [INITIALIZE\\_REPLAY Procedure](#) in order to replay the captured workload present in `capture_dir`.
- Before a workload capture can be replayed in a particular database version, the capture needs to be "processed" using this `PROCESS_CAPTURE` procedure in the same database version. Once created, a processed workload capture can be used to replay the captured workload multiple times in the same database version.

For example, suppose workload "foo" was captured in `rec_dir` in Oracle database version 10.2.0.5. In order to replay the workload "foo" in version 11.1.0.1 the workload needs to be processed in version 11.1.0.1. The following procedure needs to be executed in a 11.1.0.1 database in order to process the capture directory `rec_dir`:

```
DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE('rec_dir');
```

Now, `rec_dir` contains a valid 11.1.0.1 processed workload capture that can be used to replay the workload "foo" in 11.1.0.1 databases as many times as required.

## REMAP\_CONNECTION Procedure

This procedure remaps the captured connection to a new one so that the user sessions can connect to the database in a desired way during workload replay.

### Syntax

```
DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (
  connection_id      IN NUMBER,
  replay_connection  IN VARCHAR2);
```

### Parameters

**Table 160–19 REMAP\_CONNECTION Procedure Parameters**

Parameter	Description
connection_id	ID of the connection to be remapped. Corresponds to DBA_WORKLOAD_CONNECTION_MAP.CONN_ID
replay_connection	New connection string to be used during replay

### Usage Notes

- By default, all instances of `replay_connection` will be equal to `NULL`. When `replay_connection` is `NULL` (default), replay sessions will connect to the default host as determined by the replay client's runtime environment. Consequently, if no capture time connect strings are remapped, then all the replay sessions will simply connect to the default host to replay the workload.
- A valid `replay_connection` must specify a connect identifier or a service point. See the *Oracle Database Net Services Reference* for ways to specify connect identifiers (such as net service names, database service names, and net service aliases) and naming methods that can be used to resolve a connect identifier to a connect descriptor.
- An error is returned if no row matches the given `connection_id`.
- Use the `DBA_WORKLOAD_CONNECTION_MAP` view to review all the connection strings that will be used by the subsequent workload replay, and also to examine connection string remappings used for previous workload replays.

## REPORT Function

This function generates a report on the stipulated workload replay.

### Syntax

```
DBMS_WORKLOAD_REPLAY.REPORT (
    replay_id      IN NUMBER,
    format         IN VARCHAR2 )
RETURN CLOB;
```

### Parameters

**Table 160–20** *REPORT Function Parameters*

Parameter	Description
replay_id	Specifies the ID of the workload replay whose report is requested. (Mandatory)
format	Specifies the report format. Valid values are XML, HTML, or TEXT. (Mandatory)

### Return Values

The report body in the desired format returned as a CLOB

**Table 160–21** *Constants Used by Report Function*

Constant	Type	Value	Description
TYPE_HTML	VARCHAR2(4)	'HTML'	Generates the HTML version of the report
TYPE_TEXT	VARCHAR2(4)	'TEXT'	Use this as input to the <code>format</code> argument to generate the text version of the report.
TYPE_XML	VARCHAR2(3)	'XML'	Generates the XML version of the report

## RESUME\_REPLAY Procedure

This procedure resumes a paused workload replay.

### Syntax

```
DBMS_WORKLOAD_REPLAY.RESUME_REPLAY;
```

### Usage Notes

Prerequisite: A call to the [PAUSE\\_REPLAY Procedure](#) must have already been issued.



## REUSE\_REPLAY\_FILTER\_SET Procedure

This procedure reuses filters in the specified filter set as if each were added using the [ADD\\_FILTER Procedure](#). Each call adds one filter set, which is a collection of individual filters on various attributes. Also, a new filter rule can be added, and existing filter can be deleted before invoking the [CREATE\\_FILTER\\_SET Procedure](#) to create a new filter set.

---

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

---

### Syntax

```
DBMS_WORKLOAD_REPLAY.REUSE_REPLAY_FILTER_SET(  
    replay_dir IN VARCHAR2,  
    filter_set IN VARCHAR2);
```

### Parameters

**Table 160–22 REUSE\_REPLAY\_FILTER\_SET Procedure Parameters**

Parameter	Description
replay_dir	Capture ID of the existing filter set with which it is associated
filter_set	Name of the filter set to be reused

## SET\_ADVANCED\_PARAMETER Procedure

This procedure sets an advanced parameter for replay besides the ones used with the [PREPARE\\_REPLAY Procedure](#). The advanced parameters control aspects of the replay that are more specialized. The advanced parameters are reset to their default values after the replay has finished.

### Syntax

```
DBMS_WORKLOAD_REPLAY.SET_ADVANCED_PARAMETER (
  pname      IN   VARCHAR2,
  pvalue     IN   VARCHAR2);
```

```
DBMS_WORKLOAD_REPLAY.SET_ADVANCED_PARAMETER (
  pname      IN   VARCHAR2,
  pvalue     IN   NUMBER);
```

```
DBMS_WORKLOAD_REPLAY.SET_ADVANCED_PARAMETER (
  pname      IN   VARCHAR2,
  pvalue     IN   BOOLEAN);
```

### Parameters

**Table 160–23 SET\_ADVANCED\_PARAMETER Procedure Parameters**

Parameter	Description
pname	Name of the parameter (case insensitive)
pvalue	Value of the parameter

### Usage Notes

The current parameters and values that can be used are:

```
'DO_NO_WAIT_COMMITS': (default: FALSE)
```

This parameter controls whether the COMMITs issued by replay sessions will be NOWAIT. The default value for this parameter is FALSE. In this case all the COMMITs are issued with the mode they were captured (wait, no-wait, batch, no-batch). If the parameter is set to TRUE, then all COMMITs are issued in no-wait mode. This is useful in cases where the replay is becoming noticeably slow because of a high volume of concurrent COMMITs. Setting the parameter to TRUE will significantly decrease the waits on the 'log file sync' event during the replay with respect to capture.

## SET\_REPLAY\_TIMEOUT Procedure

This procedure sets the replay timeout setting. The purpose is to abort user calls that might make the replay much slower or even cause a replay hang.

---



---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```
DBMS_WORKLOAD_REPLAY.SET_REPLAY_TIMEOUT (
  enabled      OUT  BOOLEAN DEFAULT TRUE,
  min_delay    OUT  NUMBER DEFAULT 10,
  max_delay    OUT  NUMBER DEFAULT 120,
  delay_factor OUT  NUMBER DEFAULT 8);
```

### Parameters

**Table 160–24** SET\_REPLAY\_TIMEOUT Function Parameters

Parameter	Description
enabled	TRUE to enable the timeout action, and FALSE to disable. Default = TRUE.
min_delay	Lower bound of call delay in minutes. The replay action is activated only when the delay is equal to or more than min_delay. Default = 10.
max_delay	Upper bound of call delay in minutes. The timeout action throws ORA-15569 when the delay is more than max_delay. Default = 120.
delay_factor	Factor for the call delay that is between min_delay and max_delay. The timeout action throws ORA-15569 when the current replay elapsed time is more than the product of capture elapsed time and delay_factor. Default = 8.

### Usage Notes

- This procedure can be called anytime during replay.
- Call delay is defined as the difference of call elapsed time between replay and capture if replay elapsed time is longer.
- Once a replay timeout action is enabled, a user call will exit with ORA-15569 if it has been delayed more than the condition specified by the replay action. The call and its error will be reported as error divergence.
- Replay timeout operates as follows:
  - The timeout action has no effect if it is not enabled.
  - If the call delay in minutes is less than a lower bound specified by parameter min\_delay, then the timeout action is non-operational.
  - If the delay in minutes is more than an upper bound specified by parameter max\_delay, the timeout action will abort the user call and throw ORA-15569.
  - For delay that is between the lower bound and upper bound, the user call will abort with ORA-15569 only when the current replay elapsed time is more than the product of capture elapsed time and parameter delay\_factor.

## START\_REPLAY Procedure

This procedure starts the workload replay. All the external replay clients (WRC) that are currently connected to the replay database will automatically be notified and those replay clients (WRC) will begin issuing the captured workload.

### Syntax

```
DBMS_WORKLOAD_REPLAY.START_REPLAY;
```

### Usage Notes

- Prerequisites:
  - The call to the [PREPARE\\_REPLAY Procedure](#) was already issued.
  - Enough number of external replay clients (WRC) that can faithfully replay the captured workload have already been started. The status of such external replay clients can be monitored using `V$WORKLOAD_REPLAY_CLIENTS`.
- Use the WRC's `CALIBRATE` mode to determine the number of replay clients that might be required to faithfully replay the captured workload. For example:  

```
$ wrc mode=calibrate replaydir=.
```

## USE\_FILTER\_SET Procedure

This procedure uses the given filter set that has been created by calling the [CREATE\\_FILTER\\_SET Procedure](#) to filter the current replay.

### Syntax

```
DBMS_WORKLOAD_REPLAY.USE_FILTER_SET(  
    filter_set    IN VARCHAR2);
```

### Parameters

**Table 160–25** *USE\_FILTER\_SET Procedure Parameters*

Parameter	Description
filter_set	Name of the filter set use in this replay

### Usage Notes

This procedure must be called after the replay has been initialized and before it is prepared.



---

---

## DBMS\_WORKLOAD\_REPOSITORY

The DBMS\_WORKLOAD\_REPOSITORY package lets you manage the Workload Repository, performing operations such as managing snapshots and baselines.

**See Also:** *Oracle Database Performance Tuning Guide* for more information about the "Automatic Workload Repository"

The chapter contains the following topics:

- [Using DBMS\\_WORKLOAD\\_REPOSITORY](#)
  - Examples
- [Data Structures](#)
  - Object Types
  - Table Types
- [Summary of DBMS\\_WORKLOAD\\_REPOSITORY Subprograms](#)

---

## Using DBMS\_WORKLOAD\_REPOSITORY

This section contains topics which relate to using the DBMS\_WORKLOAD\_REPOSITORY package.

- [Examples](#)



## Examples

This example shows how to generate an AWR text report with the DBMS\_WORKLOAD\_REPOSITORY package for database identifier 1557521192, instance id 1, snapshot ids 5390 and 5391 and with default options.

```
-- make sure to set line size appropriately
-- set linesize 152
SELECT output FROM TABLE(
  DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
    1557521192, 1, 5390, 5392) );
```

You can call the DBMS\_WORKLOAD\_REPOSITORY packaged functions directly as in the example, but Oracle recommends you use the corresponding supplied SQL script (`awrrpt.sql` in this case) for the packaged function, which prompts the user for required information.

## Data Structures

The `DBMS_WORKLOAD_REPOSITORY` package defines an object and associated table types.

### OBJECT Types

- [AWR\\_BASELINE\\_METRIC\\_TYPE](#) Object Type

### TABLE Types

- [AWR\\_BASELINE\\_METRIC\\_TYPE\\_TABLE](#) Table Type
- [AWRRPT\\_INSTANCE\\_LIST\\_TYPE](#) Table Type

## AWR\_BASELINE\_METRIC\_TYPE Object Type

This type shows the values of the metrics corresponding to a baseline.

### Syntax

```
TYPE breakpoint_info AS OBJECT (
  baseline_name      VARCHAR2(64),
  dbid               NUMBER NOT NULL,
  instance_number    NUMBER NOT NULL,
  beg_time           DATE NOT NULL,
  end_time           DATE NOT NULL,
  metric_id          NUMBER NOT NULL,
  metric_name        VARCHAR2(64) NOT NULL,
  metric_unit        VARCHAR2(64) NOT NULL,
  num_interval       NUMBER NOT NULL,
  interval_size      NUMBER NOT NULL,
  average            NUMBER NOT NULL,
  minimum            NUMBER NOT NULL,
  maximum            NUMBER NOT NULL);
```

### Fields

**Table 161–1** *RUNTIME\_INFO* Fields

Field	Description
baseline_name	Name of the Baseline
dbid	Database ID for the snapshot
instance_number	Instance number for the snapshot
beg_time	Begin time of the interval
end_time	End time of the interval
metric_id	Metric ID
metric_name	Metric name
metric_unit	Unit of measurement
num_interval	Number of intervals observed
interval_size	Interval size (in hundredths of a second)
average	Average over the period
minimum	Minimum value observed
maximum	Maximum value observed

## **AWR\_BASELINE\_METRIC\_TYPE\_TABLE Table Type**

This type is used by the [SELECT\\_BASELINE\\_METRICS Function](#).

### **Syntax**

```
CREATE TYPE awr_baseline_metric_type_table AS TABLE OF awr_baseline_metric_type;
```

## **AWRRPT\_INSTANCE\_LIST\_TYPE Table Type**

This type provides an alternative to a comma-separated list.

### **Syntax**

```
CREATE TYPE awrrpt_instance_list_type AS TABLE OF NUMBER;
```

---

## Summary of DBMS\_WORKLOAD\_REPOSITORY Subprograms

**Table 161–2 DBMS\_WORKLOAD\_REPOSITORY Package Subprograms**

Subprogram	Description
<a href="#">ADD_COLORED_SQL Procedure</a> on page 161-10	Adds a colored SQL ID
<a href="#">ASH_GLOBAL_REPORT_HTML Function</a> on page 161-11	Displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in HTML format.
<a href="#">ASH_GLOBAL_REPORT_TEXT Function</a> on page 161-13	Displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in Text format.
<a href="#">ASH_REPORT_HTML Function</a> on page 161-15	Displays the ASH report in HTML
<a href="#">ASH_REPORT_TEXT Function</a> on page 161-18	Displays the ASH report in text
<a href="#">AWR_DIFF_REPORT_HTML Function</a> on page 161-21	Displays the AWR Diff-Diff report in HTML
<a href="#">AWR_DIFF_REPORT_TEXT Function</a> on page 161-22	Displays the AWR Diff-Diff report in text
<a href="#">AWR_GLOBAL_DIFF_REPORT_HTML Functions</a> on page 161-23	Displays the Global AWR Compare Periods Report in HTML
<a href="#">AWR_GLOBAL_DIFF_REPORT_TEXT Functions</a> on page 161-24	Displays the Global AWR Compare Periods Report in text
<a href="#">AWR_GLOBAL_REPORT_HTML Functions</a> on page 161-25	Displays the Global AWR report in HTML
<a href="#">AWR_GLOBAL_REPORT_TEXT Functions</a> on page 161-26	Displays the Global AWR report in text
<a href="#">AWR_REPORT_HTML Function</a> on page 161-27	Displays the AWR report in HTML
<a href="#">AWR_REPORT_TEXT Function</a> on page 161-28	Displays the AWR report in text
<a href="#">AWR_SQL_REPORT_HTML Function</a> on page 161-29	Displays the AWR SQL Report in HTML format
<a href="#">AWR_SQL_REPORT_TEXT Function</a> on page 161-30	Displays the AWR SQL Report in text format
<a href="#">CREATE_BASELINE Functions &amp; Procedures</a> on page 161-31	Creates a single baseline
<a href="#">CREATE_BASELINE_TEMPLATE Procedures</a> on page 161-31	Creates a baseline template
<a href="#">CREATE_SNAPSHOT Function and Procedure</a> on page 161-34	Creates a manual snapshot immediately
<a href="#">DROP_BASELINE Procedure</a> on page 161-35	Drops a range of snapshots
<a href="#">DROP_BASELINE_TEMPLATE Procedure</a> on page 161-36	Removes a baseline template that is no longer needed

**Table 161-2 (Cont.) DBMS\_WORKLOAD\_REPOSITORY Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DROP_SNAPSHOT_RANGE Procedure</a> on page 161-37	Activates service
<a href="#">MODIFY_SNAPSHOT_SETTINGS Procedures</a> on page 161-38	Modifies the snapshot settings
<a href="#">MODIFY_BASELINE_WINDOW_SIZE Procedure</a> on page 161-40	Modifies the window size for the Default Moving Window Baseline
<a href="#">RENAME_BASELINE Procedure</a> on page 161-42	Renames a baseline
<a href="#">SELECT_BASELINE_METRICS Function</a> on page 161-43	Shows the values of the metrics corresponding to a baseline

## ADD\_COLORED\_SQL Procedure

This procedure adds a colored SQL ID. If an SQL ID is colored, it will be captured in every snapshot, independent of its level of activities (so that it does not have to be a TOP SQL). Capture occurs if the SQL is found in the cursor cache at snapshot time.

To uncolor the SQL, invoke the [REMOVE\\_COLORED\\_SQL Procedure](#).

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(  
    sql_id          IN VARCHAR2,  
    dbid           IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–3** ADD\_COLORED\_SQL Procedure Parameters

Parameter	Description
sql_id	13-character external SQL ID
dbid	Optional dbid, defaults to Local DBID



## ASH\_GLOBAL\_REPORT\_HTML Function

This table function displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in HTML format.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_GLOBAL_REPORT_HTML (
  l_dbid          IN NUMBER,
  l_inst_num      IN VARCHAR2((1023),
  l_btime         IN DATE,
  l_etime         IN DATE,
  l_options       IN NUMBER          DEFAULT 0,
  l_slot_width    IN NUMBER          DEFAULT 0,
  l_sid           IN NUMBER          DEFAULT NULL,
  l_sql_id        IN VARCHAR2        DEFAULT NULL,
  l_wait_class    IN VARCHAR2        DEFAULT NULL,
  l_service_hash  IN NUMBER          DEFAULT NULL,
  l_module        IN VARCHAR2        DEFAULT NULL,
  l_action        IN VARCHAR2        DEFAULT NULL,
  l_client_id     IN VARCHAR2        DEFAULT NULL,
  l_plsql_entry   IN VARCHAR2        DEFAULT NULL,
  l_data_src      IN NUMBER          DEFAULT 0)
RETURN awrrpt_html_type_table PIPELINED;
```

### Parameters

**Table 161–4 ASH\_GLOBAL\_REPORT\_HTML Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	List of instances (such as '1, 2, 3'), or NULL to report on all instances in the database
l_btime	The 'begin time'
l_etime	The 'end time'
l_options	Report level (currently not used)
l_slot_width	Specifies (in seconds) how wide the slots used in the "Top Activity" section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.
l_sid	Session ID (see Usage Notes)
l_sql_id	SQL ID (see Usage Notes)
l_wait_class	Wait class name (see Usage Notes)
l_service_hash	Service name hash (see Usage Notes)
l_module	Module name (see Usage Notes)
l_action	Action name (see Usage Notes)
l_client_id	Client ID for end-to-end backtracing (see Usage Notes)
l_plsql_entry	PL/SQL entry point (see Usage Notes)
l_data_src	Ignored since the report works off of data on disk only

## Return Values

The output will be one column of VARCHAR2 (1500).

## Usage Notes

- You can call the function directly but Oracle recommends you use the `ashrpti.sql` script which prompts users for the required information.
- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as `SQL_ID 'abcdefghij123'` pass that `sql_id` value to the `l_sql_id` argument:

```
l_sql_id => 'abcdefghij123'
```

Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to generate an ASH report on `MODULE "PAYROLL"` and `ACTION "PROCESS"`, use the following predicate:

```
l_module => 'PAYROLL', l_action => 'PROCESS'
```

Valid SQL wildcards can be used in all the arguments that are of type VARCHAR2.

**Table 161–5 ASH\_GLOBAL\_REPORT\_HTML: Wildcards Allowed (or Not) in Arguments**

Argument Name	Comment	Wildcard Allowed
<code>l_sid</code>	Session ID (for example, <code>V\$SESSION.SID</code> )	No
<code>l_sql_id</code>	SQL ID (for example, <code>V\$SQL.SQL_ID</code> )	Yes
<code>l_wait_class</code>	Wait class name (for example, <code>V\$EVENT_NAME.WAIT_CLASS</code> )	Yes
<code>l_service_hash</code>	Service name hash (for example, <code>V\$ACTIVE_SERVICES.NAME_HASH</code> )	No
<code>l_module</code>	Module name (for example, <code>V\$SESSION.MODULE</code> )	Yes
<code>l_action</code>	Action name (for example, <code>V\$SESSION.ACTION</code> )	Yes
<code>l_client_id</code>	Client ID for end-to-end backtracing (for example, <code>V\$SESSION.CLIENT_IDENTIFIER</code> )	Yes
<code>l_data_src</code>	Wildcards are not allowed for <code>l_data_src</code> as it is of numeric datatype	No

## ASH\_GLOBAL\_REPORT\_TEXT Function

This table function Displays a global or Oracle Real Application Clusters (Oracle RAC) ASH Spot report in Text format.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_GLOBAL_REPORT_TEXT(
  l_dbid          IN VARCHAR2(1023),
  l_inst_num     IN NUMBER,
  l_btime       IN DATE,
  l_etime       IN DATE,
  l_options      IN NUMBER          DEFAULT 0,
  l_slot_width  IN NUMBER          DEFAULT 0,
  l_sid         IN NUMBER          DEFAULT NULL,
  l_sql_id      IN VARCHAR2        DEFAULT NULL,
  l_wait_class  IN VARCHAR2        DEFAULT NULL,
  l_service_hash IN NUMBER          DEFAULT NULL,
  l_module     IN VARCHAR2        DEFAULT NULL,
  l_action     IN VARCHAR2        DEFAULT NULL,
  l_client_id  IN VARCHAR2        DEFAULT NULL,
  l_plsql_entry IN VARCHAR2        DEFAULT NULL,
  l_data_src   IN NUMBER          DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–6 ASH\_GLOBAL\_REPORT\_TEXT Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	List of instances (such as '1, 2, 3'), or NULL to report on all instances in the database
l_btime	The 'begin time'
l_etime	The 'end time'
l_options	Report level (currently not used)
l_slot_width	Specifies (in seconds) how wide the slots used in the "Top Activity" section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.
l_sid	Session ID (see Usage Notes)
l_sql_id	SQL ID (see Usage Notes)
l_wait_class	Wait class name (see Usage Notes)
l_service_hash	Service name hash (see Usage Notes)
l_module	Module name (see Usage Notes)
l_action	Action name (see Usage Notes)
l_client_id	Client ID for end-to-end backtracing (see Usage Notes)
l_plsql_entry	PL/SQL entry point (see Usage Notes)
l_data_src	Ignored since the report works off of data on disk only

## Return Values

The output will be one column of VARCHAR2 (320).

## Usage Notes

- You can call the function directly but Oracle recommends you use the `ashrpti.sql` script which prompts users for the required information.
- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as `SQL_ID 'abcdefghij123'` pass that `sql_id` value to the `l_sql_id` argument:

```
l_sql_id => 'abcdefghij123'
```

**Table 161–7 ASH\_GLOBAL\_REPORT\_TEXT: Wildcards Allowed (or Not) in Arguments**

Argument Name	Comment	Wildcard Allowed
<code>l_sid</code>	Session ID (for example, <code>V\$SESSION.SID</code> )	No
<code>l_sql_id</code>	SQL ID (for example, <code>V\$SQL.SQL_ID</code> )	Yes
<code>l_wait_class</code>	Wait class name (for example, <code>V\$EVENT_NAME.WAIT_CLASS</code> )	Yes
<code>l_service_hash</code>	Service name hash (for example, <code>V\$ACTIVE_SERVICES.NAME_HASH</code> )	No
<code>l_module</code>	Module name (for example, <code>V\$SESSION.MODULE</code> )	Yes
<code>l_action</code>	Action name (for example, <code>V\$SESSION.ACTION</code> )	Yes
<code>l_client_id</code>	Client ID for end-to-end backtracing (for example, <code>V\$SESSION.CLIENT_IDENTIFIER</code> )	Yes
<code>l_plsql_entry</code>	PL/SQL entry point (for example, <code>"SYS.DBMS_LOB.*"</code> )	Yes
<code>l_data_src</code>	Wildcards are not allowed for <code>l_data_src</code> as it is of numeric datatype	No

- Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to generate an ASH report on `MODULE "PAYROLL"` and `ACTION "PROCESS"`, use the following predicate:

```
l_module => 'PAYROLL', l_action => 'PROCESS'
```

Valid SQL wildcards can be used in all the arguments that are of type VARCHAR2.

## ASH\_REPORT\_HTML Function

This table function displays the ASH Spot report in HTML.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(
  l_dbid          IN NUMBER,
  l_inst_num     IN NUMBER,
  l_btime        IN DATE,
  l_etime        IN DATE,
  l_options      IN NUMBER          DEFAULT 0,
  l_slot_width   IN NUMBER          DEFAULT 0,
  l_sid          IN NUMBER          DEFAULT NULL,
  l_sql_id       IN VARCHAR2        DEFAULT NULL,
  l_wait_class   IN VARCHAR2        DEFAULT NULL,
  l_service_hash IN NUMBER          DEFAULT NULL,
  l_module       IN VARCHAR2        DEFAULT NULL,
  l_action       IN VARCHAR2        DEFAULT NULL,
  l_client_id    IN VARCHAR2        DEFAULT NULL,
  l_plsql_entry  IN VARCHAR2        DEFAULT NULL,
  l_data_src     IN NUMBER          DEFAULT 0)
RETURN awrrpt_html_type_table PIPELINED;
```

### Parameters

**Table 161–8 ASH\_REPORT\_HTML Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	Instance number
l_btime	The 'begin time'
l_etime	The 'end time'
l_options	Report level (currently not used)
l_slot_width	Specifies (in seconds) how wide the slots used in the "Top Activity" section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.
l_sid	Session ID (see Usage Notes)
l_sql_id	SQL ID (see Usage Notes)
l_wait_class	Wait class name (see Usage Notes)
l_service_hash	Service name hash (see Usage Notes)
l_module	Module name (see Usage Notes)
l_action	Action name (see Usage Notes)
l_client_id	Client ID for end-to-end backtracing (see Usage Notes)
l_plsql_entry	PL/SQL entry point (see Usage Notes)

**Table 161–8 (Cont.) ASH\_REPORT\_HTML Parameters**

Parameter	Description
<code>l_data_src</code>	Can be used to specify a data source (see Usage Notes) <ul style="list-style-type: none"> <li>■ 1 =&gt; memory (<code>V\$ACTIVE_SESSON_HISTORY</code>)</li> <li>■ 2 =&gt; disk (<code>DBA_HIST_ACTIVE_SESS_HISTORY</code>)</li> <li>■ 0 =&gt; both. This is the default value. Here, the begin and end time parameters are used to get the samples from the appropriate data source, which can be memory, disk, or both.</li> </ul>

## Return Values

The output will be one column of `VARCHAR2 (500)`.

## Usage Notes

- You can call the function directly but Oracle recommends you use the `ashrpti.sql` script which prompts users for the required information.
- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as `SQL_ID 'abcdefghij123'` pass that `sql_id` value to the `l_sql_id` argument:

```
l_sql_id => 'abcdefghij123'
```

Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to generate an ASH report on `MODULE "PAYROLL"` and `ACTION "PROCESS"`, use the following predicate:

```
l_module => 'PAYROLL', l_action => 'PROCESS'
```

Valid SQL wildcards can be used in all the arguments that are of type `VARCHAR2`.

**Table 161–9 ASH\_REPORT\_HTML: Wildcards Allowed (or Not) in Arguments**

Argument Name	Comment	Wildcard Allowed
<code>l_sid</code>	Session ID (for example, <code>V\$SESSION.SID</code> )	No
<code>l_sql_id</code>	SQL ID (for example, <code>V\$SQL.SQL_ID</code> )	Yes
<code>l_wait_class</code>	Wait class name (for example, <code>V\$EVENT_NAME.WAIT_CLASS</code> )	Yes
<code>l_service_hash</code>	Service name hash (for example, <code>V\$ACTIVE_SERVICES.NAME_HASH</code> )	No
<code>l_module</code>	Module name (for example, <code>V\$SESSION.MODULE</code> )	Yes
<code>l_action</code>	Action name (for example, <code>V\$SESSION.ACTION</code> )	Yes
<code>l_client_id</code>	Client ID for end-to-end backtracing (for example, <code>V\$SESSION.CLIENT_IDENTIFIER</code> )	Yes

**Table 161-9 (Cont.) ASH\_REPORT\_HTML: Wildcards Allowed (or Not) in Arguments**

<b>Argument Name</b>	<b>Comment</b>	<b>Wildcard Allowed</b>
l_data_src	Wildcards are not allowed for l_data_src as it is of numeric datatype	No

## ASH\_REPORT\_TEXT Function

This table function displays the ASH Spot report in text.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_TEXT(
  l_dbid          IN NUMBER,
  l_inst_num     IN NUMBER,
  l_btime        IN DATE,
  l_etime        IN DATE,
  l_options      IN NUMBER      DEFAULT 0,
  l_slot_width   IN NUMBER      DEFAULT 0,
  l_sid          IN NUMBER      DEFAULT NULL,
  l_sql_id       IN VARCHAR2    DEFAULT NULL,
  l_wait_class   IN VARCHAR2    DEFAULT NULL,
  l_service_hash IN NUMBER      DEFAULT NULL,
  l_module       IN VARCHAR2    DEFAULT NULL,
  l_action       IN VARCHAR2    DEFAULT NULL,
  l_client_id    IN VARCHAR2    DEFAULT NULL,
  l_plsql_entry  IN VARCHAR2    DEFAULT NULL,
  l_data_src     IN NUMBER      DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–10 ASH\_REPORT\_TEXT Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	Instance number
l_btime	The 'begin time'
l_etime	The 'end time'
l_options	Report level (currently not used)
l_slot_width	Specifies (in seconds) how wide the slots used in the "Top Activity" section of the report should be. This argument is optional, and if it is not specified the time interval between l_btime and l_etime is appropriately split into not more than 10 slots.
l_sid	Session ID (see Usage Notes)
l_sql_id	SQL ID (see Usage Notes)
l_wait_class	Wait class name (see Usage Notes)
l_service_hash	Service name hash (see Usage Notes)
l_module	Module name (see Usage Notes)
l_action	Action name (see Usage Notes)
l_client_id	Client ID for end-to-end backtracing (see Usage Notes)
l_plsql_entry	PL/SQL entry point (see Usage Notes)



**Table 161–10 (Cont.) ASH\_REPORT\_TEXT Parameters**

Parameter	Description
<code>l_data_src</code>	Can be used to specify a data source (see Usage Notes) <ul style="list-style-type: none"> <li>■ 1 =&gt; memory (<code>V\$ACTIVE_SESSION_HISTORY</code>)</li> <li>■ 2 =&gt; disk (<code>DBA_HIST_ACTIVE_SESS_HISTORY</code>)</li> <li>■ 0 =&gt; both. This is the default value. Here, the begin and end time parameters are used to get the samples from the appropriate data source, which can be memory, disk, or both.</li> </ul>

## Return Values

The output will be one column of `VARCHAR2 (80)`.

## Usage Notes

- You can call the function directly but Oracle recommends you use the `ashrpti.sql` script which prompts users for the required information.
- By default, the report uses the begin and end time parameters (`l_btime` and `l_etime`, respectively) to find all rows in that time range either from memory, or disk, or both. However, using `l_data_src`, one can explicitly specify one of those data sources. For example, to generate an ASH report on all rows between `l_btime` and `l_etime` found in memory, use

```
l_data_src => 1
```

Similarly, to generate a report on samples found only on disk, use

```
l_data_src => 2
```

- The unspecified optional arguments are used to generate an ASH Reports that specify 'report targets' such as a SQL statement, or a session, or a particular Service/Module combination. These arguments are specified to restrict the ASH rows that would be used to generate the report. For example, to generate an ASH report on a particular SQL statement, such as `SQL_ID 'abcdefghijkl123'` pass that `sql_id` value to the `l_sql_id` argument:

```
l_sql_id => 'abcdefghijkl123'
```

**Table 161–11 ASH\_REPORT\_TEXT: Wildcards Allowed (or Not) in Arguments**

Argument Name	Comment	Wildcard Allowed
<code>l_sid</code>	Session ID (for example, <code>V\$SESSION.SID</code> )	No
<code>l_sql_id</code>	SQL ID (for example, <code>V\$SQL.SQL_ID</code> )	Yes
<code>l_wait_class</code>	Wait class name (for example, <code>V\$EVENT_NAME.WAIT_CLASS</code> )	Yes
<code>l_service_hash</code>	Service name hash (for example, <code>V\$ACTIVE_SERVICES.NAME_HASH</code> )	No
<code>l_module</code>	Module name (for example, <code>V\$SESSION.MODULE</code> )	Yes
<code>l_action</code>	Action name (for example, <code>V\$SESSION.ACTION</code> )	Yes

**Table 161–11 (Cont.) ASH\_REPORT\_TEXT: Wildcards Allowed (or Not) in Arguments**

Argument Name	Comment	Wildcard Allowed
<code>l_client_id</code>	Client ID for end-to-end backtracing (for example, <code>V\$SESSION.CLIENT_IDENTIFIER</code> )	Yes
<code>l_plsql_entry</code>	PL/SQL entry point (for example, <code>"SYS.DBMS_LOB.*"</code> )	Yes
<code>l_data_src</code>	Wildcards are not allowed for <code>l_data_src</code> as it is of numeric datatype	No

- Any combination of those optional arguments can be passed in, and only rows in ASH that satisfy all of those 'report targets' will be used. If multiple 'report targets' are specified, AND conditional logic is used to connect them. For example, to generate an ASH report on MODULE "PAYROLL" and ACTION "PROCESS", use the following predicate:

```
l_module => 'PAYROLL', l_action => 'PROCESS'
```

Valid SQL wildcards can be used in all the arguments that are of type VARCHAR2.

## AWR\_DIFF\_REPORT\_HTML Function

This table function displays the AWR Compare Periods report in HTML.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_HTML (
  dbid1      IN NUMBER,
  inst_num1  IN NUMBER,
  bid1       IN NUMBER,
  eid1       IN NUMBER,
  dbid2      IN NUMBER,
  inst_num2  IN NUMBER,
  bid2       IN NUMBER,
  eid2       IN NUMBER)
RETURN awrd rpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–12 AWR\_DIFF\_REPORT\_HTML Parameters**

Parameter	Description
dbid1	1st database identifier
inst_num1	1st instance number
bid1	1st 'Begin Snapshot' ID
eid1	1st 'End Snapshot' ID
dbid2	2nd database identifier
inst_num2	2nd instance number
bid2	2nd 'Begin Snapshot' ID
eid2	2nd 'End Snapshot' ID

### Return Values

The output will be one column of VARCHAR2 (500).

### Usage Notes

You can call the function directly but Oracle recommends you use the `awrd rpt . sql` script which prompts users for the required information.

## AWR\_DIFF\_REPORT\_TEXT Function

This table function displays the AWR Compare Periods report in text.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_TEXT (
  dbid1      IN NUMBER,
  inst_num1  IN NUMBER,
  bid1       IN NUMBER,
  eid1       IN NUMBER,
  dbid2      IN NUMBER,
  inst_num2  IN NUMBER,
  bid2       IN NUMBER,
  eid2       IN NUMBER)
RETURN awrd rpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–13 AWR\_DIFF\_REPORT\_TEXT Parameters**

Parameter	Description
dbid1	1st database identifier
inst_num1	1st instance number
bid1	1st 'Begin Snapshot' ID
eid1	1st 'End Snapshot' ID
dbid2	2nd database identifier
inst_num2	2nd instance number
bid2	2nd 'Begin Snapshot' ID
eid2	2nd 'End Snapshot' ID

### Return Values

The output will be one column of VARCHAR2 (500).

### Usage Notes

You can call the function directly but Oracle recommends you use the `awrd rpt . sql` script which prompts users for the required information.

## AWR\_GLOBAL\_DIFF\_REPORT\_HTML Functions

This table function displays Global AWR Compare Periods Report in HTML format.

The first overload accepts comma-separated lists of instance numbers for `inst_num1` and `inst_num2`. No leading zeroes are allowed and there is a limit of 1023 characters.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_HTML (
  dbid1      IN      NUMBER,
  inst_num1  IN      AWRRTPT_INSTANCE_LIST_TYPE,
  bid1       IN      NUMBER,
  eid1       IN      NUMBER,
  dbid2      IN      NUMBER,
  inst_num2  IN      AWRRTPT_INSTANCE_LIST_TYPE,
  bid2       IN      NUMBER,
  eid2       IN      NUMBER)
RETURN awrrpt_html_type_table PIPELINED;
```

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_HTML (
  dbid1      IN      NUMBER,
  inst_num1  IN      VARCHAR2,
  bid1       IN      NUMBER,
  eid1       IN      NUMBER,
  dbid2      IN      NUMBER,
  inst_num2  IN      VARCHAR2,
  bid2       IN      NUMBER,
  eid2       IN      NUMBER)
RETURN awrrpt_html_type_table PIPELINED;
```

### Parameters

**Table 161–14 AWR\_GLOBAL\_DIFF\_REPORT\_HTML Function Parameters**

Parameter	Description
<code>dbid1</code>	1st database identifier
<code>inst_num1</code>	1st list of instance numbers. If set to <code>NULL</code> , all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.
<code>bid1</code>	1st Begin Snap ID
<code>eid1</code>	1st End Snapshot ID
<code>dbid2</code>	2nd database identifier
<code>inst_num2</code>	2nd list of instance numbers to be included in report. If set to <code>NULL</code> , all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.
<code>bid2</code>	2nd Begin Snap ID
<code>eid2</code>	2nd End Snapshot ID

### Return Values

The output will be one column of `VARCHAR2 (1500)`.

## AWR\_GLOBAL\_DIFF\_REPORT\_TEXT Functions

This table function displays Global AWR Compare Periods Report in text format.

The first overload accepts comma-separated lists of instance numbers for `inst_num1` and `inst_num2`. No leading zeroes are allowed and there is a limit of 1023 characters.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_TEXT (
  dbid1      IN      NUMBER,
  inst_num1  IN      AWRRTPT_INSTANCE_LIST_TYPE,
  bid1       IN      NUMBER,
  eid1       IN      NUMBER,
  dbid2      IN      NUMBER,
  inst_num2  IN      AWRRTPT_INSTANCE_LIST_TYPE,
  bid2       IN      NUMBER,
  eid2       IN      NUMBER)
RETURN awrdrpt_text_type_table PIPELINED;
```

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_DIFF_REPORT_TEXT (
  dbid1      IN      NUMBER,
  inst_num1  IN      VARCHAR2,
  bid1       IN      NUMBER,
  eid1       IN      NUMBER,
  dbid2      IN      NUMBER,
  inst_num2  IN      VARCHAR2,
  bid2       IN      NUMBER,
  eid2       IN      NUMBER)
RETURN awrdrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–15 AWR\_GLOBAL\_DIFF\_REPORT\_TEXT Functions Parameters**

Parameter	Description
<code>dbid1</code>	1st database identifier
<code>inst_num1</code>	1st list of instance numbers. If set to <code>NULL</code> , all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.
<code>bid1</code>	1st Begin Snap ID
<code>eid1</code>	1st End Snapshot ID
<code>dbid2</code>	2nd database identifier
<code>inst_num2</code>	2nd list of instance numbers to be included in report. If set to <code>NULL</code> , all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.
<code>bid2</code>	2nd Begin Snap ID
<code>eid2</code>	2nd End Snapshot ID

### Return Values

The output will be one column of `VARCHAR2 (320)`.

## AWR\_GLOBAL\_REPORT\_HTML Functions

This table function displays the Global AWR report in HTML.

The first overload accepts a comma-separated list of instance numbers. No leading zeroes are allowed and there is a limit of 1023 characters.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_HTML (
  l_dbid      IN    NUMBER,
  l_inst_num  IN    AWRPT_INSTANCE_LIST_TYPE,
  l_bid       IN    NUMBER,
  l_eid       IN    NUMBER,
  l_options   IN    NUMBER DEFAULT 0)
RETURN awrrpt_html_type_table PIPELINED;
```

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_HTML (
  l_dbid      IN    NUMBER,
  l_inst_num  IN    VARCHAR2,
  l_bid       IN    NUMBER,
  l_eid       IN    NUMBER,
  l_options   IN    NUMBER DEFAULT 0)
RETURN awrrpt_html_type_table PIPELINED;
```

### Parameters

**Table 161–16 AWR\_GLOBAL\_REPORT\_HTML Function Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	List of instance numbers to be included in report. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.
l_bid	Begin Snap ID
l_eid	End Snapshot ID
l_options	

### Return Values

The output will be one column of VARCHAR2 (1500).

## AWR\_GLOBAL\_REPORT\_TEXT Functions

This table function displays the Global AWR report in text.

The first overload accepts a comma-separated list of instance numbers. No leading zeroes are allowed and there is a limit of 1023 characters

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_TEXT (
  l_dbid      IN      NUMBER,
  l_inst_num  IN      AWRRPT_INSTANCE_LIST_TYPE,
  l_bid       IN      NUMBER,
  l_eid       IN      NUMBER,
  l_options   IN      NUMBER DEFAULT 0)
RETURN awdrpt_text_type_table PIPELINED;
```

```
DBMS_WORKLOAD_REPOSITORY.AWR_GLOBAL_REPORT_TEXT (
  l_dbid      IN      NUMBER,
  l_inst_num  IN      VARCHAR2,
  l_bid       IN      NUMBER,
  l_eid       IN      NUMBER,
  l_options   IN      NUMBER DEFAULT 0)
RETURN awdrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–17 AWR\_GLOBAL\_REPORT\_TEXT Function Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	List of instance numbers to be included in report. If set to NULL, all instances for which begin and end snapshots are available, and which have not been restarted between snapshots, will be included in the report.
l_bid	Begin Snap ID
l_eid	End Snapshot ID
l_options	A flag to specify to control the output of the report. Currently, not used.

### Return Values

The output will be one column of VARCHAR2 (320).



## AWR\_REPORT\_HTML Function

This table function displays the AWR report in HTML.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(
  l_dbid      IN    NUMBER,
  l_inst_num  IN    NUMBER,
  l_bid       IN    NUMBER,
  l_eid       IN    NUMBER,
  l_options   IN    NUMBER DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–18 AWR\_REPORT\_HTML Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	Instance number
l_bid	The 'Begin Snapshot' ID
l_eid	The 'End Snapshot' ID
l_options	A flag to specify to control the output of the report. Currently, Oracle supports one value: <ul style="list-style-type: none"> <li>▪ l_options - 8. Displays the ADDM specific portions of the report. These sections include the Buffer Pool Advice, Shared Pool Advice, and PGA Target Advice.</li> </ul>

### Return Values

The output will be one column of VARCHAR2 (1500).

### Usage Notes

You can call the function directly but Oracle recommends you use the `awrrpt.sql` script which prompts users for the required information.

## AWR\_REPORT\_TEXT Function

This table function displays the AWR report in text.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT(
  l_dbid      IN    NUMBER,
  l_inst_num  IN    NUMBER,
  l_bid       IN    NUMBER,
  l_eid       IN    NUMBER,
  l_options   IN    NUMBER DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–19 AWR\_REPORT\_TEXT Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	Instance number
l_bid	The 'Begin Snapshot' ID
l_eid	The 'End Snapshot' ID
l_options	A flag to specify to control the output of the report. Currently, Oracle supports one value: <ul style="list-style-type: none"> <li>▪ l_options - 8. Displays the ADDM specific portions of the report. These sections include the Buffer Pool Advice, Shared Pool Advice, and PGA Target Advice.</li> </ul>

### Return Values

The output will be one column of VARCHAR2 (80).

### Usage Notes

You can call the function directly but Oracle recommends you use the `awrrpt.sql` script which prompts users for the required information.

## AWR\_SQL\_REPORT\_HTML Function

This table function displays the AWR SQL Report in HTML format.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_HTML(
  l_dbid      IN    NUMBER,
  l_inst_num  IN    NUMBER,
  l_bid       IN    NUMBER,
  l_eid       IN    NUMBER,
  l_sqlid     IN    VARCHAR2,
  l_options   IN    NUMBER DEFAULT 0)
RETURN awrrpt_html_type_table PIPELINED;
```

### Parameters

**Table 161–20 AWR\_SQL\_REPORT\_HTML Parameters**

Parameter	Description
l_dbid	Database identifier
l_inst_num	Instance number
l_bid	The 'Begin Snapshot' ID
l_eid	The 'End Snapshot' ID
l_sqlid	SQL ID of statement to be analyzed
l_options	A flag to specify to control the output of the report. Currently, not used.

### Return Values

The output will be one column of VARCHAR2 (500).

### Usage Notes

You can call the function directly but Oracle recommends you use the `awrsqrpt.sql` script which prompts users for the required information.

## AWR\_SQL\_REPORT\_TEXT Function

This table function displays the AWR SQL Report in text format.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.AWR_SQL_REPORT_TEXT(
  l_dbid      IN    NUMBER,
  l_inst_num  IN    NUMBER,
  l_bid       IN    NUMBER,
  l_eid       IN    NUMBER,
  l_sqlid     IN    VARCHAR2,
  l_options   IN    NUMBER DEFAULT 0)
RETURN awrrpt_text_type_table PIPELINED;
```

### Parameters

**Table 161–21 AWR\_SQL\_REPORT\_TEXT Parameters**

Parameter	Description
<code>l_dbid</code>	Database identifier
<code>l_inst_num</code>	Instance number
<code>l_bid</code>	The 'Begin Snapshot' ID
<code>l_eid</code>	The 'End Snapshot' ID
<code>l_sqlid</code>	SQL ID of statement to be analyzed
<code>l_options</code>	Flag to specify to control the output of the report. Currently, not used.

### Return Values

The output will be one column of VARCHAR2 (120).

### Usage Notes

You can call the function directly but Oracle recommends you use the `awrsqrpt.sql` script which prompts users for the required information.

## CREATE\_BASELINE Functions & Procedures

This function and procedure creates a baseline.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
  start_snap_id   IN NUMBER,
  end_snap_id     IN NUMBER,
  baseline_name   IN VARCHAR2,
  dbid            IN NUMBER DEFAULT NULL,
  expiration      IN NUMBER DEFAULT NULL);
```

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
  start_snap_id   IN NUMBER,
  end_snap_id     IN NUMBER,
  baseline_name   IN VARCHAR2,
  dbid            IN NUMBER DEFAULT NULL,
  expiration      IN NUMBER DEFAULT NULL)
RETURN NUMBER;
```

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
  start_time      IN DATE,
  end_time        IN DATE,
  baseline_name   IN VARCHAR2,
  dbid            IN NUMBER DEFAULT NULL,
  expiration      IN NUMBER DEFAULT NULL);
```

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(
  start_time      IN DATE,
  end_time        IN DATE,
  baseline_name   IN VARCHAR2,
  dbid            IN NUMBER DEFAULT NULL,
  expiration      IN NUMBER DEFAULT NULL);
RETURN NUMBER;
```

### Parameters

**Table 161–22 CREATE\_BASELINE Function & Procedure Parameters**

Parameter	Description
start_snap_id	Start snapshot sequence number for the baseline'
end_snap_id	End snapshot sequence number for the baseline
start_time	Start time for the baseline'
end_time	End time for the baseline
baseline_name	Name of baseline.
dbid	Database Identifier for baseline. If NULL, this takes the database identifier for the local database. Defaults to NULL.
expiration	Expiration in number of days for the baseline. If NULL, then expiration is infinite, meaning do not drop baseline ever. Defaults to NULL.

### Exceptions

- An error will be returned if this baseline name already exists in the system.

- The snapshot range that is specified for this interface must be an existing pair of snapshots in the database. An error will be returned if the inputted snapshots do not exist in the system.

## Examples

This example creates a baseline (named 'oltp\_peakload\_bl') between snapshots 105 and 107 for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (start_snap_id => 105,  
end_snap_id => 107,  
baseline_name => 'oltp_peakload_bl');
```

If you query the DBA\_HIST\_BASELINE view after the CREATE BASELINE action, you will see the newly created baseline in the Workload Repository.

## CREATE\_BASELINE\_TEMPLATE Procedures

This procedure specifies a template for how they would like baselines to be created for future time periods.

### Syntax

Specifies a template for generating a baseline for a single time period in the future.

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE(
  start_time          IN DATE,
  end_time            IN DATE,
  baseline_name       IN VARCHAR2,
  template_name       IN VARCHAR2,
  expiration          IN NUMBER,
  dbid                IN NUMBER DEFAULT NULL);
```

Specifies a template for creating and dropping baseline based on repeating time periods:

```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE(
  day_of_week         IN VARCHAR2,
  hour_in_day         IN NUMBER,
  duration            IN NUMBER,
  start_time          IN DATE,
  end_time            IN DATE,
  baseline_name_prefix IN VARCHAR2,
  template_name       IN VARCHAR2,
  expiration          IN NUMBER,
  dbid                IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–23 CREATE\_BASELINE\_TEMPLATE Procedure Parameters**

Parameter	Description
start_time	Start Time for the baseline to be created'
end_time	End Time for the baseline to be created
baseline_name	Name of baseline to be created
template_name	Name for the template
expiration	Expiration in number of days for the baseline. If NULL, then expiration is infinite, meaning do not drop baseline ever. Defaults to NULL
dbid	Database Identifier for baseline. If NULL, this takes the database identifier for the local database. Defaults to NULL.
day_of_week	Day of week that the baseline should repeat on. Specify one of the following values: SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY.
hour_in_day	Value of 0-23 to specify the Hour in the Day the baseline should start
duration	Duration (in number of hours) after hour in the day that the baseline should last
baseline_name_prefix	Name for baseline prefix. When creating the baseline, the name of the baseline will be the prefix appended with the date information.

## CREATE\_SNAPSHOT Function and Procedure

This function and procedure create snapshots. In the case of the function, the snapshot ID is returned.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(  
    flush_level IN VARCHAR2 DEFAULT 'TYPICAL');  
  
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(  
    flush_level IN VARCHAR2 DEFAULT 'TYPICAL')  
    RETURN NUMBER;
```

### Parameters

**Table 161–24** CREATE\_SNAPSHOT Parameters

Parameter	Description
flush_level	Flush level for the snapshot is either 'TYPICAL' or 'ALL'

### Examples

This example creates a manual snapshot at the TYPICAL level:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();
```

If you query the DBA\_HIST\_SNAPSHOT view after the CREATE\_SNAPSHOT action, you will see one more snapshot ID added to the Workload Repository.



## DROP\_BASELINE Procedure

This procedure drops a baseline.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE(
  baseline_name IN VARCHAR2,
  cascade       IN BOOLEAN DEFAULT FALSE,
  dbid          IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–25 DROP\_BASELINE Parameters**

Parameter	Description
baseline_name	Name of baseline to drop from the system
cascade	If TRUE, the pair of snapshots associated with the baseline will also be dropped. Otherwise, only the baseline is removed.
dbid	Database Identifier for baseline. If NULL, this takes the database identifier for the local database. Defaults to NULL.

### Examples

This example drops the baseline 'oltp\_peakload\_b1' without dropping the underlying snapshots:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE (
  baseline_name => 'oltp_peakload_b1');
```

If you query the DBA\_HIST\_BASELINE view after the DROP\_BASELINE action, you will see the specified baseline definition is removed. You can query the DBA\_HIST\_SNAPSHOT view to find that the underlying snapshots are left intact.

## DROP\_BASELINE\_TEMPLATE Procedure

This procedure removes a template that is no longer needed.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE_TEMPLATE(  
  template_name      IN VARCHAR2,  
  dbid               IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–26** *DROP\_BASELINE\_TEMPLATE Procedure Parameters*

Parameter	Description
template_name	Name of the template to remove
dbid	Database Identifier for baseline. If NULL, this takes the database identifier for the local database. Defaults to NULL.

## DROP\_SNAPSHOT\_RANGE Procedure

This procedure drops a range of snapshots.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE (
  low_snap_id   IN  NUMBER,
  high_snap_id  IN  NUMBER
  dbid          IN  NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–27 DROP\_SNAPSHOT\_RANGE Procedure Parameters**

Parameter	Description
low_snap_id	Low snapshot id of snapshots to drop.
high_snap_id	High snapshot id of snapshots to drop.
dbid	Database id (default to local DBID).

### Examples

This example drops the range of snapshots between snapshot id 102 to 105 for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(102, 105);
```

If you query the `dba_hist_snapshot` view after the Drop Snapshot action, you will see that snapshots 102 to 105 are removed from the Workload Repository.

## MODIFY\_SNAPSHOT\_SETTINGS Procedures

This procedure controls three aspects of snapshot generation.

- The `INTERVAL` setting affects how often snapshots are automatically captured.
- The `RETENTION` setting affects how long snapshots are retained in the Workload Repository.
- The number of SQL captured for each Top criteria. If the user manually specifies a value for Top N SQL, the AWR SQL collection will use the user-specified number for both automatic and manual snapshots.

There are two overloads. The first takes a `NUMBER` and the second takes a `VARCHAR2` for the `topnsql` argument. The differences are described under the Parameters description.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS (
    retention    IN    NUMBER    DEFAULT NULL,
    interval     IN    NUMBER    DEFAULT NULL,
    topnsql      IN    NUMBER    DEFAULT NULL,
    dbid         IN    NUMBER    DEFAULT NULL);
```

```
DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS (
    retention    IN    NUMBER    DEFAULT NULL,
    interval     IN    NUMBER    DEFAULT NULL,
    topnsql      IN    VARCHAR2,
    dbid         IN    NUMBER    DEFAULT NULL);
```

### Parameters

**Table 161–28** *MODIFY\_SNAPSHOT\_SETTINGS Procedure Parameters*

Parameter	Description
<code>retention</code>	<p>New retention time (in minutes). The specified value must be in the range of <code>MIN_RETENTION</code> (1 day) to <code>MAX_RETENTION</code> (100 years).</p> <p>If <code>ZERO</code> is specified, snapshots will be retained forever. A large system-defined value will be used as the retention setting.</p> <p>If <code>NULL</code> is specified, the old value for retention is preserved.</p> <p>NOTE: The retention setting must be greater than or equal to the window size of the 'SYSTEM_MOVING_WINDOW' baseline. If the retention needs to be less than the window size, the <a href="#">MODIFY_BASELINE_WINDOW_SIZE Procedure</a> can be used to adjust the window size.</p>
<code>interval</code>	<p>New interval setting between each snapshot, in units of minutes. The specified value must be in the range <code>MIN_INTERVAL</code> (10 minutes) to <code>MAX_INTERVAL</code> (1 year).</p> <p>If <code>ZERO</code> is specified, automatic and manual snapshots will be disabled. A large system-defined value will be used as the retention setting.</p> <p>If <code>NULL</code> is specified, the current value is preserved.</p>

**Table 161–28 (Cont.) MODIFY\_SNAPSHOT\_SETTINGS Procedure Parameters**

Parameter	Description
topnsql	<ul style="list-style-type: none"> <li>■ If NUMBER: Top N SQL size. The number of Top SQL to flush for each SQL criteria (Elapsed Time, CPU Time, Parse Calls, Shareable Memory, Version Count). The value for this setting will not be affected by the statistics/flush level and will override the system default behavior for the AWR SQL collection. The setting will have a minimum value of 30 and a maximum value of 50,000. Specifying NULL will keep the current setting.</li> <li>■ If VARCHAR2: Users are allowed to specify the following values: (DEFAULT, MAXIMUM, N), where N is the number of Top SQL to flush for each SQL criteria. Specifying DEFAULT will revert the system back to the default behavior of Top 30 for statistics level TYPICAL and Top 100 for statistics level ALL. Specifying MAXIMUM will cause the system to capture the complete set of SQL in the cursor cache. Specifying the number N is equivalent to setting the Top N SQL with the NUMBER type. Specifying NULL for this argument will keep the current setting.</li> </ul>
dbid	Database identifier in AWR for which to modify the snapshot settings. If NULL is specified, the local dbid will be used. Defaults to NULL.

## Examples

This example changes the `interval` setting to one hour and the `retention` setting to two weeks for the local database:

```
EXECUTE DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS (
    interval => 60,
    retention => 20160);
```

If you query the `DBA_HIST_WR_CONTROL` table after this procedure is executed, you will see the changes to these settings.

## MODIFY\_BASELINE\_WINDOW\_SIZE Procedure

This procedure modifies the window size for the Default Moving Window Baseline.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.MODIFY_BASELINE_WINDOW_SIZE(  
    window_size    IN    NUMBER,  
    dbid           IN    NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–29** *MODIFY\_BASELINE\_WINDOW\_SIZE Procedure Parameters*

Parameter	Description
window_size	New Window size for the default Moving Window Baseline, in number of days.
dbid	Database ID (default to local DBID)

### Usage Notes

The window size must be less than or equal to the AWR retention setting. If the window size needs to be greater than the retention setting, the [MODIFY\\_SNAPSHOT\\_SETTINGS Procedures](#) can be used to adjust the retention setting. A moving window can be set to a maximum of 13 weeks.

## REMOVE\_COLORED\_SQL Procedure

This procedure removes a colored SQL ID. After a SQL is uncolored, it will no longer be captured in a snapshot automatically, unless it makes the TOP list.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.ASH_REPORT_HTML(  
    sql_id      IN VARCHAR2,  
    dbid        IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–30 REMOVE\_COLORED\_SQL Procedure Parameters**

Parameter	Description
sql_id	13-character external SQL ID
dbid	Optional dbid, defaults to Local DBID

## RENAME\_BASELINE Procedure

This procedure renames a baseline.

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.RENAME_BASELINE(
  old_baseline_name  IN  VARCHAR2,
  new_baseline_name  IN  VARCHAR2,
  dbid               IN  NUMBER DEFAULT NULL);
```

### Parameters

**Table 161–31** *RENAME\_BASELINE Procedure Parameters*

Parameter	Description
old_baseline_name	Old baseline name
new_baseline_name	New baseline name
dbid	Database ID (default to local DBID)



## SELECT\_BASELINE\_METRICS Function

This table function shows the values of the metrics corresponding to a baseline. The table function will return an object of the [AWR\\_BASELINE\\_METRIC\\_TYPE Object Type](#).

### Syntax

```
DBMS_WORKLOAD_REPOSITORY.SELECT_BASELINE_METRICS(
  baseline_name      IN VARCHAR2,
  dbid               IN NUMBER DEFAULT NULL,
  instance_num       IN NUMBER DEFAULT NULL)
RETURN awr_baseline_metric_type_table PIPELINED;
```

### Parameters

**Table 161–32** *SELECT\_BASELINE\_METRICS Function Parameters*

Parameter	Description
baseline_name	Name of the baseline for which we would like to view metrics
dbid	Database Identifier for baseline. If NULL, then use the database identifier for the local database. Defaults to NULL.
instance_num	Instance for which number the user wants to see statistics. If NULL, show statistics for the local instance. Defaults to NULL.



The DBMS\_XA package contains the XA/Open interface for applications to call XA interface in PL/SQL. Using this package, application developers can switch or share transactions across SQL\*Plus sessions or processes using PL/SQL.

**See Also:** *Oracle Database Advanced Application Developer's Guide* for more information about "Developing Applications with Oracle XA"

The chapter contains the following topics:

- [Using DBMS\\_XA](#)
  - Overview
  - Security Model
  - Constants
  - Operational Notes
- [Data Structures](#)
  - OBJECT Types
  - TABLE Types
- [Summary of DBMS\\_XA Subprograms](#)

## Using DBMS\_XA

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Operational Notes](#)

## Overview

These subprograms allow a PL/SQL application to define a global transaction branch ID (XID) and associate or disassociate the current session with the transaction branch.

Subsequently, these transaction branches may be prepared and committed by following the two-phase commit protocol. A single-phase commit protocol is also supported if only one resource manager is involved.

Interfaces are also provided for a PL/SQL application to set the timeout values for any new global transaction branches that may start with the current session.

## Security Model

This package is created under `SYS`. Operations provided by this package are performed under the current calling user, not under the package owner `SYS`.

Any `DBMS_XA` subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user. Any `DBMS_XA` subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

`SELECT` privilege on `SYS.DBA_PENDING_TRANSACTIONS` is required for users who need to execute `XA_RECOVER` subprogram.

`FORCE ANY TRANSACTION` privilege is required for users who need to manipulate XA transactions created by other users.

## Constants

The DBMS\_XA package uses the constants shown in [Table 162–1](#) for use in the flag field of the [XA\\_START Function](#) and the [XA\\_END Function](#).

**Table 162–1 DBMS\_XA Constants for Flag Field of XA\_START & XA\_END Functions**

Name	Type	Value	Description
TMNOFLAGS	PLS_INTEGER	00000000	Indicates no flag value is selected.
TMSUCCESS	PLS_INTEGER	UTL_RAW.CAST_TO_BINARY_INTEGER('04000000')	Dissociates caller from transaction branch
TMJOIN	PLS_INTEGER	UTL_RAW.CAST_TO_BINARY_INTEGER('00200000')	Caller is joining existing transaction branch.
TMSUSPEND	PLS_INTEGER	UTL_RAW.CAST_TO_BINARY_INTEGER('02000000')	Caller is suspending, not ending, association
TMRESUME	PLS_INTEGER	UTL_RAW.CAST_TO_BINARY_INTEGER('08000000')	Caller is resuming association with suspended transaction branch.

The DBMS\_XA package uses the constants shown in [Table 162–2](#) for Possible Return Values

**Table 162–2 DBMS\_XA Constants for Possible Return Values**

Name	Type	Value	Description
XA_RBBASE	PLS_INTEGER	100	Inclusive lower bound of the rollback codes
XA_RBROLLBACK	PLS_INTEGER	XA_RBBASE	Rollback was caused by an unspecified reason
XA_RBCOMMFAIL	PLS_INTEGER	XA_RBBASE+1	Rollback was caused by a communication failure
XA_RBDEADLOCK	PLS_INTEGER	XA_RBBASE+2	Deadlock was detected
XA_RBINTEGRITY	PLS_INTEGER	XA_RBBASE+3	Condition that violates the integrity of the resources was detected
XA_RBOTHER	PLS_INTEGER	XA_RBBASE+4	Resource manager rolled back the transaction for an unlisted reason
XA_RBPROTO	PLS_INTEGER	XA_RBBASE+5	Protocol error occurred in the resource manager
XA_RBTIMEOUT	PLS_INTEGER	XA_RBBASE+6	transaction branch took long
XA_RBTRANSIENT	PLS_INTEGER	XA_RBBASE+7	May retry the transaction branch
XA_RBEND	PLS_INTEGER	XA_RBTRANSIENT	Inclusive upper bound of the rollback codes
XA_NOMIGRATE	PLS_INTEGER	9	Transaction branch may have been heuristically completed

**Table 162–2 (Cont.) DBMS\_XA Constants for Possible Return Values**

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
XA_HEURHAZ	PLS_INTEGER	8	Transaction branch may have been heuristically completed
XA_HEURCOM	PLS_INTEGER	7	Transaction branch has been heuristically committed
XA_HEURRB	PLS_INTEGER	6	Transaction branch has been heuristically rolled back
XA_HEURMIX	PLS_INTEGER	5	Some of the transaction branches have been heuristically committed, others rolled back
XA_RETRY	PLS_INTEGER	4	Routine returned with no effect and may be re-issued
XA_RDONLY	PLS_INTEGER	3	Transaction was read-only and has been committed
XA_OK	PLS_INTEGER	0	Normal execution
XAER_ASYNC	PLS_INTEGER	-2	Asynchronous operation already outstanding
XAER_RMERR	PLS_INTEGER	-3	Resource manager error occurred in the transaction branch
XAER_NOTA	PLS_INTEGER	-4	XID is not valid
XAER_INVALID	PLS_INTEGER	-5	Invalid arguments were given
XAER_PROTO	PLS_INTEGER	-6	Routine invoked in an improper context
XAER_RMFAIL	PLS_INTEGER	-7	Resource manager unavailable
XAER_DUPID	PLS_INTEGER	-8	XID already exists
XAER_OUTSIDE	PLS_INTEGER	-9	Resource manager doing work outside global transaction



## Operational Notes

In compliance with the XA specification of the X/Open CAE Standard for Distributed Transaction Processing, `XA_PREPARE/COMMIT/ROLLBACK/FORGET` may not be called when the transaction is still associated with the current session. Only after `XA_END` has been called so that there is not any transaction associated with the current session, the application may call `XA_PREPARE/COMMIT/ROLLBACK/FORGET`.

`XAER_PROTO` error is returned from `XA_PREPARE/COMMIT/ROLLBACK/FORGET` if a transaction is being associated with the current session.

Prior to calling any of the package subprograms, a connection/session must have already been established to the Oracle database server backend, or a resource manager. Resource manager identifiers are not supported. If multiple resource managers are involved, multiple connections/sessions must be pre-established to each resource manager before calling any the package subprograms. If multiple connections/sessions are established during the course of global transaction processing, the caller must ensure that all of those connections/sessions associated with a specific global transaction branch identifier (XID) are established to the same resource manager.

## Data Structures

The DBMS\_XA package uses the following OBJECT type and associated TABLE type.

### OBJECT Types

- [DBMS\\_XA\\_XID Object Type](#)

### TABLE Types

- [DBMS\\_XA\\_XID\\_ARRAY Table Type](#)

## DBMS\_XA\_XID Object Type

The PL/SQL XA interface allows the PL/SQL application to define a global transaction branch id (XID) and associate/disassociate the current session with the transaction branch. XID is defined as a PL/SQL object type.

---



---

**See Also:** For more information, see "Distributed Transaction Processing: The XA Specification" in the public XA Standard.

---



---

### Syntax

```
TYPE DBMS_XA_XID IS OBJECT(
  formatid      NUMBER,
  gtrid         RAW(64),
  bqual         RAW(64),
  constructor function DBMS_XA_XID(
    gtrid       IN NUMBER)
    RETURN SELF AS RESULT,
  constructor function DBMS_XA_XID (
    gtrid       IN RAW,
    bqual       IN RAW)
    RETURN SELF AS RESULT,
  constructor function DBMS_XA_XID(
    formatid    IN NUMBER,
    gtrid       IN RAW,
    bqual       IN RAW DEFAULT HEXTORAW('00000000000000000000000000000001'))
    RETURN SELF AS RESULT)
```

### Attributes

**Table 162–3 DBMS\_XA\_XID Object Type**

Attribute	Description
formatid	Format identifier, a number identifying different transaction managers (TM)
gtrid	Global transaction identifier uniquely identifying a global transaction, of which the maximum size is 64 bytes
bqual	Branch qualifier, of which the maximum size is 64 bytes

## DBMS\_XA\_XID\_ARRAY Table Type

This type is used to define an array of `xid` that represent a list of global transaction branches.

### Syntax

```
TYPE DBMS_XA_XID_ARRAY as TABLE of DBMS_XA_XID
```

---

## Summary of DBMS\_XA Subprograms

**Table 162–4 DBMS\_Xa Package Subprograms**

Subprogram	Description
<a href="#">DIST_TXN_SYNC Procedure</a> on page 162-12	Used in recovery of synchronization when utilizing Oracle Real Application Clusters (Oracle RAC)
<a href="#">XA_COMMIT Function</a> on page 162-13	Commits the global transaction specified by <code>xid</code>
<a href="#">XA_END Function</a> on page 162-14	Disassociates the current session from the transaction branch specified by <code>xid</code>
<a href="#">XA_FORGET Function</a> on page 162-15	Informs the resource manager to forget about a heuristically committed or rolled back transaction branch.
<a href="#">XA_GETLASTOER Function</a> on page 162-16	Obtains the last Oracle error code, in case of failure of previous XA calls.
<a href="#">XA_PREPARE Function</a> on page 162-17	Prepares the transaction branch specified in <code>xid</code> for committing the transaction subsequently if possible
<a href="#">XA_RECOVER Function</a> on page 162-18	Obtains a list of prepared or heuristically completed transaction branches from a resource manager
<a href="#">XA_ROLLBACK Function</a> on page 162-19	Informs the resource manager to roll back work done on behalf of a transaction branch
<a href="#">XA_SETTIMEOUT Function</a> on page 162-20	Sets the transaction timeout in seconds for the current session
<a href="#">XA_START Function</a> on page 162-21	Associates the current session with the transaction branch specified by <code>xid</code>

## **DIST\_TXN\_SYNC Procedure**

This procedure can be used to synchronize in-doubt transactions when one of the Oracle Real Application Clusters (Oracle RAC) instances fails.

### **Syntax**

```
DBMS_XA.DIST_TXN_SYNC;
```

## XA\_COMMIT Function

This function commits the global transaction specified by `xid`.

### Syntax

```
DBMS_XA.XA_COMMIT (
  xid          IN DBMS_XA_XID,
  onePhase     IN BOOLEAN)
RETURN PLS_INTEGER;
```

### Parameters

**Table 162-5 XA\_COMMIT Function Parameters**

Parameter	Description
<code>xid</code>	See <a href="#">DBMS_XA_XID Object Type</a> on page 162-9
<code>onePhase</code>	If <code>TRUE</code> , apply single phase commit

### Return Values

See [Table 162-2, "DBMS\\_XA Constants for Possible Return Values"](#). Possible return values indicating error are: `XAER_RMERR`, `XAER_RMFAIL`, `XAER_NOTA`, `XAER_INVAL`, or `XAER_PROTO`. Other possible return values include: `XA_OK`, `XA_RB*`, `XA_HEURHAZ`, `XA_HEURCOM`, `XA_HEURRB`, and `XA_HEURMIX`.

### Usage Notes

- An application must not call `COMMIT`, but instead must call `XA_COMMIT` to commit the global transaction specified by `xid`. If a user needs to commit a transaction branch that is created by other users, `FORCE ANY TRANSACTION` must be granted to the user.
- If `onePhase` is `TRUE`, the resource manager should use a one-phase commit protocol to commit the work done on behalf of `xid`. Otherwise, only if all branches of the global transaction have been prepared successfully and the preceding `XA_PREPARE` call has returned `XA_OK`, should `XA_COMMIT` be called.
- The application must make a separate `XA_COMMIT` call for each of the transaction branches of the global transaction for which `XA_PREPARE` has returned `XA_OK`.
- If the resource manager did not commit the transaction and the parameter `onePhase` is set to `TRUE`, the resource manager may return one of the `XA_RB*` code. Upon return, the resource manager has rolled back the branch's work and has released all held resources.

## XA\_END Function

This function disassociates the current session from the transaction branch specified by `xid`.

A transaction manager calls `XA_END` when a thread of control finishes, or needs to suspend work on, a transaction branch. This occurs when the application completes a portion of its work, either partially or in its entirety (for example, before blocking on some event in order to let other threads of control work on the branch). When `XA_END` successfully returns, the calling thread of control is no longer actively associated with the branch but the branch still exists.

### Syntax

```
DBMS_XA.XA_END (
    xid    IN  DBMS_XA_XID,
    flag   IN  PLS_INTEGER)
RETURN PLS_INTEGER;
```

### Parameters

**Table 162–6 XA\_END Function Parameters**

Parameter	Description
<code>xid</code>	See <a href="#">DBMS_XA_XID Object Type</a> on page 162-9
<code>flag</code>	See <a href="#">Table 162–1, "DBMS_XA Constants for Flag Field of XA_START &amp; XA_END Functions"</a> .

### Return Values

See [Table 162–2, "DBMS\\_XA Constants for Possible Return Values"](#). Possible return values in error are `XAER_RMERR`, `XAER_RMFAILED`, `XAER_NOTA`, `XAER_INVAL`, `XAER_PROTO`, or `XA_RB*`.

### Usage Notes

- `TMSUCCESS` or `TMSUSPEND` may be specified in `flag`, and the transaction branch is disassociated with the current session in detached state if the return value is `XA_OK`. `TMFAIL` is not supported. `XA_END` may be called with either `TMSUCCESS` or `TMSUSPEND` to disassociate the transaction branch identified by `xid` from the current session.
- `XA_OK` is returned if `XA_END` succeeds. An application must check the return value and handle error cases. Only when `XA_OK` is returned, the application should proceed for other normal operations.
- Executing a `ROLLBACK` statement without calling `XA_END` first will rollback the changes made by the current transaction. However, the transaction context is still associated with the current session until `XA_END` is called.
- Executing a `COMMIT` statement without calling `XA_END` first will result in `ORA-02089: COMMIT is not allowed in a subordinate session`.
- Executing a `COMMIT` or a `ROLLBACK` statement after `XA_END` has no effect on the transaction identified by `xid`, since this transaction is no longer associated with the current session. To commit the transaction ID or the [XA\\_ROLLBACK Function](#) to commit/rollback the transaction specified by the `xid`.



## XA\_FORGET Function

This function informs the resource manager to forget about a heuristically committed or rolled back transaction branch.

### Syntax

```
DBMS_XA.XA_FORGET (  
    xid          IN DBMS_XA_XID)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 162-7 XA\_FORGET Function Parameters**

Parameter	Description
xid	See <a href="#">DBMS_XA_XID Object Type</a> on page 162-9

### Return Values

See [Table 162-2, "DBMS\\_XA Constants for Possible Return Values"](#). Possible return values are XA\_OK, XAER\_RMERR, XAER\_RMFAIL, XAER\_NOTA, XAER\_INVAL, or XAER\_PROTO.

## XA\_GETLASTOER Function

This function obtains the last Oracle error code, in case of failure of previous XA calls.

### Syntax

```
DBMS_XA.XA_GETLASTOER  
RETURN PLS_INTEGER;
```

### Return Values

The return value carries the last Oracle error code.

## XA\_PREPARE Function

This function prepares the transaction branch specified in `xid` for committing the transaction subsequently if possible.

### Syntax

```
DBMS_XA.XA_PREPARE (
    xid IN DBMS_XA_XID)
RETURN PLS_INTEGER;
```

### Parameters

**Table 162–8 XA\_PREPARE Function Parameters**

Parameter	Description
<code>xid</code>	See <a href="#">DBMS_XA_XID Object Type</a> on page 162-9

### Return Values

See [Table 162–2, "DBMS\\_XA Constants for Possible Return Values"](#). Possible return codes include: `XA_OK`, `XA_RDONLY`, `XA_RB*`, `XAER_RMERR`, `XAER_RMFAIL`, `XAER_NOTA`, `XAER_INVALID`, or `XAER_PROTO`.

### Usage Notes

- If a user needs to prepare a transaction branch that is created by other users, `FORCE ANY TRANSACTION` must be granted to the user.
- An application must keep track of all the branches of one global transaction, and prepare each transaction branch. Only if all branches of the global transaction have been prepared successfully and `XA_PREPARE` has returned `XA_OK`, the application may proceed to call `XA_COMMIT`.

## XA\_RECOVER Function

This function obtains a list of prepared or heuristically completed transaction branches from a resource manager.

### Syntax

```
DBMS_XA.XA_RECOVER  
RETURN DBMS_XA_XID_ARRAY;
```

### Return Values

See [DBMS\\_XA\\_XID\\_ARRAY Table Type](#)

### Usage Notes

- The flags `TMSTARTSCAN`, `TMENDSCAN`, `TMNOFLAGS` are not supported.
- The privilege `SELECT ON DBA_PENDING_TRANSACTIONS` must be granted to the user who needs to call `XA_RECOVER`.

## XA\_ROLLBACK Function

This function informs the resource manager to roll back work done on behalf of a transaction branch.

### Syntax

```
DBMS_XA.XA_ROLLBACK (
    xid          IN  DBMS_XA_XID)
RETURN PLS_INTEGER;
```

### Parameters

**Table 162–9 XA\_ROLLBACK Function Parameters**

Parameter	Description
xid	See <a href="#">DBMS_XA_XID Object Type</a> on page 162-9

### Return Values

See [Table 162–2, "DBMS\\_XA Constants for Possible Return Values"](#). Possible return values are: XA\_OK, XA\_RB\*, XA\_HEURHAZ, XA\_HEURCOM, XA\_HEURRB, or XA\_HEURMIX.

### Usage Notes

If a user needs to rollback a transaction branch that created by other users, the privilege FORCE ANY TRANSACTION must be granted to the user.

## XA\_SETTIMEOUT Function

This function sets the transaction timeout in seconds for the current session.

### Syntax

```
DBMS_XA.XA_SETTIMEOUT (  
    seconds IN PLS_INTEGER)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 162–10 XA\_SETTIMEOUT Function Parameters**

Parameter	Description
seconds	The timeout value indicates the maximum time in seconds that a transaction branch may be disassociated from the session before the system automatically aborts the transaction. The default value is 60 seconds.

### Return Values

See [Table 162–2, "DBMS\\_XA Constants for Possible Return Values"](#). Possible return values are XA\_OK, XAER\_RMERR, XAER\_RMFAIL, or XAER\_INVALID.

### Usage Notes

Only if return value is XA\_OK, is the timeout value successfully set.

## XA\_START Function

This function associates the current session with a transaction branch specified by the `xid`.

### Syntax

```
DBMS_XA.XA_START (
  xid IN DBMS_XA_XID,
  flag IN PLS_INTEGER)
RETURN PLS_INTEGER;
```

### Parameters

**Table 162–11 XA\_START Function Parameters**

Parameter	Description
<code>xid</code>	See <a href="#">DBMS_XA_XID Object Type</a> on page 162-9
<code>flag</code>	See <a href="#">Table 162–1, "DBMS_XA Constants for Flag Field of XA_START &amp; XA_END Functions"</a> .

### Return Values

See [Table 162–2, "DBMS\\_XA Constants for Possible Return Values"](#)

### Usage Notes

- If `TMJOIN` or `TMRESUME` is specified in `flag`, the start is for joining an existing transaction branch identified by the `xid`. `TMJOIN` flag should be used when the transaction is detached with `TMSUCCESS` flag. `TMRESUME` should be used when the transaction branch is detached with `TMSUSPEND` flag. `XA_START` may be called with either flag to join an existing transaction branch.
- If `TMNOFLAGS` is specified in `flag`, and neither `TMJOIN` nor `TMRESUME` is specified, a new transaction branch is to be started. If the transaction branch specified in `xid` already exists, `XA_START` returns an `XAER_DUPID` error code.
- Possible return values in error include: `XAER_RMERR`, `XAER_RMFAIL`, `XAER_DUPID`, `XAER_OUTSIDE`, `XAER_NOTA`, `XAER_INVALID`, and `XAER_PROTO`.
- `XA_OK` is returned if `XA_START` succeeds. An application must check the return value and handle error cases. Only when `XA_OK` is returned, the PL/SQL application should proceed for other normal operations.
- Transaction stacking is not supported. If there is an active transaction associated with the current session, may not be called to start or join another transaction. `XAER_PROTO` will be returned if `XA_START` is called with an active global transaction branch associated with the session. `XAER_OUTSIDE` will be returned if `XA_START` is called with a local transaction associated with the current session.





The DBMS\_XDB package supports the following features:

- Resource Management subprograms which complement Resource Views
- The Access Control List (ACL)-based Security Mechanism
- Configuration Session Management
- Creation of the XDB username

**See Also:**

- *Oracle XML DB Developer's Guide*
- *Oracle Database New Features Guide*

This chapter contains the following topics:

- [Using DBMS\\_XDB](#)
  - Overview
  - Deprecated Subprograms
  - Security Model
  - Constants
- [Summary of DBMS\\_XDB Subprograms](#)

## Using DBMS\_XDB

---

This section contains topics which relate to using the DBMS\_XDB package.

- [Overview](#)
- [Deprecated Subprograms](#)
- [Security Model](#)
- [Constants](#)

## Overview

The DBMS\_XDB package supports the following features:

- The Resource Management functionality provides [LINK Procedure](#), [EXISTSRESOURCE Function](#), [LOCKRESOURCE Function](#), [GETLOCKTOKEN Procedure](#), [UNLOCKRESOURCE Function](#), [CREATERESOURCE Functions](#), [RENAMERESOURCE Procedure](#), [DELETERESOURCE Procedure](#), [GETRESOID Function](#), [CREATEOIDPATH Function](#), [REBUILDHIERARCHICALINDEX Procedure](#) and [CREATEFOLDER Function](#) subprograms which complement Resource Views.
- The Access Control List (ACL)-based Security Mechanism can be used with in-hierarchy ACLs stored by the database or in-memory ACLs that may be stored outside the database. Some of these methods can be used for both Oracle resources and arbitrary database objects. Use [CHECKPRIVILEGES Function](#), [GETACLDOCUMENT Function](#), [CHANGEPRIVILEGES Function](#) and [GETCHILDRESPATHS Function](#) for Oracle Resources. [ACLCHECKPRIVILEGES Function](#) provides access to Oracle's ACL-based Security mechanism without storing objects in the Hierarchy.
- Configuration Session Management is supported by [CFG\\_REFRESH Procedure](#), [CFG\\_GET Function](#) and [CFG\\_UPDATE Procedure](#). methods.
- The XDB username is created during XDB installation. This user owns a set of default tables and packages. [GETXDB\\_TABLESPACE Function](#) and [MOVEXDB\\_TABLESPACE Procedure](#) enable movement of schemas to a specified tablespace, and support the default SYSAUX tablespace introduction

## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only and may be terminated in future releases.

---

---

The following subprograms are deprecated with Oracle Database 11g:

- [MOVEXDB\\_TABLESPACE Procedure](#)
- [REBUILDHIERARCHICALINDEX Procedure](#)

This functionality is replaced by the subprograms of the same name in the [DBMS\\_XDB\\_ADMIN](#) package:

- [MOVEXDB\\_TABLESPACE Procedure](#)
- [REBUILDHIERARCHICALINDEX Procedure](#)

## Security Model

Owned by XDB, the DBMS\_XDB package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC . Subprograms in this package are executed using the privileges of the current user. Subprograms that operate on the XDB Configuration will succeed only if the current user is SYS or XDB, or the current user has the XDBADMIN or DBA role.

## Constants

**Table 163–1** *Defined Constants for DBMS\_XDB*

<b>Constant</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
DELETE_RESOURCE	NUMBER	1	Deletes a resource; fails if the resource has children.
DELETE_RECURSIVE	NUMBER	2	Deletes a resource and its children, if any.
DELETE_FORCE	NUMBER	3	Deletes the resource, even if the object it contains is invalid.
DELETE_RECURSIVE_FORCE	NUMBER	4	Deletes a resource and its children, if any, even if the object it contains is invalid.

## Summary of DBMS\_XDB Subprograms

**Table 163–2 DBMS\_XDB Package Subprograms**

Subprogram	Description
<a href="#">ACLCHECKPRIVILEGES Function</a> on page 163-10	Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter.
<a href="#">ADDHTTPEXPIREMAPPING Procedure</a> on page 163-11	Adds to <code>xdb\$config</code> a mapping of the URL pattern to an expiration date. This will control the Expire headers for URLs matching the pattern.
<a href="#">ADDMIMEMAPPING Procedure</a> on page 163-12	Adds a mime mapping to the XDB configuration
<a href="#">ADDSCHEMALOCMAPPING Procedure</a> on page 163-13	Adds a schema location mapping to the XDB configuration
<a href="#">ADDSERVLET Procedure</a> on page 163-14	Adds a servlet to XDB configuration
<a href="#">ADDSERVLETMAPPING Procedure</a> on page 163-15	Adds a servlet mapping to XDB configuration
<a href="#">ADDSERVLETSECROLE Procedure</a> on page 163-16	Adds a security role REF to a specified servlet in the XDB configuration
<a href="#">ADDXMLEXTENSION Procedure</a> on page 163-17	Adds adds an XML extension to the XDB configuration
<a href="#">APPENDPATH Procedure</a> on page 163-18	Appends a childpath to a parentpath
<a href="#">APPENDPATH Procedure</a> on page 163-18	Takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource
<a href="#">CFG_GET Function</a> on page 163-20	Retrieves the session's configuration information
<a href="#">CFG_REFRESH Procedure</a> on page 163-21	Refreshes the session's configuration information to the latest configuration
<a href="#">CFG_UPDATE Procedure</a> on page 163-22	Updates the configuration information
<a href="#">CHANGEOWNER Procedure</a> on page 163-23	Changes the owner of the resource/s to the specified owner.
<a href="#">CHANGEPRIVILEGES Function</a> on page 163-24	Adds a specified ACE to a specified resource's ACL
<a href="#">CHECKPRIVILEGES Function</a> on page 163-25	Checks access privileges granted to the current user on the specified resource
<a href="#">CREATEFOLDER Function</a> on page 163-26	Creates a new folder resource in the hierarchy
<a href="#">CREATEOIDPATH Function</a> on page 163-27	Creates a virtual path to the resource based on object ID
<a href="#">CREATERESOURCE Functions</a> on page 163-28	Creates a new resource
<a href="#">DELETEHTTPEXPIREMAPPING Procedure</a> on page 163-30	Deletes from <code>xdb\$config</code> all mappings of the URL pattern to an expiration date
<a href="#">DELETEMIMEMAPPING Procedure</a> on page 163-31	Deletes the mime mapping from the XDB configuration

**Table 163–2 (Cont.) DBMS\_XDB Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DELETERESOURCE Procedure</a> on page 163-32	Deletes a resource from the hierarchy
<a href="#">DELETERESOURCEMETADATA Procedures</a> on page 163-33	Deletes metadata from a resource (can be used for schema-based or nonschema-based metadata)
<a href="#">DELETESCHEMALOCMAPPING Procedure</a> on page 163-34	Deletes the schema location mapping for the specified schema URL from the XDB configuration.
<a href="#">DELETESERVLET Procedure</a> on page 163-35	Deletes a servlet from XDB configuration
<a href="#">DELETESERVLETMAPPING Procedure</a> on page 163-36	Deletes the servlet mapping for the specified servlet name from the XDB configuration
<a href="#">DELETESERVLETSECROLE Procedure</a> on page 163-37	Deletes the specified role from a servlet in the XDB configuration
<a href="#">DELETXMLEXTENSION Procedure</a> on page 163-38	Deletes the specified XML extension from the XDB configuration
<a href="#">EXISTSRESOURCE Function</a> on page 163-39	Determines if a resource is the hierarchy, based on its absolute path
<a href="#">GETACLDOCUMENT Function</a> on page 163-40	Retrieves ACL document that protects resource given its path name
<a href="#">GETCHILDRESPATHS Function</a> on page 163-41	Returns a cursor over the absolute paths of all the child resources
<a href="#">GETCONTENTBLOB Function</a> on page 163-42	Retrieves the contents of a resource returned as a BLOB
<a href="#">GETCONTENTCLOB Function</a> on page 163-43	Retrieves the contents of a resource returned as a CLOB
<a href="#">GETCONTENTVARCHAR2 Function</a> on page 163-44	Retrieves the contents of a resource returned as a string
<a href="#">GETCONTENTXMLREF Function</a> on page 163-45	Retrieves the contents of a resource returned as a REF to an XMLTYPE
<a href="#">GETCONTENTXMLTYPE Function</a> on page 163-46	Retrieves the contents of a resource returned as an XMLTYPE
<a href="#">GETFTPSPORT Function</a> on page 163-47	Gets the value of the current FTP port
<a href="#">GETHTTPSPORT Function</a> on page 163-48	Gets the value of the current HTTP port
<a href="#">GETLOCKTOKEN Procedure</a> on page 163-50	Returns that resource's lock token for the current user given a path to a resource
<a href="#">GETLISTENERENDPOINT Procedure</a> on page 163-49	Retrieves the parameters of a listener end point corresponding to the XML DB HTTP server
<a href="#">GETRESOID Function</a> on page 163-52	Returns the object ID of the resource from its absolute path
<a href="#">GETXDB_TABLESPACE Function</a> on page 163-53	Returns the current tablespace of the XDB (user)
<a href="#">HASBLOBCONTENT Function</a> on page 163-54	Returns TRUE if the resource has BLOB content
<a href="#">HASCHARCONTENT Function</a> on page 163-55	Returns TRUE if the resource has character content



**Table 163–2 (Cont.) DBMS\_XDB Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">HASXMLCONTENT Function</a> on page 163-56	Returns TRUE if the resource has XML content
<a href="#">HASXMLREFERENCE Function</a> on page 163-57	Returns TRUE if the resource has REF to XML content
<a href="#">ISFOLDER Function</a> on page 163-58	Returns TRUE if the resource is a folder or container
<a href="#">LINK Procedures</a> on page 163-59	Creates a link to an existing resource
<a href="#">LOCKRESOURCE Function</a> on page 163-60	Gets a WebDAV-style lock on that resource given a path to that resource
<a href="#">MOVEXDB_TABLESPACE Procedure</a> on page 163-61	[Deprecated] Moves the XDB (user) to the specified tablespace
<a href="#">PROCESSLINKS Procedure</a> on page 163-62	Processes document links in the specified resource
<a href="#">PURGERESOURCEMETADATA Procedure</a> on page 163-63	Deletes all user metadata from a resource
<a href="#">REBUILDHIERARCHICALINDEX Procedure</a> on page 163-64	[Deprecated] Rebuilds the hierarchical index after import or export operations
<a href="#">RENAMERESOURCE Procedure</a> on page 163-65	Renames the XDB resource
<a href="#">SETACL Procedure</a> on page 163-66	Sets the ACL on a specified resource
<a href="#">SETCONTENT Procedures</a> on page 163-67	Replaces the contents of a specified resource with specified datatype
<a href="#">SETFTPSPORT Procedure</a> on page 163-69	Sets the FTP port to a new value
<a href="#">SETHHTTPSPORT Procedure</a> on page 163-70	Sets the HTTP port to a new value
<a href="#">SETLISTENERENDPOINT Procedure</a> on page 163-71	Sets the parameters of a listener end point corresponding to the XML DB HTTP server
<a href="#">SETLISTENERLOCALACCESS Procedure</a> on page 163-72	Restricts all listener end points of the XML DB HTTP server to listen either only on the localhost interface or on both localhost and non-localhost interfaces
<a href="#">SPLITPATH Procedure</a> on page 163-73	Splits the path into a parentpath and childpath
<a href="#">TOUCHRESOURCE Procedure</a> on page 163-74	Changes the modification time of the resource to the current time
<a href="#">UPDATERESOURCEMETADATA Procedures</a> on page 163-75	Updates metadata for a resource
<a href="#">UNLOCKRESOURCE Function</a> on page 163-77	Unlocks the resource given a lock token and resource path

## ACLCHECKPRIVILEGES Function

This function checks access privileges granted to the current user by specified ACL document by the OWNER of the resource. Returns positive integer if all privileges are granted.

### Syntax

```
DBMS_XDB.ACLCHECKPRIVILEGES(  
    acl_path IN VARCHAR2,  
    owner    IN VARCHAR2,  
    privs   IN xmltype)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 163–3** ACLCHECKPRIVILEGES Function Parameters

Parameter	Description
acl_path	Absolute path in the Hierarchy for ACL document
owner	Resource owner name; the pseudo user "DAV:owner" is replaced by this user during ACL privilege resolution
privs	An XMLType instance of the privilege element specifying the requested set of access privileges. See description for <a href="#">CHECKPRIVILEGES Function</a> .

## ADDHTTPEXPIREMAPPING Procedure

This procedure adds to `xdb$config` a mapping of the URL pattern to an expiration date. This will control the Expire headers for URLs matching the pattern.

### Syntax

```
DBMS_XDB.ADDHTTPEXPIREMAPPING (
    pattern    IN    VARCHAR2,
    expire     IN    VARCHAR2);
```

### Parameters

**Table 163–4** ADDHTTPEXPIREMAPPING Procedure Parameters

Parameter	Description
pattern	URL pattern (only * accepted as wildcards)
expire	Expiration directive, follows the ExpireDefault in Apache's mod_expires: base [plus] (num type)* -- base: now   modification -- type: year years month months week weeks day days minute minutess second seconds

### Examples

```
DBMS_XDB.ADDHTTPEXPIREMAPPING ('/public/test1/*', 'now plus 4 weeks');
DBMS_XDB.ADDHTTPEXPIREMAPPING ('/public/test2/*', 'modification plus 1 day 30 seconds');
```

## ADDMIMEMAPPING Procedure

This procedure adds the following mime mapping to XDB configuration:

```
<mime-mapping>
<extension>extension</extension>
<mime-type>mime-type</mime-type>
</mime-mapping>
```

### Syntax

```
DBMS_XDB.ADDMIMEMAPPING (
    extension    IN    VARCHAR2,
    mime-type    IN    VARCHAR2);
```

### Parameters

**Table 163–5** *ADDMIMEMAPPING Procedure Parameters*

Parameter	Description
extension	Extension for which a mime type is being added
mime-type	Mime type

## ADDSCHEMALOCMAPPING Procedure

This procedure adds the following schema location mapping to the XDB configuration:

```
<schemaLocation-mapping>
  <namespace>namespace</namespace>
  <element>element</element>
  <schemaURL>schemaURL</schemaURL>
</schemaLocation-mapping>
```

### Syntax

```
DBMS_XDB.ADDSCHEMALOCMAPPING (
  namespace    IN   VARCHAR2,
  element      IN   VARCHAR2,
  schemaURL    IN   VARCHAR2);
```

### Parameters

**Table 163–6** *ADDSCHEMALOCMAPPING Procedure Parameters*

Parameter	Description
namespace	Namespace
element	Element
schemaURL	Schema URL

## ADDSERVLET Procedure

This procedure adds the following servlet to XDB configuration:

```
<servlet>
  <servlet-name>name</servlet-name>
  <servlet-language>language</servlet-language>
  <display-name>dispname</display-name>
  <description>descript</description>
  <servlet-class>class</servlet-class>
  <servlet-schema>schema</servlet-schema>
</servlet>
```

### Syntax

```
DBMS_XDB.ADDSERVLET (
  name          IN   VARCHAR2,
  language      IN   VARCHAR2,
  dispname      IN   VARCHAR2,
  icon          IN   VARCHAR2 := NULL,
  descript      IN   VARCHAR2 := NULL,
  class         IN   VARCHAR2 := NULL,
  jspfile       IN   VARCHAR2 := NULL,
  plsqli        IN   VARCHAR2 := NULL);
```

### Parameters

**Table 163–7** *ADDSERVLET Procedure Parameters*

Parameter	Description
name	Servlet name
language	Must be one of "C", "Java", "PL/SQL"
dispname	Display name
icon	Icon
descript	Description
class	The class / jspfile / plsqli function corresponding to this servlet. The first non-NULL argument amongst these three is chosen, and the others are treated as NULL.
jspfile	The class / jspfile / plsqli function corresponding to this servlet. The first non-NULL argument amongst these three is chosen, and the others are treated as NULL.
plsqli	The class / jspfile / plsqli function corresponding to this servlet. The first non-NULL argument amongst these three is chosen, and the others are treated as NULL.
schema	Schema

## ADDSERVLETMAPPING Procedure

This procedure adds the following servlet mapping to XDB configuration:

```
<servlet-mapping>
  <servlet-pattern>pattern</servlet-pattern>
  <servlet-name>name</servlet-name>
</servlet-mapping>
```

### Syntax

```
DBMS_XDB.ADDSERVLETMAPPING (
  pattern  IN  VARCHAR2,
  name     IN  VARCHAR2);
```

### Parameters

**Table 163–8** *ADDSERVLETMAPPING Procedure Parameters*

Parameter	Description
pattern	Servlet pattern
name	Servlet name

## ADDSERVLETSECROLE Procedure

This procedure adds the following security role REF to a specified servlet in XDB configuration:

```
<security-role-ref>
  <role-name>rolename</role-name>
  <role-link>rolelink</role-link>
  <description>descript</description>
</security-role-ref>
```

### Syntax

```
DBMS_XDB.ADDSERVLETSECROLE (
  servname    IN   VARCHAR2,
  rolename    IN   VARCHAR2,
  rolelink    IN   VARCHAR2,
  descript    IN   VARCHAR2 := NULL);
```

### Parameters

**Table 163–9** *ADDSERVLETSECROLE Procedure Parameters*

Parameter	Description
servname	Sservlet name
rolename	Role name
rolelink	Role link
descript	Description



## ADDXML\_EXTENSION Procedure

This procedure adds the following XML extension to the XDB configuration under `<xml-extensions>`:

```
<extension>extension</extension>
```

### Syntax

```
DBMS_XDB.ADDXML_EXTENSION(  
    extension IN VARCHAR2);
```

### Parameters

**Table 163–10** ADDXML\_EXTENSION Procedure Parameters

Parameter	Description
extension	XML extension to be added

## APPENDPATH Procedure

This procedure appends a childpath to a parentpath.

### Syntax

```
DBMS_XDB.APPENDPATH (  
    abspath      OUT  VARCHAR2,  
    parentpath  IN   VARCHAR2,  
    childpath   IN   VARCHAR2);
```

### Parameters

**Table 163–11** APPENDPATH Procedure

Parameter	Description
abspath	Absolute path of the resource
parentpath	Parentpath
childpath	Childpath

## APPENDRESOURCEMETADATA Procedure

This procedure takes in user-defined metadata either as a REF to XMLTYPE or an XMLTYPE and adds it to the desired resource.

### Syntax

```
DBMS_XDB.APPENDRESOURCEMETADATA (
  abspath IN VARCHAR2,
  metadata IN XMLTYPE);
```

```
DBMS_XDB.APPENDRESOURCEMETADATA (
  abspath IN VARCHAR2,
  metadata IN REF SYS.XMLTYPE);
```

### Parameters

**Table 163–12** APPENDRESOURCEMETADATA Procedure

Parameter	Description
abspath	Absolute path of the resource
metadata	Metadata can be schema based or nonschema-based. Schema-based metadata is stored in its own table.

### Usage Notes

- In the case in which a REF is passed in, the procedure stores the REF in the resource, and the metadata is stored in a separate table. In this case you are responsible for populating the RESID column for the metadata table. Note that the REF passed in must be unique. In other words, there must not be a REF with the same value in the resource metadata, as this would violate uniqueness of properties. An error is thrown if users attempt to add a REF that already exists.
- In the case where the XMLTYPE is passed in, the data is parsed to determine if it is schema-based or not and stored accordingly.

## CFG\_GET Function

This function retrieves the session's configuration information as an `XMLType` instance.

### Syntax

```
DBMS_XDB.CFG_GET  
RETURN SYS.XMLType;
```

## CFG\_REFRESH Procedure

This procedure refreshes the session's configuration information to the latest configuration.

### Syntax

```
DBMS_XDB.CFG_REFRESH;
```

## CFG\_UPDATE Procedure

This procedure updates the configuration information and commits the change.

### Syntax

```
DBMS_XDB.CFG_UPDATE(  
    xdbconfig IN SYS.XMLTYPE);
```

### Parameters

**Table 163–13** *CFG\_UPDATE Procedure Parameters*

Parameter	Description
<code>xdbconfig</code>	The new configuration data

## CHANGEOWNER Procedure

This procedure changes the owner of the resource/s to the specified owner.

### Syntax

```
DBMS_XDB.CHANGEOWNER(  
    abspath    IN    VARCHAR2,  
    owner      IN    VARCHAR2,  
    recurse    IN    BOOLEAN := FALSE);
```

### Parameters

**Table 163–14** *CHANGEOWNER Procedure Parameters*

Parameter	Description
abspath	Absolute path of the resource
owner	New owner for the resource
recurse	If TRUE, recursively change owner of all resources in the folder tree

## CHANGEPRIVILEGES Function

This function adds a specified ACE to a specified resource's ACL.

### Syntax

```
DBMS_XDB.CHANGEPRIVILEGES (  
    res_path IN VARCHAR2,  
    ace      IN XMLType)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 163–15** *CHANGEPRIVILEGES Function Parameters*

Parameter	Description
res_path	Path name of the resource for which privileges need to be changed
ace	An XMLType instance of the <ace> element which specifies the <principal>, the operation <grant> and the list of privileges

### Return Values

A positive integer if the ACL was successfully modified.

### Usage Notes

If no ACE with the same principal and the same operation (grant/deny) already exists in the ACL, the new ACE is added at the end of the ACL.



## CHECKPRIVILEGES Function

This function checks access privileges granted to the current user on the specified resource.

### Syntax

```
DBMS_XDB.CHECKPRIVILEGES (  
    res_path  IN VARCHAR2,  
    privs     IN xmltype)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 163–16** CHECKPRIVILEGES Function Parameters

Parameter	Description
res_path	Absolute path in the Hierarchy for resource
privs	An XMLType instance of the privilege element specifying the requested set of access privileges

### Return Values

A positive integer if all requested privileges granted.

## CREATEFOLDER Function

This function creates a new folder resource in the hierarchy.

### Syntax

```
DBMS_XDB.CREATEFOLDER(  
    path IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 163–17 CREATEFOLDER Function Parameters**

Parameter	Description
path	Path name for the new folder

### Return Values

TRUE if operation successful; FALSE, otherwise.

### Usage Notes

The given path name's parent folder must already exist in the hierarchy: if '/folder1/folder2' is passed as the path parameter, then '/folder1' must already exist.

## CREATEOIDPATH Function

This function creates a virtual path to the resource based on object ID.

### Syntax

```
DBMS_XDB.CREATEOIDPATH(  
    oid IN RAW)  
RETURN VARCHAR2;
```

### Parameters

**Table 163–18** *CREATEOIDPATH Function Parameters*

Parameter	Description
oid	Object ID of the resource

## CREATERESOURCE Functions

The functions create a new resource. The description of the overload options precede each version of the syntax

### Syntax

Creates a new resource with a specified string as its contents:

```
DBMS_XDB.CREATERESOURCE (
    abspath          IN  VARCHAR2,
    data             IN  VARCHAR2,
    createfolders    IN  BOOLEAN := FALSE)
RETURN BOOLEAN;
```

Creates a new resource with a specified XMLType data as its contents:

```
DBMS_XDB.CREATERESOURCE (
    abspath          IN  VARCHAR2,
    data             IN  SYS.XMLTYPE,
    createfolders    IN  BOOLEAN := FALSE)
RETURN BOOLEAN;
```

Given a REF to an existing XMLType row, creates a resource whose contents point to that row. That row should not already exist inside another resource:

```
DBMS_XDB.CREATERESOURCE (
    abspath          IN  VARCHAR2,
    datarow          IN  REF SYS.XMLTYPE,
    createfolders    IN  BOOLEAN := FALSE)
RETURN BOOLEAN;
```

Creates a resource with a specified BLOB as its contents, and specifies character set of the source BLOB:

```
DBMS_XDB.CREATERESOURCE (
    abspath          IN  VARCHAR2,
    data             IN  BLOB,
    csid             IN  NUMBER :=0,
    createfolders    IN  BOOLEAN := FALSE)
RETURN BOOLEAN;
```

Creates a resource with a specified BFILE as its contents, and specifies character set of the source BFILE:

```
DBMS_XDB.CREATERESOURCE (
    abspath          IN  VARCHAR2,
    data             IN  BFILE,
    csid             IN  NUMBER :=0,
    createfolders    IN  BOOLEAN := FALSE)
RETURN BOOLEAN;
```

Creates a resource with a specified CLOB as its contents:

```
DBMS_XDB.CREATERESOURCE (
    abspath          IN  VARCHAR2,
    data             IN  CLOB,
    createfolders    IN  BOOLEAN := FALSE)
RETURN BOOLEAN;
```

Given a string, inserts a new resource into the hierarchy with the string as the contents:

```

DBMS_XDB.CREATERESOURCE (
  abspath      IN  VARCHAR2,
  data         IN  VARCHAR2,
  schemaurl    IN  VARCHAR2 := NULL,
  elem         IN  VARCHAR2 := NULL)
RETURN BOOLEAN;

```

Given an XMLTYPE and a schema URL, inserts a new resource into the hierarchy with the XMLTYPE as the contents:

```

DBMS_XDB.CREATERESOURCE (
  abspath      IN  VARCHAR2,
  data         IN  SYS.XMLTYPE,
  schemaurl    IN  VARCHAR2 := NULL,
  elem         IN  VARCHAR2 := NULL)
RETURN BOOLEAN;

```

## Parameters

**Table 163–19** CREATERESOURCE Function Parameters

Parameter	Description
abspath	Absolute path of the resource to create. The path name's parent folder must already exist in the hierarchy. In other words, if /foo/bar.txt is passed in, then folder /foo must already exist.
data	String buffer containing new resource's contents. The data is parsed to check if it contains a schema-based XML document, and the contents are stored as schema-based in the schema's default table. Otherwise, it is saved as binary data.
datarow	REF to an XMLType row to be used as the contents
csid	Character set id of the document. Must be a valid Oracle ID; otherwise returns an error.  If CSID is not specified, or if a zero CSID is specified, then the character set id of the document is determined as follows: <ul style="list-style-type: none"> <li>■ From the abspath extension, determine the resource's MIME type.</li> <li>■ If the MIME type is */xml, then the encoding is detected based on Appendix F of the W3C XML 1.0 Reference at <a href="http://www.w3.org/TR/2000/REC-xml-20001006">http://www.w3.org/TR/2000/REC-xml-20001006</a>;</li> <li>■ Otherwise, it is defaulted to the database character set.</li> </ul>
createfolders	If TRUE, create the parent folders if they do not exist
schemaurl	For XML data, schema URL data conforms to (default NULL)
elem	Element name (default NULL)

## Return Values

TRUE if operation successful; FALSE, otherwise.

## DELETEHTTPEXPIREMAPPING Procedure

This procedure deletes from `xdb$config` all mappings of the URL pattern to an expiration date.

### Syntax

```
DBMS_XDB.DELETEHTTPEXPIREMAPPING(  
    pattern IN VARCHAR2);
```

### Parameters

**Table 163–20** *DELETEHTTPEXPIREMAPPING Procedure Parameters*

Parameter	Description
<code>pattern</code>	URL pattern (only * accepted as wildcards)

## DELETMIMEMAPPING Procedure

This procedure deletes the mime mapping for a specified extension from the XDB configuration.

### Syntax

```
DBMS_XDB.DELETMIMEMAPPING(  
    extension IN VARCHAR2);
```

### Parameters

**Table 163–21** *DELETMIMEMAPPING Procedure Parameters*

Parameter	Description
extension	Extension for which a mime type is to be deleted

## DELETERESOURCE Procedure

This procedure deletes a resource from the hierarchy.

### Syntax

```
DBMS_XDB.DELETERESOURCE(  
  path          IN          VARCHAR2,  
  delete_option IN          PLS_INTEGER);
```

### Parameters

**Table 163–22 DELETERESOURCE Procedure Parameters**

Parameter	Description
path	Path name of the resource to delete
delete_option	The option that controls how a resource is deleted; defined in <a href="#">Table 163–1</a> on page 163-6: <ul style="list-style-type: none"><li>▪ DELETE_RESOURCE</li><li>▪ DELETE_RECURSIVE</li><li>▪ DELETE_FORCE</li><li>▪ DELETE_RECURSIVE_FORCE</li></ul>



## DELETERESOURCEMETADATA Procedures

This procedure takes in a resource by absolute path and removes either the schema-based metadata identified by the REF, or the metadata identified by the namespace and name combination, which can be either schema-based or non-schema based. It also takes an additional (optional) parameter that specifies how to delete it. This parameter is only relevant for schema-based resource metadata that needs to be deleted. For non-schema based metadata, this parameter is ignored.

### Syntax

Can be used only for schema-based metadata:

```
DBMS_XDB.DELETERESOURCEMETADATA (
    abspath          IN VARCHAR2,
    metadata         IN REF SYS.XMLTYPE,
    delete_option   IN pls_integer := dbms_xdb.DELETE_RESOURCE_METADATA_CASCADE);
```

Can be used for schema-based or nonschema-based metadata:

```
DBMS_XDB.DELETERESOURCEMETADATA (
    abspath          IN VARCHAR2,
    metadatans       IN VARCHAR2,
    metadataname     IN VARCHAR2,
    delete_option   IN pls_integer := dbms_xdb.DELETE_RESOURCE_METADATA_CASCADE);
```

### Parameters

**Table 163–23 DELETERESOURCEMETADATA Procedure Parameters**

Parameter	Description
abspath	Absolute path of the resource
metadata	REF to the piece of metadata (schema based) to be deleted
metadatans	Namespace of the metadata fragment to be removed
metadataname	Local name of the metadata fragment to be removed
delete_option	Only applicable for schema-based metadata, this can be one of the following: <ul style="list-style-type: none"> <li>▪ DELETE_RES_METADATA_CASCADE - deletes the corresponding row in the metadata table</li> <li>▪ DELETE_RES_METADATA_NOCASCADE - does not delete the row in the metadata table</li> </ul>

## DELETESCHEMALOCMAPPING Procedure

This procedure deletes the schema location mapping for a specified schema URL from the XDB configuration.

### Syntax

```
DBMS_XDB.DELETESCHEMALOCMAPPING(  
    schemaURL IN VARCHAR2);
```

### Parameters

**Table 163–24 DELETESCHEMALOCMAPPING Procedure Parameters**

Parameter	Description
schemaURL	Schema URL

## DELETESERVLET Procedure

This procedure deletes a servlet from the XDB configuration.

### Syntax

```
DBMS_XDB.DELETESERVLET(  
    name          IN  VARCHAR2);
```

### Parameters

**Table 163–25** *DELETESERVLET Procedure Parameters*

Parameter	Description
name	Servlet name

## DELETESERVLETMAPPING Procedure

This procedure deletes the servlet mapping for a specified servlet name from the XDB configuration.

### Syntax

```
DBMS_XDB.DELETESERVLETMAPPING(  
    name      IN  VARCHAR2);
```

### Parameters

**Table 163–26** *DELETESERVLETMAPPING Procedure Parameters*

Parameter	Description
name	Servlet name

## DELETESERVLETSECROLE Procedure

This procedure deletes the specified role from a servlet in the XDB configuration.

### Syntax

```
DBMS_XDB.DELETESERVLETSECROLE(  
    servname    IN    VARCHAR2,  
    rolename    IN    VARCHAR2);
```

### Parameters

**Table 163–27** *DELETESERVLETSECROLE Procedure Parameters*

Parameter	Description
servname	Servlet name
rolename	Name of the role to be deleted

## DELETXMLEXTENSION Procedure

This procedure deletes the specified XML extension from the XDB configuration.

### Syntax

```
DBMS_XMLEXTENSION.DELETEXMLEXTENSION(  
    extension IN VARCHAR2);
```

### Parameters

**Table 163–28 DELETXMLEXTENSION Procedure Parameters**

Parameter	Description
extension	XML extension to be deleted

## EXISTSRESOURCE Function

This function indicates if a resource is in the hierarchy. Matches resource by a string that represents its absolute path.

### Syntax

```
DBMS_XDB.EXISTSRESOURCE(  
    abspath IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 163–29** *EXISTSRESOURCE Function Parameters*

Parameter	Description
abspath	Path name of the resource whose ACL document is required

### Return Values

TRUE if the resource is found.

## GETACLDOCUMENT Function

This function retrieves ACL document that protects resource given its path name.

### Syntax

```
DBMS_XDB.GETACLDOCUMENT(  
  abspath IN VARCHAR2)  
RETURN sys.xmltype;
```

### Parameters

**Table 163–30** *GETACLDOCUMENT Function Parameters*

Parameter	Description
abspath	Path name of the resource whose ACL document is required

### Return Values

The XMLType for ACL document.



## GETCHILDRESPATHS Function

This function returns a cursor over the absolute paths of all the child resources.

### Syntax

```
DBMS_XDB.GETCHILDRESPATHS (  
    abspath    IN    VARCHAR2);  
RETURN SYS_REFCURSOR;
```

### Parameters

**Table 163–31** *GETCHILDRESPATHS Function Parameters*

Parameter	Description
abspath	Absolute path of the resource

### Return Values

A cursor over the absolute paths of all the child resources.

## GETCONTENTBLOB Function

This function retrieves the contents of a resource returned as a BLOB.

### Syntax

```
DBMS_XDB.GETCONTENTBLOB(  
    abspath    IN    VARCHAR2,  
    csid       OUT   PLS_INTEGER,  
    locksrc    IN    BOOLEAN := FALSE)  
RETURN BLOB;
```

### Parameters

**Table 163–32** *GETCONTENTBLOB Function Parameters*

Parameter	Description
abspath	Absolute path of the resource
csid	If TRUE, lock and return the source LOB. If FALSE, return a temp LOB copy.
locksrc	Contents of the resource as a BLOB

### Return Values

The contents of the resource as a BLOB.

## GETCONTENTCLOB Function

This function gets the contents of a resource returned as a CLOB.

### Syntax

```
DBMS_XDB.GETCONTENTCLOB(  
    abspath    IN    VARCHAR2,  
    RETURN CLOB;
```

### Parameters

**Table 163–33** *GETCONTENTCLOB Function Parameters*

Parameter	Description
abspath	Absolute path of the resource

### Return Values

The contents of the resource as a CLOB.

## GETCONTENTVARCHAR2 Function

This function gets the contents of a resource returned as a string.

### Syntax

```
DBMS_XDB.GETCONTENTVARCHAR2 (  
    abspath    IN    VARCHAR2,  
    RETURN BLOB;
```

### Parameters

**Table 163–34** *GETCONTENTVARCHAR2 Function Parameters*

Parameter	Description
abspath	Absolute path of the resource

### Return Values

The contents of the resource as a string.

## GETCONTENTXMLREF Function

This function retrieves the contents of a resource returned as a REF to an XMLTYPE.

### Syntax

```
DBMS_XDB.GETCONTENTXMLREF (  
    abspath    IN    VARCHAR2,  
    RETURN SYS.XMLTYPE;
```

### Parameters

**Table 163–35** *GETCONTENTXMLREF Function Parameters*

Parameter	Description
abspath	Absolute path of the resource

### Return Values

The contents of the resource as a REF to an XMLTYPE.

## GETCONTENTXMLTYPE Function

This function retrieves the contents of a resource returned as an `XMLTYPE`.

### Syntax

```
DBMS_XDB.GETCONTENTXMLTYPE(  
    abspath IN VARCHAR2,  
    RETURN SYS.XMLTYPE;
```

### Parameters

**Table 163–36** *GETCONTENTXMLTYPE Function Parameters*

Parameter	Description
<code>abspath</code>	Absolute path of the resource

### Return Values

The contents of the resource as an `XMLTYPE`.

## GETFTPPORT Function

This procedure gets the value of the current FTP port.

### Syntax

```
DBMS_XDB.GETFTPPORT  
RETURN NUMBER;
```

## GETHTTPPORT Function

This procedure gets the value of the current HTTP port.

### Syntax

```
DBMS_XDB.GETHTTPPORT  
RETURN NUMBER;
```



## GETLISTENERENDPOINT Procedure

This procedure retrieves the parameters of a listener end point corresponding to the XML DB HTTP server. The parameters of both HTTP and HTTP2 end points can be retrieved by invoking this procedure.

### Syntax

```
DBMS_XDB.GETLISTENERENDPOINT (
  endpoint IN NUMBER,
  host     OUT VARCHAR2,
  port    OUT NUMBER,
  protocol OUT NUMBER);
```

### Parameters

**Table 163–37** *GETLISTENERENDPOINT Procedure Parameters*

Parameter	Description
endpoint	End point to be retrieved. Its value can be XDB_ENDPOINT_HTTP or XDB_ENDPOINT_HTTP2.
host	Interface on which the listener end point listens
port	Port on which the listener end point listens
protocol	Transport protocol accepted by the listener end point

## GETLOCKTOKEN Procedure

Given a path to a resource, this procedure returns that resource's lock token for the current user.

### Syntax

```
DBMS_XDB.GETLOCKTOKEN(  
  path          IN      VARCHAR2,  
  locktoken     OUT     VARCHAR2);
```

### Parameters

**Table 163–38** *GETLOCKTOKEN Procedure Parameters*

Parameter	Description
path	Path name to the resource
locktoken	Logged-in user's lock token for the resource

### Usage Notes

The user must have READPROPERTIES privilege on the resource.

## GETPRIVILEGES Function

This function gets all privileges granted to the current user on a specified resource.

### Syntax

```
DBMS_XDB.GETPRIVILEGES (  
    res_path    IN    VARCHAR2)  
RETURN sys.xmltype;
```

### Parameters

**Table 163–39** *GETPRIVILEGES Function Parameters*

Parameter	Description
res_path	Absolute path in the hierarchy of the resource

### Return Values

An XMLType instance of <privilege> element, which contains the list of all leaf privileges granted on this resource to the current user.

## GETRESOID Function

Returns the object ID of the resource from its absolute path.

### Syntax

```
DBMS_XDB.GETRESOID(  
    abspath IN VARCHAR2)  
RETURN RAW;
```

### Parameters

**Table 163–40** *GETRESOID Function Parameters*

Parameter	Description
abspath_path	Absolute path of the resource

### Return Values

NULL if the resource is not present.

## GETXDB\_TABLESPACE Function

This function returns the current tablespace of the XDB (user).

### Syntax

```
DBMS_XDB.GETXDB_TABLESPACE  
RETURN VARCHAR2;
```

## HASBLOBCONTENT Function

This function returns TRUE if the resource has BLOB content.

### Syntax

```
DBMS_XDB.DBMS_XDB.HASBLOBCONTENT  
  (abspath IN VARCHAR2)  
  RETURN BOOLEAN;
```

### Parameters

**Table 163–41 HASBLOBCONTENT Function Parameters**

Parameter	Description
abspath_path	Absolute path of the resource

### Return Values

TRUE if the resource has BOB content.

## HASCHARCONTENT Function

This function returns TRUE if the resource has character content.

### Syntax

```
DBMS_XDB.DBMS_XDB.HASCHARCONTENT  
  abspath IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 163–42 HASCHARCONTENT Function Parameters**

Parameter	Description
abspath_path	Absolute path of the resource

### Return Values

TRUE if the resource has character content.

## HASXMLCONTENT Function

This function returns TRUE if the resource has XML content.

### Syntax

```
DBMS_XDB.DBMS_XDB.HASXMLCONTENT  
  (abspath IN VARCHAR2)  
  RETURN BOOLEAN;
```

### Parameters

**Table 163–43 HASXMLCONTENT Function Parameters**

Parameter	Description
abspath_path	Absolute path of the resource

### Return Values

TRUE if the resource has XML content.



## HASXMLREFERENCE Function

This function returns TRUE if the resource has a REF to XML content.

### Syntax

```
DBMS_XDB.DBMS_XDB.HASXMLREFERENCE  
  (abspath IN VARCHAR2)  
  RETURN BOOLEAN;
```

### Parameters

**Table 163–44 HASXMLREFERENCE Function Parameters**

Parameter	Description
abspath_path	Absolute path of the resource

### Return Values

TRUE resource has a REF to XML content.

## ISFOLDER Function

This function returns TRUE if the resource is a folder or container.

### Syntax

```
DBMS_XDB.DBMS_XDB.ISFOLDER  
    (abspath IN VARCHAR2)  
    RETURN BOOLEAN;
```

### Parameters

**Table 163–45** DBMS\_XDB.ISFOLDER Function Parameters

Parameter	Description
abspath_path	Absolute path of the resource

### Return Values

TRUE if the resource is a folder or container.

## LINK Procedures

This procedure creates from a specified folder to a specified resource.

### Syntax

```
DBMS_XDB.LINK(
  srcpath      IN  VARCHAR2,
  linkfolder   IN  VARCHAR2,
  linkname     IN  VARCHAR2);

DBMS_XDB.LINK(
  srcpath      IN  VARCHAR2,
  linkfolder   IN  VARCHAR2,
  linkname     IN  VARCHAR2,
  linktype     IN  PLS_INTEGER := DBMS_XDB.LINK_TYPE_HARD);
```

### Parameters

**Table 163–46 LINK Procedure Parameters**

Parameter	Description
srcpath	Path name of the resource to which a link is created
linkfolder	Folder in which the new link is placed
linkname	Name of the new link
linktype	Type of link to be created: <ul style="list-style-type: none"> <li>■ DBMS_XDB.LINK_TYPE_HARD (default)</li> <li>■ DBMS_XDB.LINK_TYPE_WEAK</li> <li>■ DBMS_XDB.LINK_TYPE_SYMBOLIC</li> </ul>

## LOCKRESOURCE Function

Given a path to a resource, this function gets a WebDAV-style lock on that resource.

### Syntax

```
DBMS_XDB.LOCKRESOURCE(  
    path      IN  VARCHAR2,  
    depthzero IN  BOOLEAN,  
    shared    IN  boolean)  
RETURN BOOLEAN;
```

### Parameters

**Table 163–47** LOCKRESOURCE Function Parameters

Parameter	Description
path	Path name of the resource to lock.
depthzero	Currently not supported
shared	Passing TRUE obtains a shared write lock

### Return Values

TRUE if successful.

### Usage Notes

The user must have UPDATE privileges on the resource.

## MOVEXDB\_TABLESPACE Procedure

---



---

**Note:** This procedure is deprecated in Release 11g. This functionality is replaced by a subprogram of the same name in the [DBMS\\_XDB\\_ADMIN](#) package - the [MOVEXDB\\_TABLESPACE Procedure](#).

---



---

This procedure moves the XDB (user) to the specified tablespace.

### Syntax

```
DBMS_XDB.MOVEXDB_TABLESPACE(
    new_tablespace IN VARCHAR2);
```

### Parameters

**Table 163–48** *MOVEXDB\_TABLESPACE Procedure Parameters*

Parameter	Description
<code>new_tablespace</code>	Name of the tablespace to where the XDB is moved

### Usage Notes

- This operation waits for all concurrent XDB sessions to exit.
- If `MOVEXDB_TABLESPACE` fails, the user should restart the database before issuing any further command. Failure to do so will produce indeterminable results.

## PROCESSLINKS Procedure

This procedure processes document links in the specified resource.

### Syntax

```
DBMS_XDB.PURGERESOURCEMETADATA(  
  abspath IN VARCHAR2,  
  recurse IN BOOLEAN := FALSE);
```

### Parameters

**Table 163–49** *PROCESSLINKS Procedure Parameters*

Parameter	Description
abspath	Absolute path of the resource. If the path is a folder, use the <code>recurse</code> flag.
recurse	Used only if <code>abspath</code> specifies a folder. If <code>TRUE</code> , process links of all resources in the folder hierarchy rooted at the specified resource. If <code>FALSE</code> , process links of all documents in this folder only.

## PURGERESOURCEMETADATA Procedure

This procedure deletes all user metadata from a resource. Schema-based metadata is removed in cascade mode, rows being deleted from the corresponding metadata tables.

### Syntax

```
DBMS_XDB.PURGERESOURCEMETADATA (  
  
  abspath IN VARCHAR2);
```

### Parameters

**Table 163–50** *PURGERESOURCEMETADATA Procedure Parameters*

Parameter	Description
abspath	Absolute path of the resource

## REBUILDHIERARCHICALINDEX Procedure

---

---

**Note:** This procedure is deprecated in Release 11g. This functionality is replaced by a subprogram of the same name in the [DBMS\\_XDB\\_ADMIN](#) package - the [REBUILDHIERARCHICALINDEX Procedure](#).

---

---

This procedure rebuilds the hierarchical index after import or export operations. This is necessary because data cannot be exported from index tables.

### Syntax

```
DBMS_XDB.REBUILDHIERARCHICALINDEX;
```



## RENAMERESOURCE Procedure

This procedure renames the XDB resource.

### Syntax

```
DBMS_XDB.RENAMERESOURCE(  
    srcpath    IN  VARCHAR2,  
    destfolder IN  CARCHAR2,  
    newname    IN  VARCHAR2);
```

### Parameters

**Table 163–51** *RENAMERESOURCE Procedure Parameters*

Parameter	Description
srcpath	Absolute path in the Hierarchy for the source resource destination folder
destfolder	Absolute path in the Hierarchy for the destination folder
newname	Name of the child in the destination folder

## SETACL Procedure

This procedure sets the ACL on a specified resource to be the ACL specified by path.

### Syntax

```
DBMS_XDB.SETACL(  
    res_path IN VARCHAR2,  
    acl_path IN VARCHAR2);
```

### Parameters

**Table 163–52 SETACL Procedure Parameters**

Parameter	Description
res_path	Absolute path in the Hierarchy for resource
acl_path	Absolute path in the Hierarchy for ACL

### Usage Notes

The user must have `<write-acl>` privileges on the resource.

## SETCONTENT Procedures

This procedure replaces the contents of a resource with a specified datatype.

### Syntax

Replaces the contents of the a resource with a specified CLOB:

```
DBMS_XDB.SETCONTENT (
    abspath    IN   VARCHAR2,
    data       IN   CLOB);
```

Replaces the contents of a resource with a specified BLOB:

```
DBMS_XDB.SETCONTENT (
    abspath    IN   VARCHAR2,
    data       IN   BLOB,
    csid       IN   PLS_INTEGER);
```

Replaces the contents of a resource with a specified XMLTYPE:

```
DBMS_XDB.SETCONTENT (
    abspath    IN   VARCHAR2,
    data       IN   SYS.XMLTYPE);
```

Replaces the contents of a resource with a specified string:

```
DBMS_XDB.SETCONTENT (
    abspath    IN   VARCHAR2,
    data       IN   VARCHAR2);
```

Replaces the contents of a resource with a specified REF to an XMLTYPE

```
DBMS_XDB.SETCONTENT (
    abspath    IN   VARCHAR2,
    data       IN   CLOB,
    sticky     IN   BOOLEAN := TRUE);
```

Replaces the contents of a resource with a specified BFILE:

```
DBMS_XDB.SETCONTENT (
    abspath    IN   VARCHAR2,
    data       IN   CLOB,
    csid       IN   PLS_INTEGER);
```

### Parameters

**Table 163–53** *SETCONTENT Procedure Parameters*

Parameter	Description
abspath	Absolute path in the Hierarchy for resource
data	Input varying with overload: <ul style="list-style-type: none"> <li>■ CLOB data</li> <li>■ BLOB data</li> <li>■ XMLTYPE data</li> <li>■ string data</li> <li>■ REF to XMLTYPE data</li> <li>■ a BFILE</li> </ul>

**Table 163–53 (Cont.) SETCONTENT Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>csid</code>	Character set id of the BLOB or BFILE
<code>sticky</code>	Whether or not the REF is sticky

## SETFTPPOINT Procedure

This procedure sets the FTP port to a new value.

### Syntax

```
DBMS_XDB.SETFTPPOINT(  
    new_port IN NUMBER);
```

### Parameters

**Table 163–54** *SETFTPPOINT Procedure Parameters*

Parameter	Description
new_port	Value to which the FTP port is set

## SETHHTPPORT Procedure

This procedure sets the HTTP port to a new value.

### Syntax

```
DBMS_XDB.SETHHTPPORT(
    new_port IN NUMBER);
```

### Parameters

**Table 163–55** *SETHHTPPORT Procedure Parameters*

Parameter	Description
new_port	Value to which the HTTP port is set

## SETLISTENERENDPOINT Procedure

This procedure sets the parameters of a listener end point corresponding to the XML DB HTTP server. Both HTTP and HTTP2 end points can be set by invoking this procedure.

### Syntax

```
DBMS_XDB.SETLISTENERENDPOINT (
  endpoint IN NUMBER,
  host      IN VARCHAR2,
  port     IN NUMBER,
  protocol  IN NUMBER);
```

### Parameters

**Table 163–56** *SETLISTENERENDPOINT Procedure Parameters*

Parameter	Description
endpoint	End point to be set. Its value can be XDB_ENDPOINT_HTTP or XDB_ENDPOINT_HTTP2.
host	Interface on which the listener end point is to listen. Its value can be 'LOCALHOST,' NULL, or a hostname. If its value is 'LOCALHOST' the listener end point is permitted to only listen on the localhost interface. If its value is NULL or hostname, the listener end point is permitted to listen on both localhost and non-localhost interfaces.
port	Port on which the listener end point is to listen
protocol	Transport protocol that the listener end point is to accept. Its value can be XDB_PROTOCOL_TCP or XDB_PROTOCOL_TCPS

## SETLISTENERLOCALACCESS Procedure

This procedure restricts all listener end points of the XML DB HTTP server to listen either only on the localhost interface (when `l_access` is set to `TRUE`) or to listen on both localhost and non-localhost interfaces (when `l_access` is set to `FALSE`).

### Syntax

```
DBMS_XDB.SETLISTENERLOCALACCESS (  
    l_access    BOOLEAN);
```

### Parameters

**Table 163–57** *SETLISTENERLOCALACCESS Procedure Parameters*

Parameter	Description
<code>l_access.</code>	<code>TRUE</code> or <code>FALSE</code> .



## SPLITPATH Procedure

This procedure splits the path into a parentpath and childpath.

### Syntax

```
DBMS_XDB.SPLITPATH(  
    abspath      IN  VARCHAR2,  
    parentpath  OUT VARCHAR2,  
    childpath   OUT VARCHAR2);
```

### Parameters

**Table 163–58** *SPLITPATH Procedure Parameters*

Parameter	Description
abspath	Absolute path to be split
parentpath	Parentpath
childpath	Childpath

## TOUCHRESOURCE Procedure

This procedure changes the modification time of the resource to the current time.

### Syntax

```
DBMS_XDB.DBMS_XDB.TOUCHRESOURCE  
  (abspath IN VARCHAR2);
```

### Parameters

**Table 163–59** *DBMS\_XDB.TOUCHRESOURCE Procedure Parameters*

Parameter	Description
abspath_path	Absolute path of the resource

## UPDATERESOURCEMETADATA Procedures

This procedure updates metadata for a resource. The procedure takes in a resource identified by absolute path and the metadata in it to replace identified by its REF. It replaces that piece of metadata with user-defined metadata which is either in the form of a REF to XMLTYPE or an XMLTYPE.

### Syntax

Can be used to update schema-based metadata only. The new metadata must be schema-based:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
  abspath  IN VARCHAR2,
  oldmetadata  IN REF SYS.XMLTYPE,
  newmetadata  IN REF SYS.XMLTYPE)
```

Can be used to update schema-based metadata only. The new metadata must be schema-based or nonschema-based:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
  abspath      IN VARCHAR2,
  oldmetadata  IN REF SYS.XMLTYPE,
  newmetadata  IN XMLTYPE);
```

Can be used for both schema-based and nonschema-based metadata:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
  abspath      IN VARCHAR2,
  oldns        IN VARCHAR2,
  oldname      IN VARCHAR,
  newmetadata  IN XMLTYPE);
```

Can be used for both schema-based or nonschema-based metadata. New metadata must be schema-based:

```
DBMS_XDB.UPDATERESOURCEMETADATA(
  abspath      IN VARCHAR2,
  oldns        IN VARCHAR2,
  oldname      IN VARCHAR,
  newmetadata  IN REF SYS.XMLTYPE);
```

### Parameters

**Table 163–60** UPDATERESOURCEMETADATA Procedure Parameters

Parameter	Description
abspath	Absolute path of the resource
oldmetadata	REF to the old of metadata
newmetadata	REF to the new, replacement metadata (can be either schema-based or nonschema-based depending on the overload)
oldns	Namespace identifying old metadata
oldname	Local name identifying old metadata

**Usage Notes**

In the case of `REF`, it stores the `REF` in the resource and the metadata is stored in a separate table. Uniqueness of `REF`s is enforced. In the case where the `XMLTYPE` is passed in, data is parsed to determine if it is schema-based or not and is stored accordingly.

## UNLOCKRESOURCE Function

This function unlocks the resource given a lock token and a path to the resource.

### Syntax

```
DBMS_XDB.UNLOCKRESOURCE(  
    path      IN VARCHAR2,  
    deltoken IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 163–61 UNLOCKRESOURCE Function Parameters**

Parameter	Description
path	Path name to the resource
deltoken	Lock token to be removed

### Return Values

TRUE if operation successful.

### Usage Notes

The user must have UPDATE privileges on the resource.



---

---

## DBMS\_XDB\_ADMIN

The DBMS\_XDB\_ADMIN package provides an interface to manage the Oracle XML DB repository.

**See Also:** *Oracle XML DB Developer's Guide* for information about the Oracle XML DB Repository

This chapter contains the following topics:

- [Using DBMS\\_XDB\\_ADMIN](#)
  - Deprecated Subprograms
  - Security Model
- [Summary of DBMS\\_XDB\\_ADMIN Subprograms](#)

---

## Using DBMS\_XDB\_ADMIN

- [Deprecated Subprograms](#)
- [Security Model](#)



## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only and may be terminated in future releases.

---

---

The following subprograms are deprecated with Oracle Database 11g:

- [CREATEREPOSITORYXMLINDEX Procedure](#)
- [DROPREPOSITORYXMLINDEX Procedure](#)
- [XMLINDEXADDPATH Procedure](#)
- [XMLINDEXREMOVEPATH Procedure](#)

## Security Model

Owned by XDB, the DBMS\_XDB\_ADMIN package must be created by SYS or XDB. The EXECUTE privilege is granted to SYS or XDB or DBA. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XDB\_ADMIN Subprograms

This table lists the package subprograms in alphabetical order.

**Table 164–1 DBMS\_XDB\_ADM Package Subprograms**

Subprogram	Description
<a href="#">CREATEREPOSITORYXMLINDEX Procedure</a> on page 164-6	[Deprecated] Creates an XMLIndex on the XML DB repository
<a href="#">DROPREPOSITORYXMLINDEX Procedure</a> on page 164-7	[Deprecated] Drops the XMLIndex on the XML DB repository
<a href="#">MOVEXDB_TABLESPACE Procedure</a> on page 164-8	Moves the XDB (user) to the specified tablespace
<a href="#">REBUILDHIERARCHICALINDEX Procedure</a> on page 164-9	Rebuilds the hierarchical index after import or export operations
<a href="#">XMLINDEXADDPATH Procedure</a> on page 164-10	[Deprecated] Takes a path in XML DB repository as an input and index all the resources under this given path
<a href="#">XMLINDEXREMOVEPATH Procedure</a> on page 164-11	[Deprecated] Removes the index for the given path

## CREATEREPOSITORYXMLINDEX Procedure

---

---

**Note:** This procedure is deprecated in Release 11g.

---

---

This procedure creates an XMLIndex on the XML DB repository.

### Syntax

```
DBMS_XDB_ADMIN.CREATEREPOSITORYXMLINDEX;
```

## DROPREPOSITORYXMLINDEX Procedure

---

---

**Note:** This procedure is deprecated in Release 11g.

---

---

This procedure drops the XMLIndex on the XML DB repository.

### Syntax

```
DBMS_XDB_ADMIN.DROPREPOSITORYXMLINDEX;
```

## MOVEXDB\_TABLESPACE Procedure

This procedure moves the XDB (user) to the specified tablespace.

### Syntax

```
DBMS_XDB.MOVEXDB_TABLESPACE(  
    new_tablespace IN VARCHAR2);
```

### Parameters

**Table 164–2** *MOVEXDB\_TABLESPACE Procedure Parameters*

Parameter	Description
<code>new_tablespace</code>	Name of the tablespace to where the XDB is moved

### Usage Notes

- This operation waits for all concurrent XDB sessions to exit.
- If `MOVEXDB_TABLESPACE` fails, the user should restart the database before issuing any further command. Failure to do so will result into unexpected behavior from the database.

## REBUILDHIERARCHICALINDEX Procedure

This procedure rebuilds the hierarchical index after import or export operations. This is necessary because data cannot be exported from index tables.

### Syntax

```
DBMS_XDB.REBUILDHIERARCHICALINDEX;
```

## XMLINDEXADDPATH Procedure

---

---

**Note:** This procedure is deprecated in Release 11g.

---

---

This procedure adds to the repository xmlindex the resource identified by path (when `recurse` is `FALSE`) or adds to the repository xmlindex the sub-tree of resources rooted at path (when `recurse` is `TRUE`). The default value for `recurse` is `TRUE`.

### Syntax

```
DBMS_XDB_ADMIN.XMLINDEXADDPATH(  
    path          IN VARCHAR2,  
    recurse       IN BOOLEAN);
```

### Parameters

**Table 164–3** XMLINDEXADDPATH Procedure Parameters

Parameter	Description
path	Path to a resource
recurse	TRUE or FALSE



## XMLINDEXREMOVEPATH Procedure

---



---

**Note:** This procedure is deprecated in Release 11g.

---



---

This procedure removes from the repository `xmlindex` the resource identified by `path` (when `recurse` is `FALSE`) or removes from the repository `xmlindex` the sub-tree of resources rooted at `path` (when `recurse` is `TRUE`). The default value for `recurse` is `TRUE`.

### Syntax

```
DBMS_XDB_ADMIN.XMLINDEXREMOVEPATH (
  path          IN VARCHAR2,
  isrecursive   IN BOOLEAN);
```

### Parameters

**Table 164–4 XMLINDEXREMOVEPATH Procedure Parameters**

Parameter	Description
<code>path</code>	Path to a resource
<code>recurse</code>	TRUE or FALSE



---

---

## DBMS\_XDB\_VERSION

Oracle XML DB versioning interfaces are found in the `DBMS_XDB_VERSION` package. Functions and procedures of `DBMS_XDB_VERSION` help to create a VCR and manage the versions in the version history.

This chapter contains the following topic:

- [Using DBMS\\_XDB\\_VERSION](#)
  - Security Model
- [Summary of DBMS\\_XDB\\_VERSION Subprograms](#)

**See Also:** *Oracle XML DB Developer's Guide*

---

## Using DBMS\_XDB\_VERSION

- [Security Model](#)

## Security Model

Owned by XDB, the DBMS\_XDB\_VERSION package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XDB\_VERSION Subprograms

**Table 165–1 DBMS\_XDB\_VERSION Package Subprograms**

Method	Description
<a href="#">CHECKIN Function</a> on page 165-5	Checks in a checked-out VCR and returns the resource id of the newly-created version
<a href="#">CHECKOUT Procedure</a> on page 165-6	Checks out a VCR before updating or deleting it
<a href="#">GETCONTENTSBOBBYRES ID Function</a> on page 165-7	Obtain contents as a BLOB
<a href="#">GETCONTENTSCLOB BYRES ID Function</a> on page 165-8	Obtain contents as a CLOB
<a href="#">GETCONTENTSXMLBYRES ID Function</a> on page 165-9	Obtain contents as an XMLType
<a href="#">GETPREDECESSORS Function</a> on page 165-10	Retrieves the list of predecessors by path name
<a href="#">GETPREDSBYRESID Function</a> on page 165-11	Retrieves the list of predecessors by resource id
<a href="#">GETRESOURCEBYRESID Function</a> on page 165-12	Obtains the resource as an XMLType, given the resource object ID
<a href="#">GETSUCCESSORS Function</a> on page 165-13	Retrieves the list of successors by path name
<a href="#">GETSUCCSBYRESID Function</a> on page 165-14	Retrieves the list of successors by resource id
<a href="#">MAKEVERSIONED Function</a> on page 165-15	Turns a regular resource whose path name is given into a version-controlled resource
<a href="#">UNCHECKOUT Function</a> on page 165-16	Checks in a checked-out resource, returns the resource id of the version before the resource is checked out

## CHECKIN Function

This function checks in a checked-out VCR and returns the resource id of the newly-created version.

### Syntax

```
DBMS_XDB_VERSION.CHECKIN(  
    pathname VARCHAR2)  
RETURN DBMS_XDB.resid_type;
```

### Parameters

**Table 165–2** CHECKIN Function Parameters

Parameter	Description
pathname	The path name of the checked-out resource.

### Usage Notes

This is not an auto-commit SQL operation. [CHECKIN Function](#) doesn't have to take the same path name that was passed to [CHECKOUT Procedure](#) operation. However, the [CHECKIN Function](#) path name and the [CHECKOUT Procedure](#) path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to [CHECKIN Function](#) because the old name is either invalid or is currently bound with a different resource. Exception is raised if the path name does not exist. If the path name has been changed, the new path name must be used to [CHECKIN Function](#) the resource.

## CHECKOUT Procedure

This procedure checks out a VCR before updating or deleting it.

### Syntax

```
DBMS_XDB_VERSION.Checkout (  
    pathname    VARCHAR2);
```

### Parameters

**Table 165–3** CHECKOUT Procedure Parameters

Parameter	Description
pathname	The path name of the VCR to be checked out.

### Usage Notes

This is not an auto-commit SQL operation. Two users of the same workspace cannot [CHECKOUT Procedure](#) the same VCR at the same time. If this happens, one user must rollback. As a result, it is good practice to commit the [CHECKOUT Procedure](#) operation before updating a resource and avoid loss of the update if the transaction is rolled back. An exception is raised if the given resource is not a VCR, if the VCR is already checked out, if the resource doesn't exist.



## GETCONTENTSBLOBBYRESID Function

This function obtain contents as a BLOB.

### Syntax

```
DBMS_XDB_VERSION.GETCONTENTSBLOBBYRESID(  
    resid      DBMS_XDB.resid_type)  
RETURN BLOB;
```

### Parameters

**Table 165–4** *GETCONTENTSBLOBBYRESID Function Parameters*

Parameter	Description
resid	The resource id.

## GETCONTENTSCLobbyRESID Function

This function obtains contents as a CLOB.

### Syntax

```
DBMS_XDB_VERSION.GETCONTENTSCLobbyRESID(  
    resid      DBMS_XDB.resid_type)  
RETURN CLOB;
```

### Parameters

**Table 165–5** *GETCONTENTSCLobbyRESID Function Parameters*

Parameter	Description
resid	The resource id.

## GETCONTENTSXMLBYRESID Function

This function obtains contents as an XMLType.

### Syntax

```
DBMS_XDB_VERSION.GETCONTENTSXMLBYRESID(  
    resid      DBMS_XDB.resid_type)  
RETURN XMLType;
```

### Parameters

**Table 165–6** *GETCONTENTSXMLBYRESID Function Parameters*

Parameter	Description
resid	The resource id.

### Return Values

If the contents are not valid XML, returns NULL.

## GETPREDECESSORS Function

This function retrieves the list of predecessors by the path name.

### Syntax

```
DBMS_XDB_VERSION.GETPREDECESSORS (  
    pathname          VARCHAR2)  
RETURN resid_list_type;
```

### Parameters

**Table 165–7** *GETPREDECESSORS Function Parameters*

Parameter	Description
pathname	The path name of the resource.

### Return Values

An exception is raised if `pathname` is illegal.

## GETPREDSBYRESID Function

This function retrieves the list of predecessors by resource id.

### Syntax

```
DBMS_XDB_VERSION.GETPREDSBYRESID(  
    resid      resid_type)  
RETURN resid_list_type;
```

### Parameters

**Table 165–8** *GETPREDSBYRESID Function Parameters*

Parameter	Description
resid	The resource id.

### Usage Notes

Getting predecessors by RESID is more efficient than by pathname.

### Exceptions

An exception is raised if the RESID is illegal.

## GETRESOURCEBYRESID Function

This function obtains the resource as an `XMLType`, given the resource object ID. Because the system does not create a path name for versions, this function is useful for retrieving the resource using its resource id.

### Syntax

```
DBMS_XDB_VERSION.GETRESOURCEBYRESID(  
    resid      resid_type)  
RETURN XMLType;
```

### Parameters

**Table 165–9** *GETRESOURCEBYRESID Function Parameters*

Parameter	Description
resid	The resource id.

## GETSUCCESSORS Function

Given a version resource or a VCR, this function retrieves the list of the successors of the resource by the path name.

### Syntax

```
DBMS_XDB_VERSION.GETSUCCESSORS(  
    pathname VARCHAR2)  
RETURN resid_list_type;
```

### Parameters

**Table 165–10** *GETSUCCESSORS Function Parameters*

Parameter	Description
pathname	The path name of the resource.

### Usage Notes

Getting successors by RESID is more efficient than by pathname.

### Exceptions

An exception is raised if the pathname is illegal.

## GETSUCCSBYRESID Function

This function retrieves the list of the successors of the resource by resource id using version resource or VCR.

### Syntax

```
DBMS_XDB_VERSION.GETSUCCSBYRESID(  
    resid    resid_type)  
RETURN resid_list_type;
```

### Parameters

**Table 165–11** *GETSUCCSBYRESID Function Parameters*

Parameter	Description
resid	The resource id.

### Usage Notes

Getting successors by RESID is more efficient than by pathname.

### Exceptions

An exception is raised if the pathname is illegal.



## MAKEVERSIONED Function

This function turns a regular resource whose path name is given into a version-controlled resource. This new resource is then put under version control. All other path names continue to refer to the original resource.

### Syntax

```
DBMS_XDB_VERSION.MAKEVERSIONED(
    pathname  VARCHAR2)
RETURN DBMS_XDB.resid_type;
```

### Parameters

**Table 165–12** MAKEVERSIONED Function Parameters

Parameter	Description
pathname	The path name of the resource to be put under version control.

### Return Values

This function returns the resource ID of the first version, or root, of the VCR.

### Usage Notes

If two or more path names are bound with the same resource, a copy of the resource is created, and the given path name is bound with the newly-created copy.

This is not an auto-commit SQL operation. An exception is raised if the resource doesn't exist.

- This call is legal for VCR, and neither exception nor warning is raised.
- This call is illegal for folder, version history, version resource, and ACL.
- No support for Schema-based resources is provided.

## UNCHECKOUT Function

This function checks-in a checked-out resource and returns the resource id of the version before the resource is checked out.

### Syntax

```
DBMS_XDB_VERSION.UNCHECKOUT(  
    pathname    VARCHAR2)  
RETURN DBMS_XDB.resid_type;
```

### Parameters

**Table 165–13** *UNCHECKOUT Function Parameters*

Parameter	Description
pathname	The path name of the checked-out resource.

### Usage Notes

This is not an auto-commit SQL operation. The [UNCHECKOUT Function](#) does not have to take the same path name that was passed to the operation by the [CHECKOUT Procedure](#). However, the [UNCHECKOUT Function](#) path name and the [CHECKOUT Procedure](#) path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to [UNCHECKOUT Function](#), because the old name is either invalid or is currently bound with a different resource. If the path name has been changed, the new path name must be used to [UNCHECKOUT Function](#) the resource.

### Exceptions

An exception is raised if the path name doesn't exist.

---

---

## DBMS\_XDBRESOURCE

The DBMS\_XDBRESOURCE package provides the interface to operate on the resource's metadata and contents.

**See Also:** Oracle XML DB Developer's Guide for examples of "Using DBMS\_XDBRESOURCE"

This chapter contains the following topics:

- [Using DBMS\\_XDBRESOURCE](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_XDBRESOURCE Subprograms](#)

## Using DBMS\_XDBRESOURCE

- [Overview](#)
- [Security Model](#)

## Overview

The `DBMS_XDBRESOURCE` package provides routines to get and set the resource's metadata and contents. To take advantage of the DOM traversal facility, provided in `DBMS_XMLDOM` package, an `XDBResource` instance could be converted to a `DOMDocument` type by using `DBMS_XDBRESOURCE.MAKEDOCUMENT` routine.

## Security Model

Owned by XDB, the DBMS\_XDBRESOURCE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XDBRESOURCE Subprograms

**Table 166–1 DBMS\_XDBRESOURCE Package Subprograms**

Subprogram	Description
<a href="#">FREERESOURCE Procedure</a> on page 166-8	Frees any memory associated with an XDBResource
<a href="#">GETACL Function</a> on page 166-9	Given an XDBResource, returns its ACL as string
<a href="#">GETACLDOKFROMRES Function</a> on page 166-10	Returns the ACL Document for the given resource as XMLType
<a href="#">GETAUTHOR Function</a> on page 166-11	Given an XDBResource, returns its author
<a href="#">GETCHARACTERSET Function</a> on page 166-12	Given an XDBResource, returns its character set
<a href="#">GETCOMMENT Function</a> on page 166-13	Given an XDBResource, returns its comment
<a href="#">GETCONTENTBLOB Function</a> on page 166-14	Returns the contents of the resource as a BLOB
<a href="#">GETCONTENTCLOB Function</a> on page 166-15	Returns the contents of the resource as a CLOB
<a href="#">GETCONTENTREF Function</a> on page 166-16	Returns the contents of the resource as an XMLTypeRef
<a href="#">GETCONTENTTYPE Function</a> on page 166-17	Given an XDBResource, returns its content-type
<a href="#">GETCONTENTXML Function</a> on page 166-18	Returns the contents of the resource as XML
<a href="#">GETCONTENTVARCHAR2 Function</a> on page 166-19	Returns the contents of the resource as a string
<a href="#">GETCREATIONDATE Function</a> on page 166-20	Given an XDBResource, returns its creation date
<a href="#">GETCREATOR Function</a> on page 166-21	Given an XDBResource, returns its creator
<a href="#">GETCUSTOMMETADATA Function</a> on page 166-22	Returns the requested custom metadata given the xpath and namespace to the metadata
<a href="#">GETDISPLAYNAME Function</a> on page 166-23	Given an XDBResource, returns its display name
<a href="#">GETLANGUAGE Function</a> on page 166-24	Given an XDBResource, returns its language
<a href="#">GETLASTMODIFIER Function</a> on page 166-25	Given an XDBResource, returns its last modifier
<a href="#">GETMODIFICATIONDATE Function</a> on page 166-26	Given an XDBResource, returns its modification date
<a href="#">GETOWNER Function</a> on page 166-27	Given an XDBResource, returns its owner.
<a href="#">GETREFCOUNT Function</a> on page 166-28	Given an XDBResource, returns its reference count
<a href="#">GETVERSIONID Function</a> on page 166-29	Given an XDBResource, returns its version ID.
<a href="#">HASACLCHANGED Function</a> on page 166-30	Returns TRUE if the ACL of the given resource has changed, FALSE otherwise

**Table 166–1 (Cont.) DBMS\_XDBRESOURCE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">HASAUTHORCHANGED Function</a> on page 166-31	Returns <code>TRUE</code> if the ACL of the given resource has changed <code>FALSE</code> otherwise
<a href="#">HASCHANGED Function</a> on page 166-32	Returns <code>TRUE</code> if the element or attribute represented by the given XPath has changed, <code>FALSE</code> otherwise
<a href="#">HASCHARACTERSETCHANGED Function</a> on page 166-33	Returns <code>TRUE</code> if the character set of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASCOMMENTCHANGED Function</a> on page 166-34	Returns <code>TRUE</code> if the comment of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASCONTENTCHANGED Function</a> on page 166-35	Returns <code>TRUE</code> if the contents of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASCONTENTTYPECHANGED Function</a> on page 166-36	Returns <code>TRUE</code> if the content-type of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASCREATIONDATECHANGED Function</a> on page 166-37	Returns <code>TRUE</code> if the creation date of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASCREATORCHANGED Function</a> on page 166-38	Returns <code>TRUE</code> if the creator of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASCUSTOMMETADATACHANGED Function</a> on page 166-39	Returns <code>TRUE</code> if custom-metadata for this XPath has changed, <code>FALSE</code> otherwise
<a href="#">HASDISPLAYNAMECHANGED Function</a> on page 166-40	Returns <code>TRUE</code> if the display name of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASLANGUAGECHANGED Function</a> on page 166-41	Returns <code>TRUE</code> if the language of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASLASTMODIFIERCHANGED Function</a> on page 166-42	Returns <code>TRUE</code> if the last modifier of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASMODIFICATIONDATECHANGED Function</a> on page 166-43	Returns <code>TRUE</code> if the modification date of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASOWNERCHANGED Function</a> on page 166-44	Returns <code>TRUE</code> if the owner of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASREFCOUNTCHANGED Function</a> on page 166-45	Returns <code>TRUE</code> if the reference count of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">HASVERSIONIDCHANGED Function</a> on page 166-46	Returns <code>TRUE</code> if the version ID of the given resource has changed, <code>FALSE</code> otherwise
<a href="#">ISFOLDER Function</a> on page 166-47	Returns <code>TRUE</code> if the given resource is a folder, <code>FALSE</code> otherwise
<a href="#">ISNULL Function</a> on page 166-48	Returns <code>TRUE</code> if input resource is <code>NULL</code> , <code>FALSE</code> otherwise
<a href="#">MAKEDOCUMENT Function</a> on page 166-49	Converts the XDBResource to a <code>DOMDocument</code> which can be operated on using the <code>XMLDOM</code> interface
<a href="#">SAVE Procedure</a> on page 166-50	Updates the resource with any modifications
<a href="#">SETACL Procedure</a> on page 166-51	Sets the ACL of the given XDBResource to the path specified
<a href="#">SETAUTHOR Procedure</a> on page 166-52	Sets the author of the given XDBResource to the specified string
<a href="#">SETCHARACTERSET Procedure</a> on page 166-53	Sets the character set of the given XDBResource to a specified character set



**Table 166-1 (Cont.) DBMS\_XDBRESOURCE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SETCOMMENT Procedure</a> on page 166-54	Sets a comment associated with the given XDBResource
<a href="#">SETCONTENT Procedures</a> on page 166-55	Replaces the contents of the given resource with the given CLOB
<a href="#">SETCONTENTTYPE Procedure</a> on page 166-56	Sets the content-type of the given XDBResource
<a href="#">SETCUSTOMMETADATA Procedure</a> on page 166-57	Sets the custom metadata specified by the XPath and namespace to new data
<a href="#">SETDISPLAYNAME Procedure</a> on page 166-58	Sets the display name of the given XDBResource
<a href="#">SETLANGUAGE Procedure</a> on page 166-59	Sets the language of the given XDBResource
<a href="#">SETOWNER Procedure</a> on page 166-60	Sets the owner of the given XDBResource

## FREERESOURCE Procedure

This procedure frees any memory associated with an XDBResource.

### Syntax

```
DBMS_XDBRESEROUCE.FREERESOURCE (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–2** *FREERESOURCE Procedure Parameters*

Parameter	Description
res	XDBResource to free

## GETACL Function

Given an XDBResource, this function returns its ACL as string.

### Syntax

```
DBMS_XDBRESEROUCE.GETACL (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–3** *GETACL Function Parameters*

Parameter	Description
res	XDBResource

## GETACLDOCFROMRES Function

This function returns the ACL Document for the given resource as XMLType.

### Syntax

```
DBMS_XDBRESEROUCE.GETACLDOCFROMRES (  
    res IN XDBResource)  
RETURN SYS.XMLTYPE;
```

### Parameters

**Table 166–4** *GETACL Function Parameters*

Parameter	Description
res	XDBResource

## GETAUTHOR Function

Given an XDBResource, this function returns its author.

### Syntax

```
DBMS_XDBRESEROUCE.GETAUTHOR (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–5** *GETAUTHOR Function Parameters*

Parameter	Description
res	XDBResource

## GETCHARACTERSET Function

Given an XDBResource, this function returns its character set.

### Syntax

```
DBMS_XDBRESEROUCE.GETCHARACTERSET (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–6** *GETCHARACTERSET Function Parameters*

Parameter	Description
res	XDBResource

## GETCOMMENT Function

Given an XDBResource, this function returns its comment.

### Syntax

```
DBMS_XDBRESEROUCE.GETCOMMENT (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–7** *GETCOMMENT Function Parameters*

Parameter	Description
res	XDBResource

## GETCONTENTBLOB Function

This function returns the contents of the resource as a BLOB.

### Syntax

```
DBMS_XDBRESEROUCE.GETCONTENTBLOB (  
    res    IN    XDBResource,  
    csid   OUT   PLS_INTEGER)  
RETURN BLOB;
```

### Parameters

**Table 166–8** *GETCONTENTBLOB Function Parameters*

Parameter	Description
res	XDBResource
csid	Character set ID of the BLOB returned



## GETCONTENTCLOB Function

This function returns the contents of the resource as a CLOB.

### Syntax

```
DBMS_XDBRESEROUCE.GETCONTENTCLOB (  
    res IN XDBResource)  
RETURN CLOB;
```

### Parameters

**Table 166–9** *GETCONTENTCLOB Function Parameters*

Parameter	Description
res	XDBResource

## GETCONTENTREF Function

This function returns the contents of the resource as an XMLTypeRef.

### Syntax

```
DBMS_XDBRESEROUCE.GETCONTENTREF (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–10** *GETCONTENTREF Function Parameters*

Parameter	Description
res	XDBResource

## GETCONTENTTYPE Function

Given an XDBResource, this function returns its content-type.

### Syntax

```
DBMS_XDBRESEROUCE.GETCONTENTTYPE (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–11** *GETCONTENTTYPE Function Parameters*

Parameter	Description
res	XDBResource

## GETCONTENTXML Function

This function returns the contents of the resource as an XMLTypeRef.

### Syntax

```
DBMS_XDBRESEROUCE.GETCONTENTXML (  
    res IN XDBResource)  
RETURN XMLType;
```

### Parameters

**Table 166–12** *GETCONTENTXML Function Parameters*

Parameter	Description
res	XDBResource

## GETCONTENTVARCHAR2 Function

This function returns the contents of the resource as a string.

### Syntax

```
DBMS_XDBRESEROUCE.GETCONTENTVARCHAR2 (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–13** *GETCONTENTVARCHAR2 Function Parameters*

Parameter	Description
res	XDBResource

## GETCREATIONDATE Function

Given an XDBResource, this function returns its creation date.

### Syntax

```
DBMS_XDBRESEROUCE.GETCREATIONDATE (  
    res IN XDBResource)  
RETURN TIMESTAMP;
```

### Parameters

**Table 166–14** *GETCREATIONDATE Function Parameters*

Parameter	Description
res	XDBResource

## GETCREATOR Function

Given an XDBResource, this function returns its creator.

### Syntax

```
DBMS_XDBRESEROUCE.GETCREATOR (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–15** *GETCREATOR Function Parameters*

Parameter	Description
res	XDBResource

## GETCUSTOMMETADATA Function

This function returns the requested custom metadata given the xpath and namespace to the metadata.

### Syntax

```
DBMS_XDBRESEROUCE.GETCUSTOMMETADATA (  
    res          IN    XDBResource,  
    xpath        IN    VARCHAR2,  
    namespace    IN    VARCHAR2)  
RETURN XMLType;
```

### Parameters

**Table 166–16** *GETCUSTOMMETADATA Function Parameters*

Parameter	Description
res	XDBResource
xpath	XPath for custom metadata
namespace	Namespace

### Usage Notes

The first component of the XPath expression must be "Resource".



## GETDISPLAYNAME Function

Given an XDBResource, this function returns its display name.

### Syntax

```
DBMS_XDBRESEROUCE.GETDISPLAYNAME (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–17** *GETDISPLAYNAME Function Parameters*

Parameter	Description
res	XDBResource

## GETLANGUAGE Function

Given an XDBResource, this function returns its language.

### Syntax

```
DBMS_XDBRESEROUCE.GETLANGUAGE (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–18** *GETLANGUAGE Function Parameters*

Parameter	Description
res	XDBResource

## GETLASTMODIFIER Function

Given an XDBResource, this function returns its last modifier.

### Syntax

```
DBMS_XDBRESEROUCE.GETLASTMODIFIER (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–19** *GETLASTMODIFIER Function Parameters*

Parameter	Description
res	XDBResource

## GETMODIFICATIONDATE Function

Given an XDBResource, this function returns its modification date.

### Syntax

```
DBMS_XDBRESEROUCE.GETMODIFICATIONDATE (  
    res IN XDBResource)  
RETURN TIMESTAMP;
```

### Parameters

**Table 166–20** *GETMODIFICATIONDATE Function Parameters*

Parameter	Description
res	XDBResource

## GETOWNER Function

Given an XDBResource, this function returns its owner.

### Syntax

```
DBMS_XDBRESOURCE.GETOWNER (  
    res IN XDBResource)  
RETURN VARCHAR2;
```

### Parameters

**Table 166–21** *GETOWNER Function Parameters*

Parameter	Description
res	XDBResource

## GETREFCOUNT Function

Given an XDBResource, this function returns its reference count.

### Syntax

```
DBMS_XDBRESEROUCE.GETREFCOUNT (  
    res IN XDBResource)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 166–22** *GETREFCOUNT Function Parameters*

Parameter	Description
res	XDBResource

## GETVERSIONID Function

Given an XDBResource, this function returns its version ID.

### Syntax

```
DBMS_XDBRESEROUCE.GETVERSIONID (  
    res IN XDBResource)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 166–23** *GETVERSIONID Function Parameters*

Parameter	Description
res	XDBResource

## HASACLCHANGED Function

This function returns `TRUE` if the ACL of the given resource has changed, `FALSE` otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASACLCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–24** *GETACL Function Parameters*

Parameter	Description
<code>res</code>	XDBResource



## HASAUTHORCHANGED Function

This function returns TRUE if the author of the given resource has changed, FALSE otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASAUTHORCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–25 HASAUTHORCHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASCHANGED Function

Given an XPath, this function determines whether the element or attribute represented by the XPath has changed.

### Syntax

```
DBMS_XDBRESEROUCE.HASCHANGED (  
    res          IN    XDBResource,  
    xpath        IN    VARCHAR2,  
    namespace    IN    VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–26** *HASCHANGED Function Parameters*

Parameter	Description
res	XDBResource
xpath	XPath to check
bnamespace	Namespace to use

## HASCHARACTERSETCHANGED Function

This function returns TRUE if the character set of the given resource has changed, FALSE otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASCHARACTERSETCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–27 HASCHARACTERSETCHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASCOMMENTCHANGED Function

This function returns `TRUE` if the comment of the given resource has changed, `FALSE` otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASCOMMENTCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–28 HASCOMMENTCHANGED Function Parameters**

Parameter	Description
<code>res</code>	XDBResource

## HASCONTENTCHANGED Function

This function returns TRUE if the contents of the given resource has changed, FALSE otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASCONTENTCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–29 HASCONTENTCHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASCONTENTTYPECHANGED Function

This function returns TRUE if the content-type of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASCONTENTTYPECHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–30 HASCONTENTTYPECHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASCREATIONDATECHANGED Function

This function returns TRUE if the creation date of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASCREATIONDATECHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–31 HASCREATIONDATECHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASCREATORCHANGED Function

This function returns TRUE if the creator of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASCREATORCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–32 HASCREATORCHANGED Function Parameters**

Parameter	Description
res	XDBResource



## HASCUSTOMMETADATACHANGED Function

This function checks whether the custom-metadata for a given resource has changed.

### Syntax

```
DBMS_XDBRESEROUCE.HASCUSTOMMETADATACHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–33** *HASCUSTOMMETADATACHANGED Function Parameters*

Parameter	Description
res	XDBResource

## HASDISPLAYNAMECHANGED Function

This function returns TRUE if the display name of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASDISPLAYNAMECHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–34 HASDISPLAYNAMECHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASLANGUAGECHANGED Function

This function returns TRUE if the language of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASLANGUAGECHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–35 HASLANGUAGECHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASLASTMODIFIERCHANGED Function

This function returns TRUE if the last modifier of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASLASTMODIFIERCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–36 HASLASTMODIFIERCHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASMODIFICATIONDATECHANGED Function

This function returns TRUE if the modification date of the given resource has changed, FALSE otherwise

### Syntax

```
DBMS_XDBRESEROUCE.HASMODIFICATIONDATECHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–37 HASMODIFICATIONDATECHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASOWNERCHANGED Function

This function returns TRUE if the owner of the given resource has changed, FALSE otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASOWNERCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–38 HASOWNERCHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASREFCOUNTCHANGED Function

This function returns TRUE if the reference count of the given resource has changed, FALSE otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASREFCOUNTCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–39 HASREFCOUNTCHANGED Function Parameters**

Parameter	Description
res	XDBResource

## HASVERSIONIDCHANGED Function

This function returns `TRUE` if the version ID of the given resource has changed, `FALSE` otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.HASVERSIONIDCHANGED (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–40 HASVERSIONIDCHANGED Function Parameters**

Parameter	Description
<code>res</code>	XDBResource



## ISFOLDER Function

This function returns TRUE if the given resource is a folder, FALSE otherwise.

### Syntax

```
DBMS_XDBRESEROUCE.ISFOLDER (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–41** *ISFOLDER Function Parameters*

Parameter	Description
res	XDBResource

## ISNULL Function

This function returns TRUE if input resource is NULL.

### Syntax

```
DBMS_XDBRESEROUCE.ISNULL (  
    res IN XDBResource)  
RETURN BOOLEAN;
```

### Parameters

**Table 166–42 ISNULL Function Parameters**

Parameter	Description
res	Input resource

## MAKEDOCUMENT Function

This function converts the XDBResource to a DOMDocument which can be operated on using the XMLDOM interface.

**See Also:** The [DBMS\\_XMLDOM](#) package

### Syntax

```
DBMS_XDBRESOURCE.MAKEDOCUMENT (  
    res IN XDBResource)  
RETURN DBMS_XMLDOM.DOMDocument;
```

### Parameters

**Table 166-43** *MAKEDOCUMENT Function Parameters*

Parameter	Description
res	XDBResource

## SAVE Procedure

This procedure updates the resource with any modifications.

### Syntax

```
DBMS_XDBRESEROUCE.SAVE (  
    res IN XDBResource);
```

### Parameters

**Table 166–44** *SAVE Procedure Parameters*

Parameter	Description
res	XDBResource

## SETACL Procedure

This procedure sets the ACL of the given XDBResource to the path specified.

### Syntax

```
DBMS_XDBRESOURCE.SETACL (  
    res      IN OUT  XDBResource,  
    ACLPath  IN      VARCHAR2);
```

### Parameters

**Table 166–45** SETACL Procedure Parameters

Parameter	Description
res	XDBResource
ACLPath	Absolute path of the new ACL

## SETAUTHOR Procedure

This procedure sets the author of the given XDBResource to the specified string.

### Syntax

```
DBMS_XDBRESEROUCE.SETAUTHOR (  
    res      IN OUT  XDBResource,  
    author   IN      VARCHAR2);
```

### Parameters

**Table 166–46** *SETAUTHOR Procedure Parameters*

Parameter	Description
res	XDBResource
author	Author

## SETCHARACTERSET Procedure

This procedure sets the character set of the given XDBResource to a specified character set.

### Syntax

```
DBMS_XDBRESEROUCE.SETCHARACTERSET (  
    res      IN OUT  XDBResource,  
    charSet  IN      VARCHAR2);
```

### Parameters

**Table 166–47** *SETCHARACTERSET Procedure Parameters*

Parameter	Description
res	XDBResource
charset	New character set

## SETCOMMENT Procedure

This procedure sets a comment associated with the given XDBResource.

### Syntax

```
DBMS_XDBRESEROUCE.SETCOMMENT (  
    res      IN OUT  XDBResource,  
    comment IN      VARCHAR2);
```

### Parameters

**Table 166–48** *SETCOMMENT Procedure Parameters*

Parameter	Description
res	XDBResource
comment	New comment



## SETCONTENT Procedures

This procedure replaces the contents of the given resource with the given datatype.

### Syntax

```
DBMS_XDBRESOURCE.SETCONTENT (
  res      IN OUT  XDBResource,
  data     IN      BFILE,
  csid     IN      NUMBER);
```

```
DBMS_XDBRESOURCE.SETCONTENT (
  res      IN OUT  XDBResource,
  data     IN      BLOB,
  csid     IN      PLS_INTEGER);
```

```
DBMS_XDBRESOURCE.SETCONTENT (
  res      IN OUT  XDBResource,
  data     IN      CLOB);
```

```
DBMS_XDBRESOURCE.SETCONTENT (
  res      IN OUT  XDBResource,
  data     IN      REF SYS.XMLType,
  sticky   IN      BOOLEAN := TRUE);
```

```
DBMS_XDBRESOURCE.SETCONTENT (
  res      IN OUT  XDBResource,
  data     IN      VARCHAR2);
```

```
DBMS_XDBRESOURCE.SETCONTENT (
  res      IN OUT  XDBResource,
  data     IN      SYS.XMLType);
```

### Parameters

**Table 166–49** *SETCONTENT Procedure Parameters*

Parameter	Description
res	XDBResource
data	Data input as BFILE, BLOB, CLOB, string, XMLType
csid	Character set ID of the BFILE, BLOB
sticky	If TRUE creates a sticky REF, otherwise non-sticky

## SETCONTENTTYPE Procedure

This procedure sets the content-type of the given XDBResource.

### Syntax

```
DBMS_XDBRESEROUCE.SETCONTENTTYPE (  
    res          IN OUT  XDBResource,  
    conttype     IN      VARCHAR2);
```

### Parameters

**Table 166–50** *SETCONTENTTYPE Procedure Parameters*

Parameter	Description
res	XDBResource
conttype	New content-type

## SETCUSTOMMETADATA Procedure

This procedure sets the custom metadata specified by the xpath and namespace to new data.

### Syntax

```
DBMS_XDBRESOURCE.SETCUSTOMMETADATA (
  res          IN OUT  XDBResource,
  xpath        IN      VARCHAR2,
  namespace    IN      VARCHAR2,
  newMetadata  IN      XMLType);
```

### Parameters

**Table 166–51** SETCUSTOMMETADATA Procedure Parameters

Parameter	Description
res	XDBResource
xpath	XPath to change
namespace	Namespace to use
newMetadata	New data that should replace the metadata at the given XPath

### Usage Notes

The first component of the XPath expression must be "Resource".

## SETDISPLAYNAME Procedure

This procedure sets the display name of the given XDBResource.

### Syntax

```
DBMS_XDBRESEROUCE.SETDISPLAYNAME (  
    res      IN OUT  XDBResource,  
    name     IN     VARCHAR2);
```

### Parameters

**Table 166–52** *SETDISPLAYNAME Procedure Parameters*

Parameter	Description
res	XDBResource
name	New display name

## SETLANGUAGE Procedure

This procedure sets the language of the given XDBResource.

### Syntax

```
DBMS_XDBRESOURCE.SETLANGUAGE (  
    res      IN OUT  XDBResource,  
    ACLPath  IN      VARCHAR2);
```

### Parameters

**Table 166–53** *SETLANGUAGE Procedure Parameters*

Parameter	Description
res	XDBResource
ACLPath	New path

## SETOWNER Procedure

This procedure sets the owner of the given XDBResource.

### Syntax

```
DBMS_XDBRESEROUCE.SETOWNER (  
    res      IN OUT  XDBResource,  
    owner    IN      VARCHAR2);
```

### Parameters

**Table 166–54** *SETOWNER Procedure Parameters*

Parameter	Description
res	XDBResource
owner	New owner

### Usage Notes

The user must have the XDBADMIN privilege to call this subprogram.

The `DBMS_XDBT` package provides a convenient mechanism for administrators to set up a `CONTEXT` index on the Oracle XML DB hierarchy. The package contains procedures to create default preferences, create the index and set up automatic synchronization of the `CONTEXT` index

The `DBMS_XDBT` package also contains a set of package variables that describe the configuration settings for the index. These are intended to cover the basic customizations that installations may require, but is by no means a complete set.

**See Also:** *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_XDBT](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of DBMS\\_XDBT Subprograms](#)

## Using DBMS\_XDBT

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)



## Overview

The DBMS\_XDBT package can be used in the following fashion:

- Customize the package to set up the appropriate configuration.
- Use the [DROPPREFERENCES Procedure](#) to drop any existing index preferences
- Create new index preferences using the [CREATEPREFERENCES Procedure](#) procedure
- Create the CONTEXT index using the [CREATEINDEX Procedure](#) procedure
- Set up automatic synchronization of the index using the [CONFIGUREAUTOSYNC Procedure](#)

## Security Model

Owned by XDB, the DBMS\_XDBT package must be created by SYS or XDB. The EXECUTE privilege is granted to SYS or XDB. Subprograms in this package are executed using the privileges of the current user.

## Operational Notes

The DBMS\_XDBT package can be customized by using a PL/SQL procedure or an anonymous block to set the relevant package variables, configuration settings, and then execute the procedures. A more general approach would be to introduce the appropriate customizations by modifying this package in place, or as a copy. The system must be configured to use job queues, and the jobs can be viewed through the USER\_JOBS catalog views. This section describes the configuration settings, or package variables, available to customize the DBMS\_XDBT package.

**Table 167–1 General Indexing Settings for Customizing DBMS\_XDBT**

Parameter	Default Value	Description
IndexName	XDB\$CI	Name of the CONTEXT index.
IndexTablespace	XDB\$RESINFO	Tablespace used by tables and indexes comprising the CONTEXT index.
IndexMemory	128M	Memory used by index creation and SYNC; less than or equal to the MAX_INDEX_MEMORY system parameter (see the CTX_ADMIN package).
LogFile	'XdbCtxLog'	The log file used for ROWID during indexing. The LOG_DIRECTORY system parameter must be set already. NULL turns off ROWID logging.

**Table 167–2 Filtering Settings for Customizing DBMS\_XDBT**

Parameter	Default Value	Description
SkipFilter_Types	image/%, audio/%, video/%, model/%	List of mime types that should not be indexed.
NullFilter_Types	text/plain, text/html, text/xml	List of mime types that do not need to use the INSO filter. Use this for text-based documents.
FilterPref	XDB\$CI_FILTER	Name of the filter preference.

**Table 167–3 Stoplist Settings for Customizing DBMS\_XDBT**

Parameter	Default Value	Description
StoplistPref	XDB\$CI_STOPLIST	Name of the stoplist.
StopWords	0..9; 'a'..'z'; 'A'..'Z'	List of stopwords, in excess of CTXSYS.DEFAULT_STOPLIST.

**Table 167–4 Sectioning and Section Group Settings for Customizing DBMS\_XDBT**

Parameter	Default Value	Description
SectionGroup	HTML_SECTION_GROUP	Default sectioner. Use PATH_SECTION_GROUP or AUTO_SECTION_GROUP if repository contains mainly XML documents.
Sectiongroupref	XDB\$CI_SECTIONGROUP	Name of the section group.

**Table 167–5 Other Index Preference Settings for Customizing DBMS\_XDBT**

Parameter	Default Value	Description
DatastorePref	XDB\$CI_DATASTORE	Name of the datastore preference
StoragePref	XDB\$CI_STORAGE	Name of the storage preference.
WordlistPref	XDB\$CI_WORDLIST	Name of the wordlist preference.
DefaultLexerPreference	XDB\$CI_DEFAULT_LEXER	Name of the default lexer preference.

**Table 167–6 SYNC (CONTEXT Synchronization) Settings for Customizing DBMS\_XDBT**

Parameter	Default Value	Description
AutoSyncPolicy	SYNC_BY_PENDING_COUNT	Indicates when the index should be SYNCed. One of SYNC_BY_PENDING_COUNT, SYNC_BY_TIME, or SYNC_BY_PENDING_COUNT_AND_TIME.
MaxPendingCount	2	Maximum number of documents in the CTX_USER_PENDING queue before an index SYNC is triggered. Only if the AutoSyncPolicy is SYNC_BY_PENDING_COUNT or SYNC_BY_PENDING_COUNT_AND_TIME.
CheckPendingCountInterval	10 minutes	How often, in minutes, the pending queue should be checked. Only if the AutoSyncPolicy is SYNC_BY_PENDING_COUNT or SYNC_BY_PENDING_COUNT_AND_TIME.
SyncInterval	60 minutes	Indicates how often, in minutes, the index should be SYNCed. Only if the AutoSyncPolicy is SYNC_BY_TIME or SYNC_BY_PENDING_COUNT_AND_TIME.

---

## Summary of DBMS\_XDBT Subprograms

**Table 167–7 DBMS\_XDBT Package Subprograms**

Subprogram	Description
<a href="#">CONFIGUREAUTOSYNC Procedure</a> on page 167-8	Configures the CONTEXT index for automatic maintenance, SYNC
<a href="#">CREATEDATASTOREPREF Procedure</a> on page 167-9	Creates a USER datastore preference for the CONTEXT index
<a href="#">CREATEFILTERPREF Procedure</a> on page 167-10	Creates a filter preference for the CONTEXT index
<a href="#">CREATEINDEX Procedure</a> on page 167-11	Creates the CONTEXT index on the XML DB hierarchy
<a href="#">CREATELEXERPREF Procedure</a> on page 167-12	Creates a lexer preference for the CONTEXT index
<a href="#">CREATEPREFERENCES Procedure</a> on page 167-13	Creates preferences required for the CONTEXT index on the XML DB hierarchy
<a href="#">CREATESECTIONGROUPPREF Procedure</a> on page 167-14	Creates a storage preference for the CONTEXT index
<a href="#">CREATESTOPLISTPREF Procedure</a> on page 167-15	Creates a section group for the CONTEXT index
<a href="#">CREATESTORAGEPREF Procedure</a> on page 167-16	Creates a wordlist preference for the CONTEXT index
<a href="#">CREATEWORLDSLIPREF Procedure</a> on page 167-17	Creates a stoplist for the CONTEXT index
<a href="#">DROPPREFERENCES Procedure</a> on page 167-18	Drops any existing preferences

## CONFIGUREAUTOSYNC Procedure

This procedure sets up jobs for automatic SYNCs of the CONTEXT index.

### Syntax

```
DBMS_XDBT.CONFIGUREAUTOSYNC;
```

### Usage Notes

- The system must be configured for job queues for automatic synchronization. The jobs can be viewed using the USER\_JOBS catalog views
- The configuration parameter `AutoSyncPolicy` can be set to choose an appropriate synchronization policy.

The synchronization can be based on one of the following:

Sync Basis	Description
SYNC_BY_PENDING_COUNT	The SYNC is triggered when the number of documents in the pending queue is greater than a threshold (See the <a href="#">MaxPendingCount</a> configuration setting on page 167-6). The pending queue is polled at regular intervals (See the <a href="#">CheckPendingCountInterval</a> configuration parameter on page 167-6) to determine if the number of documents exceeds the threshold.
SYNC_BY_TIME	The SYNC is triggered at regular intervals. (See the <a href="#">SyncInterval</a> configuration parameter on page 167-6).
SYNC_BY_PENDING_COUNT_AND_TIME	A combination of both of the preceding options.

## CREATEDATASTOREPREF Procedure

This procedure creates a user datastore preference for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATEDATASTOREPREF;
```

### Usage Notes

- The name of the datastore preference can be modified; see the `DatastorePref` configuration setting.
- The default `USER` datastore procedure also filters the incoming document. The `DBMS_XDBT` package provides a set of configuration settings that control the filtering process.
- The `SkipFilter_Types` array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not indexed. Some of the properties of the document metadata, such as author, remain unindexed.
  - The `NullFilter_Types` array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not filtered; however, they are still indexed. This is intended to be used for documents that are text-based, such as HTML, XML and plain-text.
  - All other documents use the `INSO` filter through the `IFILTER` API.

## CREATEFILTERPREF Procedure

This procedure creates a `NULL` filter preference for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATEFILTERPREF;
```

### Usage Notes

- The name of the filter preference can be modified; see `FilterPref` configuration setting.
- The `USER` datastore procedure filters the incoming document; see [CREATEDATASTOREPREF Procedure](#) for more details.



## CREATEINDEX Procedure

This procedure creates the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATEINDEX;
```

### Usage Notes

- The name of the index can be changed; see the `IndexName` configuration setting.
- Set the `LogFile` configuration parameter to enable `ROWID` logging during index creation.
- Set the `IndexMemory` configuration parameter to determine the amount of memory that index creation, and later `SYNCs`, will use.

## CREATELEXERPREF Procedure

This procedure creates a BASIC `lexer` preference for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATELEXERPREF;
```

### Usage Notes

- The name of the `lexer` preference can be modified; see `LexerPref` configuration setting. No other configuration settings are provided.
- `MultiLexer` preferences are not supported.
- Base letter translation is turned on by default.

## **CREATEPREFERENCES Procedure**

This procedure creates a set of default preferences based on the configuration settings.

### **Syntax**

```
DBMS_XDBT.CREATEPREFERENCES;
```

## CREATESECTIONGROUPPREF Procedure

This procedure creates a section group for the CONTEXT index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATESECTIONGROUPPREF;
```

### Usage Notes

- The name of the section group can be changed; see the `SectiongroupPref` configuration setting.
- The HTML sectioner is used by default. No zone sections are created by default. If the vast majority of documents are XML, consider using the `AUTO_SECTION_GROUP` or the `PATH_SECTION_GROUP`; see the `SectionGroup` configuration setting.

## CREATESTOPLISTPREF Procedure

This procedure creates a stoplist for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATESTOPLISTPREF;
```

### Usage Notes

- The name of the stoplist can be modified; see the `StoplistPref` configuration setting.
- Numbers are not indexed.
- The `StopWords` array is a configurable list of stopwords. These are meant to be stopwords in addition to the set of stopwords in `CTXSYS.DEFAULT_STOPLIST`.

## CREATESTORAGEPREF Procedure

This procedure creates a `BASIC_STORAGE` preference for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATESTORAGEPREF;
```

### Usage Notes

- The name of the storage preference can be modified; see the `StoragePref` configuration setting.
- A tablespace can be specified for the tables and indexes comprising the `CONTEXT` index; see the `IndexTablespace` configuration setting.
- Prefix and Substring indexing are not turned on by default.
- The `I_INDEX_CLAUSE` uses key compression.

## CREATEWORLDPREF Procedure

This procedure creates a word list preference for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XDBT.CREATEWORLDPREF;
```

### Usage Notes

- The name of the word list preference can be modified; see the `WordlistPref` configuration setting. No other configuration settings are provided.
- `FUZZY_MATCH` and `STEMMER` attributes are set to `AUTO` (auto-language detection)

## DROPPREFERENCES Procedure

This procedure drops any previously created preferences for the `CONTEXT` index on the XML DB hierarchy.

### Syntax

```
DBMS_XMLDB.DROPPREFERENCES;
```



The DBMS\_XDBZ package controls the Oracle XML DB repository security, which is based on Access Control Lists (ACLs).

This chapter contains the following topics:

- [Using DBMS\\_XDBZ](#)
  - Security Model
  - Constants
- [Summary of DBMS\\_XDBZ Subprograms](#)

**See Also:** *Oracle XML DB Developer's Guide*

---

## Using DBMS\_XDBZ

This section contains topics which relate to using the DBMS\_XDBZ package.

- [Security Model](#)
- [Constants](#)

## Security Model

Owned by XDB, the DBMS\_XDBZ package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Constants

The DBMS\_XDBZ package uses the constants shown in following tables.

- [DBMS\\_XDBZ Constants - Name Format](#) on page 168-4
- [DBMS\\_XDBZ Constants - Enable Option](#) on page 168-4
- [DBMS\\_XDBZ Constants - Enable Option Exercised](#) on page 168-4

**Table 168–1 DBMS\_XDBZ Constants - Name Format**

Constant	Type	Value	Description
NAME_FORMAT_SHORT	PLS_INTEGER	1	DB user name or LDAP nickname
NAME_FORMAT_DISTINGUISHED	PLS_INTEGER	2	LDAP distinguished name

**Table 168–2 DBMS\_XDBZ Constants - Enable Option**

Constant	Type	Value	Description
ENABLE_CONTENTS	PLS_INTEGER	1	Enables hierarchy for contents and is used by users when calling the <a href="#">ENABLE_HIERARCHY Procedure</a>
ENABLE_RESMETADATA	PLS_INTEGER	2	Enables hierarchy for resource metadata, that is, this table will store schema based custom metadata for resources

**Table 168–3 DBMS\_XDBZ Constants - Enable Option Exercised**

Constant	Type	Value	Description
IS_ENABLED_CONTENTS	PLS_INTEGER	1	If hierarchy was enabled for contents, that is, the <a href="#">ENABLE_HIERARCHY Procedure</a> was called with <code>hierarchy_type</code> as <code>ENABLE_CONTENTS</code>
IS_ENABLED_RESMETADATA	PLS_INTEGER	2	If hierarchy was enabled for resource metadata, that is, the <a href="#">ENABLE_HIERARCHY Procedure</a> was called with <code>hierarchy_type</code> as <code>ENABLE_RESMETADATA</code>

---

## Summary of DBMS\_XDBZ Subprograms

**Table 168–4** *DBMS\_XDBZ Package Subprograms*

Method	Description
<a href="#">DISABLE_HIERARCHY Procedure</a> on page 168-6	Disables repository support for the specified XMLTYPE table or view
<a href="#">ENABLE_HIERARCHY Procedure</a> on page 168-7	Enables repository support for the specified XMLType table or view
<a href="#">GET_ACLOID Function</a> on page 168-8	Retrieves the ACL Object ID for the specified resource
<a href="#">GET_USERID Function</a> on page 168-9	Retrieves the user ID for the specified user
<a href="#">IS_HIERARCHY_ENABLED Function</a> on page 168-10	Determines if repository support for the specified XMLType table or view is enabled
<a href="#">PURGELDAPCACHE Function</a> on page 168-11	Purges the LDAP nickname cache

## DISABLE\_HIERARCHY Procedure

This procedure disables repository support for a particular XMLType table or view.

### Syntax

```
DBMS_XDBZ.DISABLE_HIERARCHY(  
    object_schema IN VARCHAR2,  
    object_name   IN VARCHAR2);
```

### Parameters

**Table 168–5** *DISABLE\_HIERARCHY Procedure Parameters*

Parameter	Description
object_schema	The schema name of the XMLType table or view
object_name	The name of the XMLType table or view

## ENABLE\_HIERARCHY Procedure

This procedure enables repository support for a particular XMLType table or view. This allows the use of a uniform ACL-based security model across all documents in the repository.

**See Also:** *Oracle XML DB Developer's Guide* for more information about

### Syntax

```
DBMS_XDBZ.ENABLE_HIERARCHY (
  object_schema IN VARCHAR2,
  object_name   IN VARCHAR2,
  hierarchy_type IN PLS_INTEGER := DBMS_XDBZ.ENABLE_CONTENTS);
```

### Parameters

**Table 168–6** *ENABLE\_HIERARCHY Procedure Parameters*

Parameter	Description
object_schema	The schema name of the XMLType table or view
object_name	The name of the XMLType table or view
hierarchy_type	How to enable the hierarchy. <ul style="list-style-type: none"> <li>▪ ENABLE_CONTENTS - enable hierarchy for contents, that is, this table will store contents of resources in the repository</li> <li>▪ ENABLE_RESMETADATA - enable hierarchy for resource metadata, that is, this table will store schema based custom metadata for resources</li> </ul> <p>If this subprogram is called on a table, another call will have no effect. Note that you cannot enable hierarchy for both contents and resource metadata.</p>

## GET\_ACLOID Function

This function retrieves the ACL Object ID for the specified resource, if the repository path is known.

### Syntax

```
DBMS_XDBZ.GET_ACLOID(  
    aclpath IN VARCHAR2,  
    acloid  OUT RAW)  
RETURN BOOLEAN;
```

### Parameters

**Table 168–7** GET\_ACLOID Function Parameters

Parameter	Description
aclpath	ACL resource path for the repository
acloid	The returned Object ID

### Return Values

Returns TRUE if successful.



## GET\_USERID Function

This function retrieves the user ID for the specified user name. The local database is searched first, and if found, the `USERID` is returned in 4-byte database format. Otherwise, the LDAP directory is searched, if available, and if found, the `USERID` is returned in 4-byte database format.

### Syntax

```
DBMS_XDBZ.GET_USERID(
  username IN VARCHAR2,
  userid   OUT RAW,
  format   IN BINARY_INTEGER := NAME_FORMAT_SHORT)
RETURN BOOLEAN;
```

### Parameters

**Table 168–8** *GET\_USERID Function Parameters*

Parameter	Description
<code>username</code>	Name of the database or LDAP user.
<code>userid</code>	Return parameter for the matching user id.
<code>format</code>	Format of the specified user name; valid options are: <ul style="list-style-type: none"> <li>▪ <code>DBMS_XDBZ.NAME_FORMAT_SHORT</code> (default) -- DB user name or LDAP nickname</li> <li>▪ <code>DBMS_XDBZ.NAME_FORMAT_DISTINGUISHED</code> -- LDAP distinguished name.</li> </ul>

### Return Values

Returns `TRUE` if successful.

## IS\_HIERARCHY\_ENABLED Function

This function determines if repository support for the specified XMLType table or view is enabled.

### Syntax

```
DBMS_XDBZ.IS_HIERARCHY_ENABLED(  
  object_schema IN VARCHAR2,  
  object_name   IN VARCHAR2,  
  hierarchy_type IN PLS_INTEGER := IS_ENABLED_CONTENTS)  
RETURN BOOLEAN;
```

### Parameters

**Table 168–9 IS\_HIERARCHY\_ENABLED Function Parameters**

Parameter	Description
object_schema	The schema name of the XMLType table or view
object_name	The name of the XMLType table or view
hierarchy_type	The type of hierarchy to check for. <ul style="list-style-type: none"><li>■ IS_ENABLED_CONTENTS - if hierarchy was enabled for contents, that is, the <a href="#">ENABLE_HIERARCHY Procedure</a> was called with <code>hierarchy_type</code> as <code>ENABLE_CONTENTS</code></li><li>■ IS_ENABLED_RESMETADATA - if hierarchy was enabled for resource metadata, that is, the <a href="#">ENABLE_HIERARCHY Procedure</a> was called with <code>hierarchy_type</code> as <code>ENABLE_RESMETADATA</code></li></ul>

### Return Values

Returns `TRUE` if the given XMLTYPE table or view has the XDB Hierarchy enabled with the specified type.

## PURGELDAPCACHE Function

This function purges the LDAP nickname cache. Returns `TRUE` if successful.

### Syntax

```
DBMS_XDBZ.PURGELDAPCACHE  
RETURN BOOLEAN;
```



The DBMS\_XEVENT package provides event-related types and supporting subprograms.

**See Also:** *Oracle XML DB Developer's Guide* for more information about "Oracle XML DB Repository Events"

This chapter contains the following topics:

- [Using DBMS\\_XEVENT](#)
  - Security Model
  - Constants
- [Subprogram Groups](#)
  - XDBEvent Type Subprograms
  - XDBRepositoryEvent Type Subprograms
  - XDBHandlerList Type Subprograms
  - XDBHandler Type Subprograms
  - XDBPath Type Subprograms
  - XDBLink Type Subprograms
- [Summary of DBMS\\_XEVENT Subprograms](#)

---

## Using DBMS\_XEVENT

- [Security Model](#)
- [Constants](#)

## Security Model

Owned by XDB, the DBMS\_XEVENT package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Constants

The DBMS\_XEVENT package uses the constants shown in [Table 169-1](#):

**Table 169-1 DBMS\_XEVENT Constants**

Name	Type	Value	Description
RENDER_EVENT	PLS_INTEGER	1	
PRE_CREATE_EVENT	PLS_INTEGER	2	
POST_CREATE_EVENT	PLS_INTEGER	3	
PRE_DELETE_EVENT	PLS_INTEGER	4	
POST_DELETE_EVENT	PLS_INTEGER	5	
PRE_UPDATE_EVENT	PLS_INTEGER	6	
POST_UPDATE_EVENT	PLS_INTEGER	7	
PRE_LOCK_EVENT	PLS_INTEGER	8	
POST_LOCK_EVENT	PLS_INTEGER	9	
PRE_UNLOCK_EVENT	PLS_INTEGER	10	
POST_UNLOCK_EVENT	PLS_INTEGER	11	
PRE_LINKIN_EVENT	PLS_INTEGER	12	
POST_LINKIN_EVENT	PLS_INTEGER	13	
PRE_LINKTO_EVENT	PLS_INTEGER	14	
POST_LINKTO_EVENT	PLS_INTEGER	15	
PRE_UNLINKIN_EVENT	PLS_INTEGER	16	
POST_UNLINKIN_EVENT	PLS_INTEGER	17	
PRE_UNLINKFROM_EVENT	PLS_INTEGER	18	
POST_UNLINKFROM_EVENT	PLS_INTEGER	19	
PRE_CHECKIN_EVENT	PLS_INTEGER	20	
POST_CHECKIN_EVENT	PLS_INTEGER	21	
PRE_CHECKOUT_EVENT	PLS_INTEGER	22	
POST_CHECKOUT_EVENT	PLS_INTEGER	23	
PRE_UNCHECKOUT_EVENT	PLS_INTEGER	24	
POST_UNCHECKOUT_EVENT	PLS_INTEGER	25	
PRE_VERSIONCONTROL_EVENT	PLS_INTEGER	26	
POST_VERSIONCONTROL_EVENT	PLS_INTEGER	27	
PRE_OPEN_EVENT	PLS_INTEGER	28	
POST_OPEN_EVENT	PLS_INTEGER	29	
PRE_INCONSISTENT_UPDATE_EVENT	PLS_INTEGER	30	
POST_INCONSISTENT_UPDATE_EVENT	PLS_INTEGER	21	
POST_CHECKIN_EVENT	PLS_INTEGER	21	



## Subprogram Groups

- [XDBEvent Type Subprograms](#)
- [XDBRepositoryEvent Type Subprograms](#)
- [XDBHandlerList Type Subprograms](#)
- [XDBHandler Type Subprograms](#)
- [XDBPath Type Subprograms](#)
- [XDBLink Type Subprograms](#)

## XDBEvent Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBEvent type.

**Table 169–2 XDBEvent Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GETCURRENTUSER Function</a> on page 169-18	Returns the name of the user executing the operation that triggers the event
<a href="#">GETEVENT Function</a> on page 169-19	Returns a value identifying the triggering event
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL

The [Summary of DBMS\\_XEVENT Subprograms](#) contains a complete listing of all subprograms in the package.

## XDBRepositoryEvent Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBRepositoryEvent type.

**Table 169–3 XDBRepositoryEvent Subprograms**

Subprogram	Description
<a href="#">GETAPPLICATIONDATA Function</a> on page 169-16	Returns the <applicationData> element extracted from the resource configuration that defines the invoking handler
<a href="#">GETHANDLERLIST Function</a> on page 169-21	Returns an XDBHandlerList object containing the list of handlers that will be executed after the currently executing handler
<a href="#">GETINTERFACE Function</a> on page 169-22	Returns the top-level interface used to initiate the operation that triggered the event
<a href="#">GETLINK Function</a> on page 169-24	Returns an XDBLink object for the target resource
<a href="#">GETLOCK Function</a> on page 169-26	Returns the lock object corresponding to the current operation
<a href="#">GETOLDRESOURCE Function</a> on page 169-30	Returns the original XDBResource object before the operation was executed
<a href="#">GETOPENACCESSMODE Function</a> on page 169-31	Returns the access mode for the open operation
<a href="#">GETOPENDENYMODE Function</a> on page 169-32	Returns the deny mode for the open operation
<a href="#">GETOUTPUTSTREAM Function</a> on page 169-33	Returns the output BLOB in which the handler can write the rendered data
<a href="#">GETPARAMETER Function</a> on page 169-34	Returns the value of a request or session-specific parameter
<a href="#">GETPARENT Function</a> on page 169-35	Returns the resource object corresponding to a parent folder of the target resource
<a href="#">GETPATH Function</a> on page 169-39	Returns the XDBPath object representing the path of the resource for which the event was fired
<a href="#">GETRESOURCE Function</a> on page 169-40	Returns an XDBResource object that provides methods to access and modify the contents and metadata of the target resource
<a href="#">GETUPDATEBYTECOUNT Function</a> on page 169-43	If the current operation is a byte-range write, returns the byte count
<a href="#">GETUPDATEBYTEOFFSET Function</a> on page 169-44	If the current operation is a byte-range write, function returns the byte offset at which the range begins
<a href="#">GETXDBEVENT Function</a> on page 169-45	Converts an XDBRepositoryEvent object to an XDBEvent type
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL
<a href="#">SETRENDERPATH Procedure</a> on page 169-49	Specifies the path of the resource that contains the rendered contents
<a href="#">SETRENDERSTREAM Procedure</a> on page 169-50	sets the BLOB from which the rendered contents can be read

The [Summary of DBMS\\_XEVENT Subprograms](#) contains a complete listing of all subprograms in the package.

## XDBHandlerList Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBHandlerList type.

**Table 169–4 XDBHandlerList Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CLEAR Procedure</a> on page 169-15	Clears the handler list
<a href="#">GETFIRST Function</a> on page 169-20	Returns the first handler in the list
<a href="#">GETNAME Function</a> on page 169-28	Returns the next handler in the list
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL
<a href="#">REMOVE Procedure</a> on page 169-48	Removes the specified handler from the handler list

The [Summary of DBMS\\_XEVENT Subprograms](#) contains a complete listing of all subprograms in the package.

## XDBHandler Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBHandler type.

**Table 169–5 XDBHandler Type Subprograms**

Subprogram	Description
<a href="#">GETLANGUAGE Function</a> on page 169-23	Returns the implementation language of the handler
<a href="#">GETSCHEMA Function</a> on page 169-41	Returns the schema of the handler's source
<a href="#">GETSOURCE Function</a> on page 169-42	Returns the name of the Java class, PL/SQL package or object type implementing the handler
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL

The [Summary of DBMS\\_XEVENT Subprograms](#) contains a complete listing of all subprograms in the package.

## XDBPath Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBPath type.

**Table 169–6** XDBPath Type Subprograms

Subprogram	Description
<a href="#">GETNAME Function</a> on page 169-28	Returns the string representation of the path
<a href="#">GETPARENTPATH Function</a> on page 169-38	Returns the parent's path
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL

The [Summary of DBMS\\_XEVENT Subprograms](#) contains a complete listing of all subprograms in the package.

## XDBLink Type Subprograms

This subprogram group provides an interface for use in conjunction with the XDBLink type.

**Table 169–7 XDBLink Type Subprograms**

Subprogram	Description
<a href="#">GETCHILDROID Function</a> on page 169-17	Returns the OID of the resource to which the link is pointing
<a href="#">GETPARENTNAME Function</a> on page 169-36	Returns the link's parent folder's name
<a href="#">GETPARENTOID Function</a> on page 169-37	Returns the link's parent folder's OID
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL

The [Summary of DBMS\\_XEVENT Subprograms](#) contains a complete listing of all subprograms in the package.

## Summary of DBMS\_XEVENT Subprograms

**Table 169–8 DBMS\_XEVENT Package Subprograms**

Subprogram	Description	Group
<a href="#">CLEAR Procedure</a> on page 169-15	Clears the handler list	<a href="#">XDBHandlerList Type Subprograms</a> on page 169-8
<a href="#">GETAPPLICATIONDATA Function</a> on page 169-16	Returns the <applicationData> element extracted from the resource configuration that defines the invoking handler	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-7
<a href="#">GETCHILDROID Function</a> on page 169-17	Returns the OID of the resource to which the link is pointing	<a href="#">XDBLink Type Subprograms</a> on page 169-11
<a href="#">GETCURRENTUSER Function</a> on page 169-18	Returns the name of the user executing the operation that triggers the event	<a href="#">XDBEvent Type Subprograms</a> on page 169-6
<a href="#">GETEVENT Function</a> on page 169-19	Returns a value identifying the triggering event	<a href="#">XDBEvent Type Subprograms</a> on page 169-6
<a href="#">GETFIRST Function</a> on page 169-20	Returns the first handler in the list	<a href="#">XDBHandlerList Type Subprograms</a> on page 169-8
<a href="#">GETHANDLERLIST Function</a> on page 169-21	Returns an <code>XDBHandlerList</code> object containing the list of handlers that will be executed after the currently executing handler	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-7
<a href="#">GETINTERFACE Function</a> on page 169-22	Returns the top-level interface used to initiate the operation that triggered the event	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-7
<a href="#">GETLANGUAGE Function</a> on page 169-23	Returns the implementation language of the handler	<a href="#">XDBHandler Type Subprograms</a> on page 169-9
<a href="#">GETLINK Function</a> on page 169-24	Returns an <code>XDBLink</code> object for the target resource	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-7
<a href="#">GETLOCK Function</a> on page 169-26	Returns the lock object corresponding to the current operation	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETNAME Function</a> on page 169-28	Returns the string representation of the path	<a href="#">XDBPath Type Subprograms</a> on page 169-10
<a href="#">GETNAME Function</a> on page 169-28	Returns the next handler in the list	<a href="#">XDBHandlerList Type Subprograms</a> on page 169-8
<a href="#">GETOLDRESOURCE Function</a> on page 169-30	Returns the original <code>XDBResource</code> object before the operation was executed	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETOPENACCESSMODE Function</a> on page 169-31	Returns the access mode for the open operation	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6



**Table 169–8 (Cont.) DBMS\_XEVENT Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">GETOPENDENYMODE Function</a> on page 169-32	Returns the deny mode for the open operation	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETOUTPUTSTREAM Function</a> on page 169-33	Returns the output BLOB in which the handler can write the rendered data	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETPARAMETER Function</a> on page 169-34	Returns the value of a request or session-specific parameter	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETPARENT Function</a> on page 169-35	Returns the resource object corresponding to a parent folder of the target resource	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETPARENTNAME Function</a> on page 169-36	Returns the link's parent folder's name	<a href="#">XDBLink Type Subprograms</a> on page 169-11
<a href="#">GETPARENTOID Function</a> on page 169-37	Returns the link's parent folder's OID	<a href="#">XDBLink Type Subprograms</a> on page 169-11
<a href="#">GETPARENTNAME Function</a> on page 169-36	Returns the parent's path	<a href="#">XDBPath Type Subprograms</a> on page 169-10
<a href="#">GETPATH Function</a> on page 169-39	Returns the <code>XDBPath</code> object representing the path of the resource for which the event was fired	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETRESOURCE Function</a> on page 169-40	Returns an <code>XDBResource</code> object that provides methods to access and modify the contents and metadata of the target resource	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETSCHEMA Function</a> on page 169-41	Returns the schema of the handler's source	<a href="#">XDBHandler Type Subprograms</a> on page 169-9
<a href="#">GETSOURCE Function</a> on page 169-42	Returns the name of the Java class, PL/SQL package or object type implementing the handler	<a href="#">XDBHandler Type Subprograms</a> on page 169-9
<a href="#">GETUPDATEBYTECOUNT Function</a> on page 169-43	If the current operation is a byte-range write, returns the byte count	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETUPDATEBYTEOFFSET Function</a> on page 169-44	If the current operation is a byte-range write, function returns the byte offset at which the range begins	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">GETXDBEVENT Function</a> on page 169-45	Converts an <code>XDBRepositoryEvent</code> object to an <code>XDBEvent</code> type	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6

**Table 169–8 (Cont.) DBMS\_XEVENT Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">ISNULL Functions</a> on page 169-46	Returns TRUE if input argument is NULL	<a href="#">XDBEvent Type Subprograms</a> on page 169-6 <a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-7 <a href="#">XDBHandlerList Type Subprograms</a> on page 169-8 <a href="#">XDBHandler Type Subprograms</a> on page 169-9 <a href="#">XDBPath Type Subprograms</a> on page 169-10
<a href="#">REMOVE Procedure</a> on page 169-48	Removes the specified handler from the handler list	<a href="#">XDBHandlerList Type Subprograms</a> on page 169-8
<a href="#">SETRENDERPATH Procedure</a> on page 169-49	Specifies the path of the resource that contains the rendered contents	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6
<a href="#">SETRENDERSTREAM Procedure</a> on page 169-50	sets the BLOB from which the rendered contents can be read	<a href="#">XDBRepositoryEvent Type Subprograms</a> on page 169-6

## CLEAR Procedure

this procedure clears the handler list.

**See Also:** [XDBHandlerList Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.CLEAR (  
    hl          IN OUT  XDBHandlerList);
```

### Parameters

**Table 169–9** *CLEAR Procedure Parameters*

Parameter	Description
hl	Handler list

## GETAPPLICATIONDATA Function

This function returns the <applicationData> element extracted from the resource configuration that defines the invoking handler.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETAPPLICATIONDATA (  
    ev IN XDBRepositoryEvent)  
    RETURN XMLType;
```

### Parameters

**Table 169–10** *GETAPPLICATIONDATA Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETCHILDROID Function

This function returns the OID of the resource to which the link is pointing.

**See Also:** [XDBLink Type Subprograms](#) on page 169-11 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETCHILDROID (  
    link IN XDBLink)  
RETURN RAW;
```

### Parameters

**Table 169–11** *GETCHILDROID Function Parameters*

Parameter	Description
link	Link

## GETCURRENTUSER Function

This function returns the name of the user executing the operation that triggers the event.

**See Also:** [XDBEvent Type Subprograms](#) on page 169-6 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETCURRENTUSER (  
    ev IN XDBEvent)  
    RETURN VARCHAR2;
```

### Parameters

**Table 169–12** *GETCURRENTUSER Function Parameters*

Parameter	Description
ev	Event of XDBEvent type

## GETEVENT Function

This function returns the name of the user executing the operation that triggers the event.

**See Also:** [XDBEvent Type Subprograms](#) on page 169-6 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETEVENT (  
    ev IN XDBEvent)  
RETURN XDBEventID;
```

### Parameters

**Table 169–13** *GETEVENT Function Parameters*

Parameter	Description
ev	Event of XDBEvent type

## GETFIRST Function

This function returns the first handler in the list.

**See Also:** [XDBHandlerList Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETFIRST (  
    hl IN XDBHandlerList)  
RETURN XDBHandler;
```

### Parameters

**Table 169–14** *GETFIRST Function Parameters*

Parameter	Description
hl	Handler list



## GETHANDLERLIST Function

This function returns an `XDBHandlerList` object containing the list of handlers that will be executed after the currently executing handler. The current handler can then filter out some of the subsequent handlers if necessary, subject to security checks. An insufficient privilege exception is raised if the executing user does not have the required access privilege to any of the resource configuration associating with a handler in the list.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETHANDLERLIST (  
    ev IN XDBRepositoryEvent)  
RETURN XDBHandlerList;
```

### Parameters

**Table 169–15** *GETHANDLERLIST Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETINTERFACE Function

This function returns the top-level interface used to initiate the operation that triggered the event. This could be HTTP, FTP or SQL.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETINTERFACE (  
    ev IN XDBRepositoryEvent)  
    RETURN VARCHAR2;
```

### Parameters

**Table 169–16** *GETINTERFACE Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETLANGUAGE Function

This function returns the implementation language of the handler.

**See Also:** [XDBHandler Type Subprograms](#) on page 169-9 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETLANGUAGE (  
    handler IN XDBHandler)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–17** GETLANGUAGE Function Parameters

Parameter	Description
handler	Handler

## GETLINK Function

This function returns an `XDBLink` object for the target resource. For a `link*` or `unlink*` event, this will be the link involved in the operation. For other events, an error is returned. Using this object the handler can access link properties, such as, `ParentName`, `ParentOID`, `ChildOID` and `LinkName`.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETLINK (  
    ev IN XDBRepositoryEvent)  
    RETURN XDBLink;
```

### Parameters

**Table 169–18** *GETLINK Function Parameters*

Parameter	Description
ev	Event of <code>XDBRepositoryEvent</code> type

## GETLINKNAME Function

This function returns the name of the link.

**See Also:** [XDBLink Type Subprograms](#) on page 169-11 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETLINKNAME (  
    link IN XDBLink)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–19** GETLINKNAME Function Parameters

Parameter	Description
link	Link

## GETLOCK Function

This function returns the lock object corresponding to the current operation. It is only valid for lock and unlock events.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETLOCK (  
    ev IN XDBRepositoryEvent)  
    RETURN XDBLock;
```

### Parameters

**Table 169–20** *GETLOCK Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETLANGUAGE Function

This function returns the implementation language of the handler.

**See Also:** [XDBHandler Type Subprograms](#) on page 169-9 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETLANGUAGE (  
    handler IN XDBHandler)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–21** GETLANGUAGE Function Parameters

Parameter	Description
handler	Handler

## GETNAME Function

This function returns the string representation of the path.

**See Also:** [XDBPath Type Subprograms](#) on page 169-10 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETNAME (  
    path IN XDBPath)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–22** *GETNAME Function Parameters*

Parameter	Description
path	Path



## GETNEXT Function

This function returns the next handler in the list.

**See Also:** [XDBHandlerList Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETNEXT (  
    hl IN XDBHandlerList)  
RETURN XDBHandler;
```

### Parameters

**Table 169–23** GETNEXT Function Parameters

Parameter	Description
hl	Handler list

## GETOLDRESOURCE Function

This function returns the original `XDBResource` object before the operation was executed. This method applies only to update event. For other events, an error is returned. This is a read-only object, and consequently none of the modifier methods will work on this object.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETOLDRESOURCE (  
    ev IN XDBRepositoryEvent)  
RETURN XDBResource;
```

### Parameters

**Table 169–24** *GETOLDRESOURCE Function Parameters*

Parameter	Description
ev	Event of <code>XDBRepositoryEvent</code> type

## GETOPENACCESSMODE Function

This function returns the access mode for the open operation.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETOPENACCESSMODE (  
    ev IN XDBRepositoryEvent)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 169–25** *GETOPENACCESSMODE Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

### Return Values

- XDBRepositoryEvent.OPEN\_ACCESS\_READ (value 1)
- XDBRepositoryEvent.OPEN\_ACCESS\_WRITE (value 2)
- XDBRepositoryEvent.OPEN\_ACCESS\_READ\_WRITE (value 3)

## GETOPENENYMODE Function

This function returns the deny mode for the open operation. It is only valid for the open event.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETOPENENYMODE (  
    ev IN XDBRepositoryEvent)  
    RETURN PLS_INTEGER;
```

### Parameters

**Table 169–26** *GETOPENENYMODE Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

### Return Values

- XDBRepositoryEvent.OPEN\_DENY\_NONE (value 0)
- XDBRepositoryEvent.OPEN\_DENY\_READ (value 1)
- XDBRepositoryEvent.OPEN\_DENY\_READ\_WRITE (value 2)

## GETOUTPUTSTREAM Function

This function returns the output BLOB in which the handler can write the rendered data. It is only valid for the render event.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETOUTPUTSTREAM (  
    ev IN XDBRepositoryEvent)  
RETURN BLOB;
```

### Parameters

**Table 169–27** *GETOUTPUTSTREAM Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETPARAMETER Function

This function returns the value of a request or session-specific parameter. The definition of the `key` parameter can be found in RFC 2616 (HTTP/1.1). They will be mapped to equivalent SQL session parameters (if any).

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETPARAMETER (  
    ev    IN    XDBRepositoryEvent,  
    key   IN    VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–28** *GETPARAMETER Function Parameters*

Parameter	Description
<code>ev</code>	Event of <code>XDBRepositoryEvent</code> type
<code>key</code>	Supported parameters: <ul style="list-style-type: none"><li>▪ <code>ACCEPT</code></li><li>▪ <code>ACCEPT-LANGUAGE</code></li><li>▪ <code>ACCEPT-CHARSET</code></li><li>▪ <code>ACCEPT_ENCODING</code></li></ul>

## GETPARENT Function

This function returns the resource object corresponding to a parent folder of the target resource. Note that this could be any folder that contains a link to the target resource. This is a read-only object, and consequently none of the modifier methods will work on this object. For a link\* or unlink\* event, this method returns the link's parent folder.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETPARENT (  
    ev IN XDBRepositoryEvent)  
RETURN XDBResource;
```

### Parameters

**Table 169–29** GETPARENT Function Parameters

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETPARENTNAME Function

This function returns the link's parent folder's name.

**See Also:** [XDBLink Type Subprograms](#) on page 169-11 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETPARENTNAME (  
    link IN XDBLink)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–30** GETPARENTNAME Function Parameters

Parameter	Description
link	Link



## GETPARENTOID Function

This function returns the link's parent folder's OID.

**See Also:** [XDBLink Type Subprograms](#) on page 169-11 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETPARENTOID (  
    link IN XDBLink)  
RETURN RAW;
```

### Parameters

**Table 169–31** *GETPARENTOID Function Parameters*

Parameter	Description
link	Link

## GETPARENTPATH Function

This function returns the parent's path. The level indicates the number of levels up the hierarchy. This value must be greater than zero. Level 1 means the immediate parent. If level exceeds the height of the tree then a NULL is returned.

**See Also:** [XDBPath Type Subprograms](#) on page 169-10 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETPARENTPATH (  
    path    IN    XDBPath,  
    level   IN    INTEGER)  
RETURN XDBPath;
```

### Parameters

**Table 169–32** *GETPARENTPATH Function Parameters*

Parameter	Description
path	Path
level	Number of levels up the hierarchy

## GETPATH Function

This function returns the `XDBPath` object representing the path of the resource for which the event was fired. From this object, functions are provided to get the different path segments.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETPATH (  
    ev IN XDBRepositoryEvent)  
RETURN XDBPath;
```

### Parameters

**Table 169–33** *GETPATH Function Parameters*

Parameter	Description
ev	Event of <code>XDBRepositoryEvent</code> type

## GETRESOURCE Function

This function returns an `XDBResource` object that provides methods to access and modify the contents and metadata of the target resource. This object reflects any changes made by previous handlers to the resource.

The modifier methods will work only in the pre-create and pre-update event handlers. For a `link*` or `unlink*` event, this method returns the resource that the link is pointing to. For a create event, this method returns the resource that is being created.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETRESOURCE (
    ev IN XDBRepositoryEvent)
RETURN XDBResource;
```

### Parameters

**Table 169–34** *GETRESOURCE Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETSCHEMA Function

This function returns the schema of the handler's source.

**See Also:** [XDBHandler Type Subprograms](#) on page 169-9 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETSCHEMA (  
    handler IN XDBHandler)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–35** GETSCHEMA Function Parameters

Parameter	Description
handler	Handler

## GETSOURCE Function

This function returns the name of the Java class, PL/SQL package or object type implementing the handler.

**See Also:** [XDBHandler Type Subprograms](#) on page 169-9 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETSOURCE (  
    handler IN XDBHandler)  
RETURN VARCHAR2;
```

### Parameters

**Table 169–36** *GETSOURCE Function Parameters*

Parameter	Description
handler	Handler

## GETUPDATEBYTECOUNT Function

If the current operation is a byte-range write, the function returns the byte count. It is only valid for the inconsistent-update event.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETUPDATEBYTECOUNT (  
    ev IN XDBRepositoryEvent)  
RETURN NUMBER;
```

### Parameters

**Table 169–37** *GETUPDATEBYTECOUNT Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type

## GETUPDATEBYTEOFFSET Function

If the current operation is a byte-range write, function returns the byte offset at which the range begins. It is only valid for the inconsistent-update event.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETUPDATEBYTEOFFSET (  
    ev IN XDBRepositoryEvent)  
RETURN NUMBER;
```

### Parameters

**Table 169–38** *GETUPDATEBYTEOFFSET Function Parameters*

Parameter	Description
ev	Event of XDBRepositoryEvent type



## GETXDBEVENT Function

This function converts an `XDBRepositoryEvent` object to an `XDBEvent` type.

**See Also:** [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.GETXDBEVENT (
    ev IN XDBRepositoryEvent)
RETURN XDBEvent;
```

### Parameters

**Table 169–39** *GETXDBEVENT Function Parameters*

Parameter	Description
ev	Event of <code>XDBRepositoryEvent</code> type

## ISNULL Functions

This function returns TRUE if input argument is NULL.

### See Also:

- [XDBEvent Type Subprograms](#) on page 169-6 for other subprograms in this group
- [XDBRepositoryEvent Type Subprograms](#) on page 169-7 for other subprograms in this group
- [XDBHandlerList Type Subprograms](#) on page 169-8 for other subprograms in this group
- [XDBHandler Type Subprograms](#) on page 169-9 for other subprograms in this group
- [XDBPath Type Subprograms](#) on page 169-10 for other subprograms in this group
- [XDBLink Type Subprograms](#) on page 169-11 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.ISNULL (
  ev          IN  XDBEvent)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
  ev          IN  XDBRepositoryEvent)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
  hl          IN  XDBHandlerList)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
  handler     IN  XDBHandler)
RETURN BOOLEAN;
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
  path       IN  XDBPath)
RETURN BOOLEAN;

DBMS_XEVENT.ISNULL (
  link       IN  XDBLink)
RETURN BOOLEAN;
```

### Parameters

**Table 169–40 ISNULL Function Parameters**

Parameter	Description
ev	Event of specified type
hl	Handler list
handler	Handler

**Table 169-40 (Cont.) ISNULL Function Parameters**

<b>Parameter</b>	<b>Description</b>
path	Path

## REMOVE Procedure

This procedure removes the specified handler from the handler list.

**See Also:** [XDBHandlerList Type Subprograms](#) on page 169-7 for other subprograms in this group

### Syntax

```
DBMS_XEVENT.REMOVE (  
    hl          IN OUT  XDBHandlerList,  
    handler    IN      XDBHandler);
```

### Parameters

**Table 169–41 REMOVE Procedure Parameters**

Parameter	Description
hl	Handler list
handler	Handler

## SETRENDERPATH Procedure

This procedure specifies the path of the resource that contains the rendered contents. This should not be called after the stream returned by [GETOUTPUTSTREAM Function](#) is written to or after the [SETRENDERSTREAM Procedure](#) is called; doing so will result in an error. This is only valid for the render event.

### Syntax

```
DBMS_XEVENT.SETRENDERPATH (  
    ev      IN    XDBRepositoryEvent,  
    path   IN    VARCHAR2);
```

### Parameters

**Table 169–42** *SETRENDERPATH Procedure Parameters*

Parameter	Description
ev	XDB Repository Event object
path	Path of the resource containing the rendered contents

## SETRENDERSTREAM Procedure

This procedure sets the BLOB from which the rendered contents can be read. This should not be called after the stream returned by GETOUTPUTSTREAM is written to or after SETRENDERPATH is called; doing so will result in an error. This is only valid for the render event.

### Syntax

```
DBMS_XEVENT.SETRENDERSTREAM (  
    ev    IN    XDBRepositoryEvent,  
    istr  IN    BLOB);
```

### Parameters

**Table 169–43** SETRENDERSTREAM Procedure Parameters

Parameter	Description
ev	XDBRepositoryEvent object
istr	Input stream from which to get the rendered contents

The DBMS\_XMLDOM package is used to access XMLType objects, and implements the Document Object Model (DOM), an application programming interface for HTML and XML documents.

**See Also:** *Oracle XML Developer's Kit Programmer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_XMLDOM](#)
  - Overview
  - Security Model
  - Constants
  - Types
  - Exceptions
- [Subprogram Groups](#)
  - DOMNode Subprograms
  - DOMAttr Subprograms
  - DOMCDATASection Subprograms
  - DOMCharacterData Subprograms
  - DOMComment Subprograms
  - DOMDocument Subprograms
  - DOMDocumentFragment Subprograms
  - DOMDocumentType Subprograms
  - DOMElement Subprograms
  - DOMEntity Subprograms
  - DOMEntityReference Subprograms
  - DOMImplementation Subprograms
  - DOMNamedNodeMap Subprograms
  - DOMNodeList Subprograms
  - DOMNotation Subprograms
  - DOMProcessingInstruction Subprograms

- 
- DOMText Subprograms
  - [Summary of DBMS\\_XMLDOM Subprograms](#)



## Using DBMS\_XMLDOM

---

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Types](#)
- [Exceptions](#)
- [Subprogram Groups](#)

## Overview

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents, and the manner in which they are accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense. XML is being increasingly used to represent many different kinds of information that may be stored in diverse systems. This information has been traditionally be seen as "data"; nevertheless, XML presents this data as documents, and the `DBMS_XMLDOM` package allows you access to both schema-based and non schema-based documents.

**Note:**

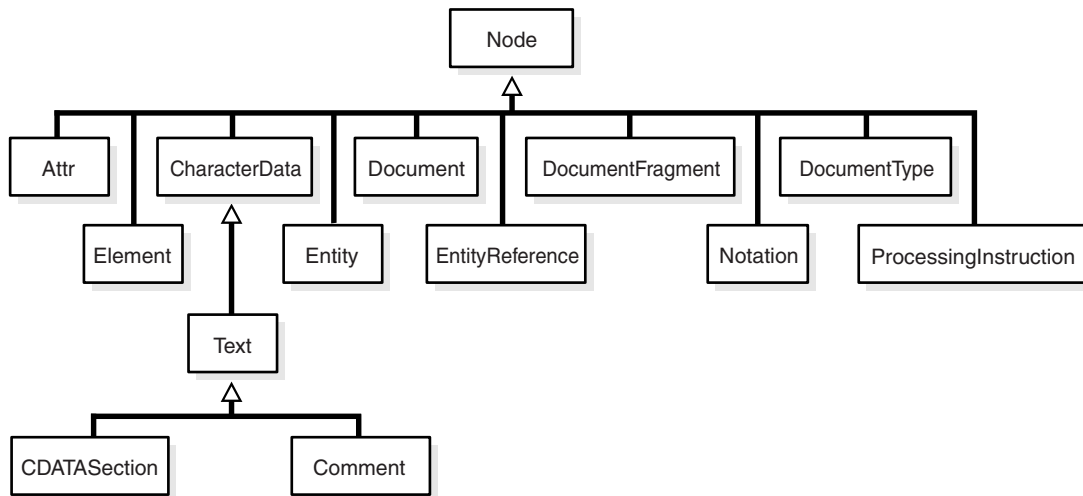
- Before database startup, the read-from and write-to directories in the `init.ORA` file must be specified; for example: `UTL_FILE_DIR=/mypath/insidemypath`.
- Read-from and write-to files must be on the server file system.

With DOM, anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions. In particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

One important objective of the W3C DOM specification is to provide a standard programming interface that can be used in a wide variety of environments, programming languages, and applications. Because the DOM standard is object-oriented while PL/SQL is essentially a procedural language, some changes had to be made:

- Various DOM interfaces such as `Node`, `Element`, and others have equivalent PL/SQL types `DOMNode`, `DOMElement`, respectively.
- Various `DOMException` codes such as `WRONG_DOCUMENT_ERR`, `HIERARCHY_REQUEST_ERR`, and others, have similarly named PL/SQL exceptions.
- Various DOM `Node` type codes such as `ELEMENT_NODE`, `ATTRIBUTE_NODE`, and others, have similarly named PL/SQL constants.
- Subprograms defined on a DOM type become functions or procedures that accept it as a parameter. For example, to perform [APPENDCHILD Function](#) on a `DOMNode n`, the [APPENDCHILD Function](#) PL/SQL function on page 170-42 is provided.
- To perform `setAttribute` on a `DOMElement elem` [SETATTRIBUTE Procedures](#), use PL/SQL procedure on page 170-142.

DOM defines an inheritance hierarchy. For example, `Document`, `Element`, and `Attr` are defined to be subtypes of `Node` (see [Figure 170-1](#)). Thus, a method defined in the `Node` interface should be available in these as well. Since such inheritance is not supported in PL/SQL, it is implemented through direct invocation of the `MAKENODE` function. Calling `MAKENODE` on various DOM types converts these types into a `DOMNode`. The appropriate functions or procedures that accept `DOMNodes` can then be called to operate on these types. If, subsequently, type specific functionality is desired, the `DOMNode` can be converted back into the original type by the `makeXXX` functions, where `DOMXXX` is the desired DOM type.

**Figure 170–1 Inheritance Diagram for DOM Types**

This is a text description of arpls010.gif. This figure provides a graphical representation of inheritance-type relationship between DOM types; methods defined on the supertype should be available on the subtype. Description of the inheritance structure follows:

1. Attr, CharacterData, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, and ProcessingInstruction are all subtypes of Node
2. Text is a subtype of CharacterData
3. CDATASection and Comment are subtypes of Text

\*\*\*\*\*

The implementation of this interface follows the REC-DOM-Level-1-19981001.

## Security Model

Owned by XDB, the DBMS\_XMLDOM package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Constants

Defined constants of DBMS\_XMLDOM are listed in [Table 170-1](#).

**Table 170-1** *Defined Constants for DBMS\_XMLDOM*

Constant	Type	Value	Description
ELEMENT_NODE	PLS_INTEGER	1	The Node is an Element.
ATTRIBUTE_NODE	PLS_INTEGER	2	The Node is an Attribute.
TEXT_NODE	PLS_INTEGER	3	The Node is a Text node.
CDATA_SECTION_NODE	PLS_INTEGER	4	The Node is a CDATA section.
ENTITY_REFERENCE_NODE	PLS_INTEGER	5	The Node is an Entity Reference.
ENTITY_NODE	PLS_INTEGER	6	The Node is an Entity.
PROCESSING_INSTRUCTION_NODE	PLS_INTEGER	7	The Node is a Processing Instruction.
COMMENT_NODE	PLS_INTEGER	8	The Node is a Comment.
DOCUMENT_NODE	PLS_INTEGER	9	The Node is a Document.
DOCUMENT_TYPE_NODE	PLS_INTEGER	10	The Node is a Document Type Definition.
DOCUMENT_FRAGMENT_NODE	PLS_INTEGER	11	The Node is a Document fragment.
NOTATION_NODE	PLS_INTEGER	12	The Node is a Notation.

## Types

The following types for `DBMS_XMLDOM.DOMTYPE` are defined in [Table 170–2](#):

**Table 170–2 XDB\_XMLDOM Types**

Type	Description
DOMATTR	Implements the DOM Attribute interface.
DOMCDATASECTION	Implements the DOM CDATASection interface.
DOMCHARACTERDATA	Implements the DOM Character Data interface.
DOMCOMMENT	Implements the DOM Comment interface.
DOMDOCUMENT	Implements the DOM Document interface.
DOMDOCUMENTFRAGMENT	Implements the DOM DocumentFragment interface.
DOMDOCUMENTTYPE	Implements the DOM Document Type interface.
DOMELEMENT	Implements the DOM Element interface.
DOMENTITY	Implements the DOM Entity interface.
DOMENTITYREFERENCE	Implements the DOM EntityReference interface.
DOMIMPLEMENTATION	Implements the DOM Implementation interface.
DOMNAMEDNODEMAP	Implements the DOM Named Node Map interface.
DOMNODE	Implements the DOM Node interface.
DOMNODELIST	Implements the DOM NodeList interface.
DOMNOTATION	Implements the DOM Notation interface.
DOMPROCESSINGINSTRUCTION	Implements the DOM Processing instruction interface.
DOMTEXT	Implements the DOM Text interface.

## Exceptions

The exceptions listed in [Table 170–3](#) are defined for DBMS\_XMLDOM:

**Table 170–3** *Exceptions for DBMS\_XMLDOM*

Exception	Description
DOMSTRING_SIZE_ERR	If the specified range of text does not fit into a DOMString.
HIERARCHY_REQUEST_ERR	If any node is inserted somewhere it doesn't belong.
INDEX_SIZE_ERR	If index or size is negative, or greater than the allowed value.
INUSE_ATTRIBUTE_ERR	If an attempt is made to add an attribute that is already in use elsewhere.
INVALID_CHARACTER_ERR	If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.
NO_DATA_ALLOWED_ERROR	If data is specified for a node that does not support data.
NOT_FOUND_ERR	If an attempt is made to reference a node in a context where it does not exist.
NO_MODIFICATION_ALLOWED_ERR	If an attempt is made to modify an object where modifications are not allowed.
NOT_SUPPORTED_ERR	If the implementation does not support the requested type of object or operation.
WRONG_DOCUMENT_ERR	If a node is used in a different document than the one that created it (that doesn't support it).

---

## Subprogram Groups

DBMS\_XMLDOM subprograms are divided into groups according to W3C Interfaces.

- [DOMNode Subprograms](#) on page 170-11
- [DOMAttr Subprograms](#) on page 170-13
- [DOMCDATASection Subprograms](#) on page 170-14
- [DOMCharacterData Subprograms](#) on page 170-15
- [DOMComment Subprograms](#) on page 170-16
- [DOMDocument Subprograms](#) on page 170-17
- [DOMDocumentFragment Subprograms](#) on page 170-19
- [DOMDocumentType Subprograms](#) on page 170-20
- [DOMElement Subprograms](#) on page 170-21
- [DOMEntity Subprograms](#) on page 170-22
- [DOMEntityReference Subprograms](#) on page 170-23
- [DOMImplementation Subprograms](#) on page 170-24
- [DOMNamedNodeMap Subprograms](#) on page 170-25
- [DOMNodeList Subprograms](#) on page 170-26
- [DOMNotation Subprograms](#) on page 170-27
- [DOMProcessingInstruction Subprograms](#) on page 170-28
- [DOMText Subprograms](#) on page 170-29



## DOMNode Subprograms

**Table 170–4 Summary of DOMNode Subprograms; DBMS\_XMLDOM**

Subprogram	Description
<a href="#">ADOPTNODE Function</a> on page 170-41	Adopts a node from another document
<a href="#">APPENDCHILD Function</a> on page 170-42	Appends a new child to the node
<a href="#">CLONENODE Function</a> on page 170-44	Clones the node
<a href="#">FREENODE Procedure</a> on page 170-60	Frees all resources associated with the node
<a href="#">GETATTRIBUTES Function</a> on page 170-64	Retrieves the attributes of the node
<a href="#">GETCHILDNODES Function</a> on page 170-66	Retrieves the children of the node
<a href="#">GETEXPANDEDNAME Procedure and Functions</a> on page 170-73	Retrieves the expanded name of the node
<a href="#">GETFIRSTCHILD Function</a> on page 170-74	Retrieves the first child of the node
<a href="#">GETLASTCHILD Function</a> on page 170-76	Retrieves the last child of the node
<a href="#">GETLOCALNAME Procedure and Functions</a> on page 170-78	Retrieves the local part of the qualified name
<a href="#">GETNAMESPACE Procedure and Functions</a> on page 170-81	Retrieves the node's namespace URI
<a href="#">GETNEXTSIBLING Function</a> on page 170-82	Retrieves the next sibling of the node
<a href="#">GETNODENAME Function</a> on page 170-84	Retrieves the Name of the Node
<a href="#">GETNODETYPE Function</a> on page 170-83	Retrieves the Type of the node
<a href="#">GETNODEVALUE Function</a> on page 170-85	Retrieves the Value of the Node
<a href="#">GETNODEVALUEASBINARYSTREAM Function &amp; Procedure</a> on page 170-86	Retrieves Node Value as binary stream
<a href="#">GETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</a> on page 170-87	Retrieves Node Value as character stream
<a href="#">GETOWNERDOCUMENT Function</a> on page 170-91	Retrieves the owner document of the node
<a href="#">GETPARENTNODE Function</a> on page 170-93	Retrieves the parent of this node
<a href="#">GETPREFIX Function</a> on page 170-94	Retrieves the namespace prefix
<a href="#">GETPREVIOUSIBLING Function</a> on page 170-95	Retrieves the previous sibling of the node
<a href="#">GETSCHEMANODE Function</a> on page 170-98	Retrieves the associated schema URI
<a href="#">HASATTRIBUTES Function</a> on page 170-107	Tests if the node has attributes
<a href="#">HASCHILDNODES Function</a> on page 170-108	Tests if the node has child nodes
<a href="#">IMPORTNODE Function</a> on page 170-110	Imports a node from another document
<a href="#">INSERTBEFORE Function</a> on page 170-111	Inserts a child before the reference child
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the node is NULL
<a href="#">MAKEATTR Function</a> on page 170-117	Casts the node to an Attribute
<a href="#">MAKECDATASECTION Function</a> on page 170-118	Casts the node to a CData Section

**Table 170–4 (Cont.) Summary of DOMNode Subprograms; DBMS\_XMLDOM**

<b>Subprogram</b>	<b>Description</b>
<a href="#">MAKECHARACTERDATA Function</a> on page 170-119	Casts the node to Character Data
<a href="#">MAKECOMMENT Function</a> on page 170-120	Casts the node to a Comment
<a href="#">MAKEDOCUMENT Function</a> on page 170-121	Casts the node to a DOM Document
<a href="#">MAKEDOCUMENTFRAGMENT Function</a> on page 170-122	Casts the node to a DOM Document Fragment
<a href="#">MAKEDOCUMENTTYPE Function</a> on page 170-123	Casts the node to a DOM Document Type
<a href="#">MAKEELEMENT Function</a> on page 170-124	Casts the node to a DOM Element
<a href="#">MAKEENTITY Function</a> on page 170-125	Casts the node to a DOM Entity
<a href="#">MAKEENTITYREFERENCE Function</a> on page 170-126	Casts the node to a DOM Entity Reference
<a href="#">MAKENOTATION Function</a> on page 170-130	Casts the node to a DOM Notation
<a href="#">MAKEPROCESSINGINSTRUCTION Function</a> on page 170-131	Casts the node to a DOM Processing Instruction
<a href="#">MAKETEXT Function</a> on page 170-132	Casts the node to a DOM Text
<a href="#">REMOVECHILD Function</a> on page 170-137	Removes a specified child from a node
<a href="#">REPLACECHILD Function</a> on page 170-139	Replaces the old child with a new child
<a href="#">SETNODEVALUE Procedure</a> on page 170-148	Sets the Value of the node
<a href="#">SETNODEVALUEASBINARYSTREAM Function &amp; Procedure</a> on page 170-149	Sets the Node Value as binary stream
<a href="#">SETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</a> on page 170-150	Sets the Node Value as a character stream
<a href="#">SETPREFIX Procedure</a> on page 170-151	Sets the namespace prefix
<a href="#">USEBINARYSTREAM Function</a> on page 170-157	Establishes that the stream is valid
<a href="#">WRITETOBUFFER Procedures</a> on page 170-158	Writes the contents of the node to a buffer
<a href="#">WRITETOCLOB Procedures</a> on page 170-159	Writes the contents of the node to a CLOB
<a href="#">WRITETOFILE Procedures</a> on page 170-160	Writes the contents of the node to a file

## DOMAttr Subprograms

**Table 170–5 Summary of DOMAttr Subprograms; DBMS\_XMLDOM**

Method	Description
<a href="#">GETEXPANDEDNAME Procedure and Functions</a> on page 170-73	Retrieves the expanded name of the attribute
<a href="#">GETLOCALNAME Procedure and Functions</a> on page 170-78	Retrieves the local name of the attribute
<a href="#">GETNAME Functions</a> on page 170-79	Retrieves the name of the attribute
<a href="#">GETNAMESPACE Procedure and Functions</a> on page 170-81	Retrieves the NS URI of the attribute
<a href="#">GETOWNERELEMENT Function</a> on page 170-92	Retrieves the Element node, parent of the attribute
<a href="#">GETQUALIFIEDNAME Functions</a> on page 170-97	Retrieves the Qualified Name of the attribute
<a href="#">GETSPECIFIED Function</a> on page 170-99	Tests if attribute was specified in the element
<a href="#">GETVALUE Function</a> on page 170-103	Retrieves the value of the attribute
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the Attribute node is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the Attribute to a node
<a href="#">SETVALUE Procedure</a> on page 170-153	Sets the value of the attribute

## DOMCDataSection Subprograms

**Table 170–6 Summary of DOMCdata Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the CDataSection is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the CDataSection to a node

## DOMCharacterData Subprograms

**Table 170–7 Summary of DOMCharacterData Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">APPENDDATA Procedure</a> on page 170-43	Appends the specified data to the node data
<a href="#">DELETEDATA Procedure</a> on page 170-54	Deletes the data from the specified offSets
<a href="#">GETDATA Functions</a> on page 170-68	Retrieves the data of the node
<a href="#">GETLENGTH Functions</a> on page 170-77	Retrieves the length of the data
<a href="#">INSERTDATA Procedure</a> on page 170-112	Inserts the data in the node at the specified offSets
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the CharacterData is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the CharacterData to a node
<a href="#">REPLACEDATA Procedure</a> on page 170-140	Changes a range of characters in the node
<a href="#">SETDATA Procedures</a> on page 170-145	Sets the data to the node
<a href="#">SUBSTRINGDATA Function</a> on page 170-156	Retrieves the substring of the data

## DOMComment Subprograms

**Table 170–8 Summary of DOMComment Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the comment is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the Comment to a node

## DOMDocument Subprograms

**Table 170–9 Summary of DOMDocument Subprograms; DBMS\_XMLDOM**

Method	Description
<a href="#">CREATEATTRIBUTE Functions</a> on page 170-45	Creates an Attribute
<a href="#">CREATEDATASECTION Function</a> on page 170-46	Creates a <code>CDataSection</code> node
<a href="#">CREATECOMMENT Function</a> on page 170-47	Creates a Comment node
<a href="#">CREATEDOCUMENT Function</a> on page 170-48	Creates a new Document
<a href="#">CREATEDOCUMENTFRAGMENT Function</a> on page 170-49	Creates a new Document Fragment
<a href="#">CREATEELEMENT Functions</a> on page 170-50	Creates a new Element
<a href="#">CREATEENTITYREFERENCE Function</a> on page 170-51	Creates an Entity reference
<a href="#">CREATEPROCESSINGINSTRUCTION Function</a> on page 170-52	Creates a Processing Instruction
<a href="#">CREATETEXTNODE Function</a> on page 170-53	Creates a Text node
<a href="#">FREEDOCFRAG Procedure</a> on page 170-57	Frees the document fragment
<a href="#">FREEDOCUMENT Procedure</a> on page 170-58	Frees the document
<a href="#">GETCHARSET Function</a> on page 170-65	Retrieves the character set of the DOM document
<a href="#">GETDOCTYPE Function</a> on page 170-69	Retrieves the DTD of the document
<a href="#">GETDOCUMENTELEMENT Function</a> on page 170-70	Retrieves the root element of the document
<a href="#">GETELEMENTSBYTAGNAME Functions</a> on page 170-71	Retrieves <ul style="list-style-type: none"> <li>■ the elements in the DOMNODELIST by tag name</li> <li>■ elements in the subtree of a DOMNODELIST by tagname</li> </ul>
<a href="#">GETIMPLEMENTATION Function</a> on page 170-75	Retrieves the DOM implementation
<a href="#">GETSTANDALONE Function</a> on page 170-100	Retrieves the standalone property of the document
<a href="#">GETVERSION Function</a> on page 170-104	Retrieves the version of the document
<a href="#">GETXMLTYPE Function</a> on page 170-105	Retrieves the <code>XMLType</code> associated with the DOM Document
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the document is <code>NULL</code>
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the document to a node
<a href="#">NEWDOMDOCUMENT Functions</a> on page 170-133	Creates a new document
<a href="#">SETCHARSET Procedure</a> on page 170-144	Sets the character set of the DOM document
<a href="#">SETDOCTYPE Procedure</a> on page 170-146	Sets the DTD of the document

**Table 170–9 (Cont.) Summary of DOMDocument Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">SETSTANDALONE Procedure</a> on page 170-152	Sets the standalone property of the document
<a href="#">SETVERSION Procedure</a> on page 170-154	Sets the version of the document
<a href="#">WRITETOBUFFER Procedures</a> on page 170-158	Writes the document to a buffer
<a href="#">WRITETOCLOB Procedures</a> on page 170-159	Writes the document to a CLOB
<a href="#">WRITETOFILE Procedures</a> on page 170-160	Writes the document to a file



## DOMDocumentFragment Subprograms

**Table 170–10 Summary of DOMDocumentFragment Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">FREEDOCFRAG Procedure</a> on page 170-57	Frees the specified document fragment
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the DocumentFragment is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the Document Fragment to a node
<a href="#">WRITETOBUFFER Procedures</a> on page 170-158	Writes the contents of a document fragment into a buffer

## DOMDocumentType Subprograms

**Table 170–11 Summary of DOMDocumentType Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">FINDENTITY Function</a> on page 170-55	Finds the specified entity in the document type
<a href="#">FINDNOTATION Function</a> on page 170-56	Finds the specified notation in the document type
<a href="#">GETENTITIES Function</a> on page 170-72	Retrieves the nodemap of entities in the Document type
<a href="#">GETNAME Functions</a> on page 170-79	Retrieves the name of the Document type
<a href="#">GETNOTATIONS Function</a> on page 170-89	Retrieves the nodemap of the notations in the Document type
<a href="#">GETPUBLICID Functions</a> on page 170-96	Retrieves the public ID of the document type
<a href="#">GETSYSTEMID Functions</a> on page 170-101	Retrieves the system ID of the document type
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the Document Type is <code>NULL</code>
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the document type to a node

## DOMElement Subprograms

**Table 170–12 Summary of DOMElement Subprograms; DBMS\_XMLDOM**

Method	Description
<a href="#">FREEELEMENT Procedure</a> on page 170-59	Frees memory allocated to a DOMElement handle
<a href="#">GETATTRIBUTE Functions</a> on page 170-62	Retrieves the attribute node by name
<a href="#">GETATTRIBUTENODE Functions</a> on page 170-63	Retrieves the attribute node by name
<a href="#">GETCHILDRENBYTAGNAME Functions</a> on page 170-67	Retrieves children of the element by tag name
<a href="#">GETELEMENTSBYTAGNAME Functions</a> on page 170-71	Retrieves <ul style="list-style-type: none"> <li>▪ the elements in the DOMNODELIST by tag name</li> <li>▪ elements in the subtree of a DOMNODELIST by tagname</li> </ul>
<a href="#">GETEXPANDEDNAME Procedure and Functions</a> on page 170-73	Retrieves the expanded name of the element
<a href="#">GETLOCALNAME Procedure and Functions</a> on page 170-78	Retrieves the local name of the element
<a href="#">GETNAMESPACE Procedure and Functions</a> on page 170-81	Retrieves the NS URI of the element
<a href="#">GETQUALIFIEDNAME Functions</a> on page 170-97	Retrieves the qualified name of the element
<a href="#">GETTAGNAME Function</a> on page 170-102	Retrieves the Tag name of the element
<a href="#">HASATTRIBUTE Functions</a> on page 170-106	Tests if an attribute exists
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the Element is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the Element to a node
<a href="#">NORMALIZE Procedure</a> on page 170-134	Normalizes the text children of the element
<a href="#">REMOVEATTRIBUTE Procedures</a> on page 170-135	Removes the attribute specified by the name
<a href="#">REMOVEATTRIBUTENODE Function</a> on page 170-136	Removes the attribute node in the element
<a href="#">RESOLVENAMESPACEPREFIX Function</a> on page 170-141	Resolve the prefix to a namespace URI
<a href="#">SETATTRIBUTE Procedures</a> on page 170-142	Sets the attribute specified by the name
<a href="#">SETATTRIBUTENODE Functions</a> on page 170-143	Sets the attribute node in the element

## DOMEntity Subprograms

**Table 170–13 Summary of DOMEntity Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">GETNOTATIONNAME Function</a> on page 170-88	Retrieves the notation name of the entity
<a href="#">GETPUBLICID Functions</a> on page 170-96	Retrieves the public Id of the entity
<a href="#">GETSYSTEMID Functions</a> on page 170-101	Retrieves the system Id of the entity
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the Entity is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the Entity to a node

## DOMEntityReference Subprograms

**Table 170–14 Summary of DOMEntityReference Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the DOMEntityReference is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the DOMEntityReference to NULL

## DOMImplementation Subprograms

**Table 170–15** Summary of DOMImplementation Subprograms; DBMS\_XMLDOM

Method	Description
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the DOMImplementation node is NULL
<a href="#">HASFEATURE Function</a> on page 170-109	Tests if the DOMImplementation implements a feature

## DOMNamedNodeMap Subprograms

**Table 170–16** Summary of DOMNamedNodeMap Subprograms; DBMS\_XMLDOM

Method	Description
<a href="#">GETLENGTH Functions</a> on page 170-77	Retrieves the number of items in the map
<a href="#">GETNAMEDITEM Function</a> on page 170-80	Retrieves the item specified by the name
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the NamedNodeMap is NULL
<a href="#">ITEM Functions</a> on page 170-116	Retrieves the item given the index in the map
<a href="#">REMOVENAMEDITEM Function</a> on page 170-138	Removes the item specified by name
<a href="#">SETNAMEDITEM Function</a> on page 170-147	Sets the item in the map specified by the name

## DOMNodeList Subprograms

**Table 170–17 Summary of DOMNodeList Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">FREENODELIST Procedure</a> on page 170-61	Frees all resources associated with a nodelist
<a href="#">GETLENGTH Functions</a> on page 170-77	Retrieves the number of items in the list
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the <code>NodeList</code> is <code>NULL</code>
<a href="#">ITEM Functions</a> on page 170-116	Retrieves the item given the index in the list



## DOMNotation Subprograms

**Table 170–18 Summary of DOMNotation Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">GETPUBLICID Functions</a> on page 170-96	Retrieves the public Id of the notation
<a href="#">GETSYSTEMID Functions</a> on page 170-101	Retrieves the system Id of the notation
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the Notation is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the notation to a node

## DOMProcessingInstruction Subprograms

**Table 170–19 Summary of DOMProcessingInstruction Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">GETDATA Functions</a> on page 170-68	Retrieves the data of the processing instruction
<a href="#">GETTARGET Function</a> on page 170-90	Retrieves the target of the processing instruction
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the Processing Instruction is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the Processing Instruction to a node
<a href="#">SETDATA Procedures</a> on page 170-145	Sets the data of the processing instruction

## DOMText Subprograms

**Table 170–20 Summary of DOMText Subprograms; DBMS\_XMLDOM**

<b>Method</b>	<b>Description</b>
<a href="#">ISNULL Functions</a> on page 170-113	Tests if the text is NULL
<a href="#">MAKENODE Functions</a> on page 170-127	Casts the text to a node
<a href="#">SPLITTEXT Function</a> on page 170-155	Splits the contents of the text node into 2 text nodes

## Summary of DBMS\_XMLDOM Subprograms

**Table 170–21 Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">ADOPTNODE Function</a> on page 170-41	Adopts a node from another document	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">APPENDCHILD Function</a> on page 170-42	Appends a new child to the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">APPENDDATA Procedure</a> on page 170-43	Appends the specified data to the node data	<a href="#">DOMCharacterData Subprograms</a> on page 170-15
<a href="#">CLONENODE Function</a> on page 170-44	Clones the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">CREATEATTRIBUTE Functions</a> on page 170-45	Creates an Attribute	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATECDATASECTION Function</a> on page 170-46	Creates a CDataSection node	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATECOMMENT Function</a> on page 170-47	Creates a Comment node	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATEDOCUMENT Function</a> on page 170-48	Creates a new Document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATEDOCUMENTFRAGMENT Function</a> on page 170-49	Creates a new Document Fragment	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATEELEMENT Functions</a> on page 170-50	Creates a new Element	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATEENTITYREFERENCE Function</a> on page 170-51	Creates an Entity reference	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATEPROCESSINGINSTRUCTION Function</a> on page 170-52	Creates a Processing Instruction	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">CREATETEXTNODE Function</a> on page 170-53	Creates a Text node	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">DELETEDATA Procedure</a> on page 170-54	Deletes the data from the specified offSets	<a href="#">DOMCharacterData Subprograms</a> on page 170-15
<a href="#">FINDENTITY Function</a> on page 170-55	Finds the specified entity in the document type	<a href="#">DOMDocumentType Subprograms</a> on page 170-20
<a href="#">FINDNOTATION Function</a> on page 170-56	Finds the specified notation in the document type	<a href="#">DOMDocumentType Subprograms</a> on page 170-20

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">FREEDOCFRAG Procedure</a> on page 170-57	Frees the document fragment	<a href="#">DOMDocument Subprograms</a> on page 170-17 and <a href="#">DOMDocumentFragment Subprograms</a> on page 170-19
<a href="#">FREEDOCUMENT Procedure</a> on page 170-58	Frees the document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">FREEELEMENT Procedure</a> on page 170-59	Frees memory allocated to a DOMELEMENT handle	<a href="#">DOMELEMENT Subprograms</a> on page 170-21
<a href="#">FREENODE Procedure</a> on page 170-60	Frees all resources associated with the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">FREENODELIST Procedure</a> on page 170-61	Frees all resources associated with a nodelist	<a href="#">DOMNodeList Subprograms</a> on page 170-26
<a href="#">GETATTRIBUTE Functions</a> on page 170-62	Retrieves the attribute node by name	<a href="#">DOMELEMENT Subprograms</a> on page 170-21
<a href="#">GETATTRIBUTENODE Functions</a> on page 170-63	Retrieves the attribute node by name	<a href="#">DOMELEMENT Subprograms</a> on page 170-21
<a href="#">GETATTRIBUTES Function</a> on page 170-64	Retrieves the attributes of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETCHARSET Function</a> on page 170-65	Retrieves the charset of the DOM document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">GETCHILDNODES Function</a> on page 170-66	Retrieves the children of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETCHILDRENBYTAGNAME Functions</a> on page 170-67	Retrieves children of the element by tag name	<a href="#">DOMCharacterData Subprograms</a> on page 170-15
<a href="#">GETDATA Functions</a> on page 170-68	Retrieves <ul style="list-style-type: none"> <li>■ the data of the node</li> <li>■ the data of the processing instruction</li> </ul>	<ul style="list-style-type: none"> <li>■ <a href="#">DOMCharacterData Subprograms</a> on page 170-15</li> <li>■ <a href="#">DOMProcessingInstruction Subprograms</a> on page 170-28</li> </ul>
<a href="#">GETDOCTYPE Function</a> on page 170-69	Retrieves the DTD of the document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">GETDOCUMENTELEMENT Function</a> on page 170-70	Retrieves the root element of the document	<a href="#">DOMDocument Subprograms</a> on page 170-17

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">GETELEMENTSBYTAGNAME Functions</a> on page 170-71	Retrieves <ul style="list-style-type: none"> <li>the elements in the DOMNODELIST by tag name</li> <li>elements in the subtree of a DOMNODELIST by tagname</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">DOMDocument Subprograms</a> on page 170-17</li> <li><a href="#">DOMEElement Subprograms</a> on page 170-21</li> </ul>
<a href="#">GETENTITIES Function</a> on page 170-72	Retrieves the nodemap of entities in the Document type	<a href="#">DOMDocumentType Subprograms</a> on page 170-20
<a href="#">GETEXPANDEDNAME Procedure and Functions</a> on page 170-73	Retrieves <ul style="list-style-type: none"> <li>the expanded name of the node</li> <li>the expanded name of the attribute</li> <li>the expanded name of the element</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">DOMNode Subprograms</a> on page 170-11</li> <li><a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li><a href="#">DOMEElement Subprograms</a> on page 170-21</li> </ul>
<a href="#">GETFIRSTCHILD Function</a> on page 170-74	Retrieves the first child of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETIMPLEMENTATION Function</a> on page 170-75	Retrieves the DOM implementation	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">GETLASTCHILD Function</a> on page 170-76	Retrieves the last child of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETLENGTH Functions</a> on page 170-77	Retrieves <ul style="list-style-type: none"> <li>the length of the data</li> <li>the number of items in the map</li> <li>the number of items in the list</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">DOMCharacterData Subprograms</a> on page 170-15</li> <li><a href="#">DOMNamedNodeMap Subprograms</a> on page 170-25</li> <li><a href="#">DOMNodeList Subprograms</a> on page 170-26</li> </ul>
<a href="#">GETLOCALNAME Procedure and Functions</a> on page 170-78	Retrieves <ul style="list-style-type: none"> <li>the local part of the qualified name</li> <li>the local name of the attribute</li> <li>the local name of the element</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">DOMNode Subprograms</a> on page 170-11</li> <li><a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li><a href="#">DOMEElement Subprograms</a> on page 170-21</li> </ul>
<a href="#">GETNAME Functions</a> on page 170-79	Retrieves <ul style="list-style-type: none"> <li>the name of the attribute</li> <li>the name of the Document type</li> </ul>	<ul style="list-style-type: none"> <li><a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li><a href="#">DOMDocumentType Subprograms</a> on page 170-20</li> </ul>

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">GETNAMEDITEM Function</a> on page 170-80	Retrieves <ul style="list-style-type: none"> <li>▪ an item specified by name</li> <li>▪ and namespace URI on page 170-25)</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNamedNodeMap Subprograms</a> on page 170-25</li> <li>▪ <a href="#">DOMNamedNodeMap Subprograms</a></li> </ul>
<a href="#">GETNAMESPACE Procedure and Functions</a> on page 170-81	Retrieves <ul style="list-style-type: none"> <li>▪ the node's namespace URI</li> <li>▪ the NS URI of the attribute</li> <li>▪ the NS URI of the element</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNode Subprograms</a> on page 170-11</li> <li>▪ <a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li>▪ <a href="#">DOMELEMENT Subprograms</a> on page 170-21</li> </ul>
<a href="#">GETNEXTSIBLING Function</a> on page 170-82	Retrieves the next sibling of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETNODENAME Function</a> on page 170-84	Retrieves the Name of the Node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETNODETYPE Function</a> on page 170-83	Retrieves the Type of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETNODEVALUE Function</a> on page 170-85	Retrieves the Value of the Node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETNODEVALUEASBINARY STREAM Function &amp; Procedure</a> on page 170-86	Retrieves the Node Value as binary stream	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</a> on page 170-87	Retrieves the Node Value as character stream	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETNOTATIONNAME Function</a> on page 170-88	Retrieves the notation name of the entity	<a href="#">DOMEntity Subprograms</a> on page 170-22
<a href="#">GETNOTATIONS Function</a> on page 170-89	Retrieves the nodemap of the notations in the Document type	<a href="#">DOMDocumentType Subprograms</a> on page 170-20
<a href="#">GETTARGET Function</a> on page 170-90	Retrieves the target of the processing instruction	<a href="#">DOMProcessingInstruction Subprograms</a> on page 170-28
<a href="#">GETOWNERDOCUMENT Function</a> on page 170-91	Retrieves the owner document of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETOWNERELEMENT Function</a> on page 170-92	Retrieves the Element node, parent of the attribute	<a href="#">DOMAttr Subprograms</a> on page 170-13
<a href="#">GETPARENTNODE Function</a> on page 170-93	Retrieves the parent of this node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETPREFIX Function</a> on page 170-94	Retrieves the namespace prefix )	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETPREVIOUSIBLING Function</a> on page 170-95	Retrieves the previous sibling of the node	<a href="#">DOMNode Subprograms</a> on page 170-11

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">GETPUBLICID Functions</a> on page 170-96	Retrieves <ul style="list-style-type: none"> <li>▪ the public ID of the document type</li> <li>▪ the public Id of the entity</li> <li>▪ the public Id of the notation</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMDocumentType Subprograms</a> on page 170-20</li> <li>▪ <a href="#">DOMEntity Subprograms</a> on page 170-22</li> <li>▪ <a href="#">DOMNotation Subprograms</a> on page 170-27</li> </ul>
<a href="#">GETQUALIFIEDNAME Functions</a> on page 170-97	Retrieves <ul style="list-style-type: none"> <li>▪ the Qualified Name of the attribute</li> <li>▪ the qualified name of the element</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li>▪ <a href="#">DOMElement Subprograms</a> on page 170-21</li> </ul>
<a href="#">GETSCHEMANODE Function</a> on page 170-98	Retrieves the associated schema URI	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">GETSPECIFIED Function</a> on page 170-99	Tests if attribute was specified in the element.	<a href="#">DOMAttr Subprograms</a> on page 170-13
<a href="#">GETSTANDALONE Function</a> on page 170-100	Retrieves the standalone property of the document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">GETSYSTEMID Functions</a> on page 170-101	Retrieves <ul style="list-style-type: none"> <li>▪ the system ID of the document type</li> <li>▪ the system Id of the entity</li> <li>▪ the system Id of the notation</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMDocumentType Subprograms</a> on page 170-20</li> <li>▪ <a href="#">DOMEntity Subprograms</a> on page 170-22</li> <li>▪ <a href="#">DOMNotation Subprograms</a> on page 170-27</li> </ul>
<a href="#">GETTAGNAME Function</a> on page 170-102	Retrieves the Tag name of the element	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">GETVALUE Function</a> on page 170-103	Retrieves the value of the attribute	<a href="#">DOMAttr Subprograms</a> on page 170-13
<a href="#">GETVERSION Function</a> on page 170-104	Retrieves the version of the document	<a href="#">DOMDocument Subprograms</a> on page 170-17)
<a href="#">GETXMLTYPE Function</a> on page 170-105	Retrieves the XMLType associated with the DOM Document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">HASATTRIBUTES Function</a> on page 170-107	Tests if the node has attributes	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">HASATTRIBUTE Functions</a> on page 170-106	Tests if an attribute exists	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">HASCHILDNODES Function</a> on page 170-108	Tests if the node has child nodes	<a href="#">DOMNode Subprograms</a> on page 170-11



**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">HASFEATURE Function</a> on page 170-109	Tests if the DOMImplementation implements a feature	<a href="#">DOMImplementation Subprograms</a> on page 170-24
<a href="#">IMPORTNODE Function</a> on page 170-110	Imports a node from another document	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">INSERTBEFORE Function</a> on page 170-111	Inserts a child before the reference child	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">INSERTDATA Procedure</a> on page 170-112	Inserts the data in the node at the specified offSets	<a href="#">DOMCharacterData Subprograms</a> on page 170-15

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">ISNULL Functions</a> on page 170-113	Tests	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNode Subprograms</a> on page 170-11</li> <li>▪ <a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li>▪ <a href="#">DOMCDataSection Subprograms</a> on page 170-14</li> <li>▪ <a href="#">DOMCharacterData Subprograms</a> on page 170-15</li> <li>▪ <a href="#">DOMComment Subprograms</a> on page 170-16</li> <li>▪ <a href="#">DOMDocument Subprograms</a> on page 170-17</li> <li>▪ <a href="#">DOMDocumentFragment Subprograms</a> on page 170-19</li> <li>▪ <a href="#">DOMDocumentType Subprograms</a> on page 170-20</li> <li>▪ <a href="#">DOMElement Subprograms</a> on page 170-21</li> <li>▪ <a href="#">DOMEntity Subprograms</a> on page 170-22</li> <li>▪ <a href="#">DOMEntityReference Subprograms</a> on page 170-23</li> <li>▪ <a href="#">DOMImplementation Subprograms</a> on page 170-24</li> <li>▪ <a href="#">DOMNamedNodeMap Subprograms</a> on page 170-25</li> <li>▪ <a href="#">DOMNodeList Subprograms</a> on page 170-26</li> <li>▪ <a href="#">DOMNotation Subprograms</a> on page 170-27</li> <li>▪ <a href="#">DOMProcessingInstruction Subprograms</a> on page 170-28</li> <li>▪ <a href="#">DOMText Subprograms</a> on page 170-29</li> </ul>
	▪ if the node is NULL	
	▪ if the Attribute node is NULL	
	▪ if the CDataSection is NULL	
	▪ if the CharacterData is NULL	
	▪ if the comment is NULL	
	▪ if the document is NULL	
	▪ if the DocumentFragment is NULL	
	▪ if the Document Type is NULL	
	▪ if the Element is NULL	
	▪ if the Entity is NULL	
	▪ if the DOMEntityReference is NULL	
	▪ if the DOMImplementation node is NULL	
	▪ if the NamedNodeMap is NULL	
	▪ if the NodeList is NULL	
	▪ if the Notation is NULL	
	▪ if the Processing Instruction is NULL	
▪ if the text is NULL		

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">ITEM Functions</a> on page 170-116	Retrieves <ul style="list-style-type: none"> <li>▪ the item given the index in the map</li> <li>▪ the item given the index in the <code>NodeList</code></li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNamedNodeMap Subprograms</a> on page 170-25</li> <li>▪ <a href="#">DOMNodeList Subprograms</a> on page 170-26</li> </ul>
<a href="#">MAKEATTR Function</a> on page 170-117	Casts the node to an Attribute	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKECDATASECTION Function</a> on page 170-118	Casts the node to a CDATA Section	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKECHARACTERDATA Function</a> on page 170-119	Casts the node to Character Data	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKECOMMENT Function</a> on page 170-120	Casts the node to a Comment	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKEDOCUMENT Function</a> on page 170-121	Casts the node to a DOM Document	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKEDOCUMENTFRAGMENT Function</a> on page 170-122	Casts the node to a DOM Document Fragment	<a href="#">DOMNode Subprograms</a> on page 170-11)
<a href="#">MAKEDOCUMENTTYPE Function</a> on page 170-123	Casts the node to a DOM Document Type	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKEELEMENT Function</a> on page 170-124	Casts the node to a DOM Element	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKEENTITY Function</a> on page 170-125	Casts the node to a DOM Entity	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKEENTITYREFERENCE Function</a> on page 170-126	Casts the node to a DOM Entity Reference	<a href="#">DOMNode Subprograms</a> on page 170-11

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">MAKENODE Functions</a> on page 170-127	Casts <ul style="list-style-type: none"> <li>▪ the Attribute to a node</li> <li>▪ the CDatasection to a node</li> <li>▪ the CharacterData to a node</li> <li>▪ the Comment to a node</li> <li>▪ the document to a node</li> <li>▪ the Document Fragment to a node</li> <li>▪ the document type to a node</li> <li>▪ the Element to a node</li> <li>▪ the Entity to a node</li> <li>▪</li> <li>▪ the DOMEntityReference to NULL</li> <li>▪ the notation to a node</li> <li>▪ the Processing Instruction to a node</li> <li>▪ the text to a node</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMAttr Subprograms</a> on page 170-13</li> <li>▪ <a href="#">DOMCDATASection Subprograms</a> on page 170-14</li> <li>▪ <a href="#">DOMCharacterData Subprograms</a> on page 170-15</li> <li>▪ <a href="#">DOMComment Subprograms</a> on page 170-16</li> <li>▪ <a href="#">DOMDocument Subprograms</a> on page 17</li> <li>▪ <a href="#">DOMDocumentFragment Subprograms</a> on page 170-19</li> <li>▪ <a href="#">DOMDocumentType Subprograms</a> on page 170-20</li> <li>▪ <a href="#">DOMElement Subprograms</a> on page 170-21</li> <li>▪ <a href="#">DOMEntity Subprograms</a> on page 170-22</li> <li>▪ <a href="#">DOMEntityReference Subprograms</a> on page 170-23</li> <li>▪ <a href="#">DOMNotation Subprograms</a> on page 170-27</li> <li>▪ <a href="#">DOMProcessingInstruction Subprograms</a> on page 170-28</li> <li>▪ <a href="#">DOMText Subprograms</a> on page 170-29</li> </ul>
<a href="#">MAKENOTATION Function</a> on page 170-130	Casts the node to a DOM Notation	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKEPROCESSINGINSTRUCTION Function</a> on page 170-131	Casts the node to a DOM Processing Instruction	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">MAKETEXT Function</a> on page 170-132	Casts the node to a DOM Text	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">NEWDOMDOCUMENT Functions</a> on page 170-133	Creates a new document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">NORMALIZE Procedure</a> on page 170-134	Normalizes the text children of the element	<a href="#">DOMElement Subprograms</a> on page 170-21

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

Subprogram	Description	Group
<a href="#">REMOVEATTRIBUTE Procedures</a> on page 170-135	Removes the attribute specified by the name	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">REMOVEATTRIBUTENODE Function</a> on page 170-136	Removes the attribute node in the element	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">REMOVECHILD Function</a> on page 170-137	Removes a specified child from a node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">REMOVENAMEDITEM Function</a> on page 170-138	Removes the item specified by name	<a href="#">DOMNamedNodeMap Subprograms</a> on page 170-25
<a href="#">REPLACECHILD Function</a> on page 170-139	Replaces the old child with a new child	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">REPLACEDATA Procedure</a> on page 170-140	Changes a range of characters in the node	<a href="#">DOMCharacterData Subprograms</a> on page 170-15
<a href="#">RESOLVENAMESPACEPREFIX Function</a> on page 170-141	Resolve the prefix to a namespace URI	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">SETATTRIBUTE Procedures</a> on page 170-142	Sets the attribute specified by the name	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">SETATTRIBUTENODE Functions</a> on page 170-143	Sets the attribute node in the element	<a href="#">DOMElement Subprograms</a> on page 170-21
<a href="#">SETCHARSET Procedure</a> on page 170-144	Sets the character set of the DOM document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">SETDATA Procedures</a> on page 170-145	Sets <ul style="list-style-type: none"> <li>■ the data to the node</li> <li>■ the data of the processing instruction</li> </ul>	<ul style="list-style-type: none"> <li>■ <a href="#">DOMCharacterData Subprograms</a> on page 170-15</li> <li>■ <a href="#">DOMProcessingInstruction Subprograms</a> on page 170-28</li> </ul>
<a href="#">SETDOCTYPE Procedure</a> on page 170-146	Sets the DTD of the document.	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">SETNAMEDITEM Function</a> on page 170-147	Sets the item in the map specified by the name	<a href="#">DOMNamedNodeMap Subprograms</a> on page 170-25
<a href="#">SETNODEVALUE Procedure</a> on page 170-148	Sets the Value of the node	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">SETNODEVALUEASBINARYSTREAM Function &amp; Procedure</a> on page 170-149	Sets the Node Value as a binary stream	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">SETNODEVALUEASCHARACTERSTREAM Function &amp; Procedure</a> on page 170-150	Sets the Node Value as a character stream	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">SETPREFIX Procedure</a> on page 170-151	Sets the namespace prefix	<a href="#">DOMNode Subprograms</a> on page 170-11

**Table 170–21 (Cont.) Summary of DBMS\_XMLDOM Package Subprogram**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">SETSTANDALONE Procedure</a> on page 170-152	Sets the standalone property of the document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">SETVALUE Procedure</a> on page 170-153	Sets the value of the attribute	<a href="#">DOMAttr Subprograms</a> on page 170-13
<a href="#">SETVERSION Procedure</a> on page 170-154	Sets the version of the document	<a href="#">DOMDocument Subprograms</a> on page 170-17
<a href="#">SPLITTEXT Function</a> on page 170-155	Splits the contents of the text node into 2 text nodes	<a href="#">DOMText Subprograms</a> on page 170-29
<a href="#">SUBSTRINGDATA Function</a> on page 170-156	Retrieves the substring of the data	<a href="#">DOMCharacterData Subprograms</a> on page 170-15
<a href="#">USEBINARYSTREAM Function</a> on page 170-157	Strabismus that the stream is valid for use	<a href="#">DOMNode Subprograms</a> on page 170-11
<a href="#">WRITETOBUFFER Procedures</a> on page 170-158	Writes <ul style="list-style-type: none"> <li>▪ the contents of the node to a buffer</li> <li>▪ the document to a buffer</li> <li>▪ the contents of a document fragment into a buffer</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNode Subprograms</a> on page 170-11</li> <li>▪ <a href="#">DOMDocument Subprograms</a> on page 170-17</li> <li>▪ <a href="#">DOMDocumentFragment Subprograms</a> on page 170-19</li> </ul>
<a href="#">WRITETOCLOB Procedures</a> on page 170-159	Writes <ul style="list-style-type: none"> <li>▪ the contents of the node to a CLOB</li> <li>▪ the document to a CLOB</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNode Subprograms</a> on page 170-11</li> <li>▪ <a href="#">DOMDocument Subprograms</a> on page 170-17</li> </ul>
<a href="#">WRITETOFILE Procedures</a> on page 170-160	Writes <ul style="list-style-type: none"> <li>▪ the contents of the node to a file</li> <li>▪ the document to a file</li> </ul>	<ul style="list-style-type: none"> <li>▪ <a href="#">DOMNode Subprograms</a> on page 170-11</li> <li>▪ <a href="#">DOMDocument Subprograms</a> on page 170-17</li> </ul>

## ADOPTNODE Function

This function adopts a node from another document, and returns this new node.

**See Also:** [DOMNode Subprograms](#) on page 170-11 for other subprograms in this group

### Syntax

```
DBMS_XMLDOM.ADOPTNODE(  
    doc          IN  DOMDocument,  
    importedNode IN  DOMNode)  
RETURN DOMNODE;
```

### Parameters

**Table 170–22** *ADOPTNODE Function Parameters*

Parameter	Description
doc	Document that is adopting the node
importedNode	Node to adopt

### Usage Notes

Note that the [ADOPTNODE Function](#) removes the node from the source document while the [IMPORTNODE Function](#) clones the node in the source document.

## APPENDCHILD Function

This function adds the node `newchild` to the end of the list of children of this node, and returns the newly added node. If the `newchild` is already in the tree, it is first removed.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.APPENDCHILD(  
    n           IN      DOMNode,  
    newchild   IN      DOMNode)  
RETURN DOMNODE;
```

### Parameters

**Table 170–23** *APPENDCHILD Function Parameters*

Parameter	Description
<code>n</code>	DOMNode
<code>newchild</code>	The child to be appended to the list of children of node <code>n</code>



## APPENDDATA Procedure

This procedure appends the string to the end of the character data of the node. Upon success, data provides access to the concatenation of data and the specified string argument.

**See Also:** [DOMCharacterData Subprograms](#) on page 170-15

### Syntax

```
DBMS_XMLDOM.APPENDDATA (  
  cd      IN      DOMCHARACTERDATA,  
  arg     IN      VARCHAR2);
```

### Parameters

**Table 170–24** APPENDDATA Procedure Parameters

Parameter	Description
cd	DOMCHARACTERDATA
arg	The data to append to the existing data

## CLONENODE Function

This function returns a duplicate of this node, and serves as a generic copy constructor for nodes. The duplicate node has no parent, its parent node is NULL.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.CLONENODE (  
    n          IN    DOMNODE,  
    deep       IN    BOOLEAN)  
RETURN DOMNODE;
```

### Parameters

**Table 170–25** CLONENODE Function Parameters

Parameter	Description
n	DOMNODE
deep	Determines if children are to be cloned

### Usage Notes

- Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node.
- Cloning an Attribute directly, as opposed to be cloned as part of an Element cloning operation, returns a specified attribute (specified is TRUE).
- Cloning any other type of node simply returns a copy of this node.

## CREATEATTRIBUTE Functions

This function creates a DOMATTR node.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

Creates a DOMATTR with the specified name:

```
DBMS_XMLDOM.CREATEATTRIBUTE (
  doc      IN   DOMDOCUMENT,
  name     IN   VARCHAR2)
RETURN DOMATTR;
```

Creates a DOMATTR with the specified name and namespace URI:

```
DBMS_XMLDOM.CREATEATTRIBUTE (
  doc      IN   DOMDOCUMENT,
  qname    IN   VARCHAR2,
  ns       IN   VARCHAR2)
RETURN DOMATTR;
```

### Parameters

**Table 170–26 CREATEATTRIBUTE Function Parameters**

Parameter	Description
doc	DOMDOCUMENT
qname	New attribute qualified name
ns	Namespace

## CREATECDATASECTION Function

This function creates a DOMCDATASECTION node.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATECDATASECTION(  
  doc      IN      DOMDOCUMENT,  
  data     IN      VARCHAR2)  
RETURN DOMCDATASECTION;
```

### Parameters

**Table 170–27** CREATECDATASECTION Function Parameters

Parameter	Description
doc	DOMDOCUMENT
data	Content of the DOMCDATASECTION node

## CREATECOMMENT Function

This function creates a DOMCOMMENT node.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATECOMMENT (  
    doc      IN      DOMDOCUMENT,  
    data     IN      VARCHAR2)  
RETURN DOMCOMMENT;
```

### Parameters

**Table 170–28** *CREATECOMMENT Function Parameters*

Parameter	Description
doc	DOMDOCUMENT
data	Content of the DOMComment node

## CREATEDOCUMENT Function

This function creates a DOMDOCUMENT with specified namespace URI, root element name, DTD.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATEDOCUMENT(  
  namespaceURI      IN      VARCHAR2,  
  qualifiedName     IN      VARCHAR2,  
  doctype           IN      DOMTYPE := NULL)  
RETURN DOMDOCUMENT;
```

### Parameters

**Table 170–29** *CREATEDOCUMENT Function Parameters*

Parameter	Description
namespaceURI	Namespace URI
qualifiedName	Root element name
doctype	Document type

## CREATEDOCUMENTFRAGMENT Function

This function creates a DOMDOCUMENTFRAGMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATEDOCUMENTFRAGMENT (  
    doc      IN      DOMDOCUMENT)  
RETURN DOMDOCUMENTFRAGMENT;
```

### Parameters

**Table 170–30** *CREATEDOCUMENTFRAGMENT Function Parameters*

Parameter	Description
doc	DOMDocument

## CREATEELEMENT Functions

This function creates a DOMELEMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

Creates a DOMELEMENT with specified name:

```
DBMS_XMLDOM.CREATEELEMENT (  
    doc          IN      DOMDOCUMENT,  
    tagName     IN      VARCHAR2)  
RETURN DOMELEMENT;
```

Creates a DOMELEMENT with specified name and namespace URI:

```
DBMS_XMLDOM.CREATEELEMENT (  
    doc          IN      DOMDOCUMENT,  
    tagName     IN      VARCHAR2,  
    ns          IN      VARCHAR2)  
RETURN DOMELEMENT;
```

### Parameters

**Table 170–31 CREATEELEMENT Function Parameters**

Parameter	Description
doc	DOMDOCUMENT
tagName	Tagname for new DOMELEMENT
ns	Namespace



## CREATEENTITYREFERENCE Function

This function creates a DOENTITYREFERENCE node.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATEENTITYREFERENCE (  
    doc          IN      DOMDOCUMENT,  
    name         IN      VARCHAR2)  
RETURN DOENTITYREFERENCE;
```

### Parameters

**Table 170–32** *CREATEENTITYREFERENCE Function Parameters*

Parameter	Description
doc	DOMDOCUMENT
name	New entity reference name

## CREATEPROCESSINGINSTRUCTION Function

This function creates a DOMPROCESSINGINSTRUCTION node.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATEPROCESSINGINSTRUCTION(  
  doc      IN      DOMDocument,  
  target   IN      VARCHAR2,  
  data     IN      VARCHAR2)  
RETURN DOMPROCESSINGINSTRUCTION;
```

### Parameters

**Table 170–33** *CREATEPROCESSINGINSTRUCTION Function Parameters*

Parameter	Description
doc	DOMDOCUMENT
target	Target of the new processing instruction
data	Content data of the new processing instruction

## CREATETEXTNODE Function

This function creates a DOMTEXT node.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.CREATETEXTNODE (  
    doc      IN      DOMDocument,  
    data     IN      VARCHAR2)  
RETURN DOMTEXT;
```

### Parameters

**Table 170–34** *CREATETEXTNODE Function Parameters*

Parameter	Description
doc	DOMDOCUMENT
data	Content of the DOMText node

## DELETEDATA Procedure

This procedure removes a range of characters from the node. Upon success, data and length reflect the change.

**See Also:** [DOMCharacterData Subprograms](#) on page 170-15

### Syntax

```
DBMS_XMLDOM.DELETEDATA (  
  cd          IN      DOMCHARACTERDATA,  
  offset      IN      NUMBER,  
  cnt         IN      NUMBER);
```

### Parameters

**Table 170–35 DELETEDATA PROCEDURE Parameters**

Parameter	Description
cd	DOMCHARACTERDATA
offset	The offset from which to delete the data
cnt	The number of characters (starting from offset) to delete

## FINDENTITY Function

This function finds an entity in the specified DTD, and returns that entity if found.

**See Also:** [DOMDocumentType Subprograms](#) on page 170-20

### Syntax

```
DBMS_XMLDOM.FINDENTITY (  
    dt      IN      DOMDOCUMENTTYPE,  
    name    IN      VARCHAR2,  
    par     IN      BOOLEAN)  
RETURN DUMENTITY;
```

### Parameters

**Table 170–36** *FINDENTITY Function Parameters*

Parameter	Description
dt	The DTD
name	Entity to find
par	Flag to indicate type of entity; TRUE for parameter entity and FALSE for normal entity

## FINDNOTATION Function

This function finds the notation in the specified DTD, and returns it, if found.

**See Also:** [DOMDocumentType Subprograms](#) on page 170-20

### Syntax

```
DBMS_XMLDOM.FINDNOTATION(  
    dt          IN      DOMDocumentType,  
    name       IN      VARCHAR2)  
RETURN DOMNOTATION;
```

### Parameters

**Table 170–37** *FINDNOTATION Function Parameters*

Parameter	Description
dt	The DTD
name	The notation to find

## FREEDOCFRAG Procedure

This procedure frees the specified document fragment.

**See Also:** [DOMDocument Subprograms](#) on page 170-17 and [DOMDocumentFragment Subprograms](#) on page 170-19

### Syntax

```
DBMS_XMLDOM.FREEDOCFRAG (  
    df      IN      DOMDOCUMENTFRAGMENT) ;
```

### Parameters

**Table 170–38** *FREEDOCFRAG Procedure Parameters*

Parameter	Description
df	DOM document fragment

## FREEDOCUMENT Procedure

This procedure frees DOMDOCUMENT object.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.FREEDOCUMENT (  
    doc      IN      DOMDOCUMENT) ;
```

### Parameters

**Table 170–39** *FREEDOCUMENT Procedure Parameters*

Parameter	Description
doc	DOMDOCUMENT



## FREEELEMENT Procedure

This procedure frees memory allocated to a DOMELEMENT handle.

**See Also:** [DOMELEMENT Subprograms](#) on page 170-21

### Syntax

```
DBMS_XMLDOM.FREENODE (  
    elem      IN      DOMELEMENT);
```

### Parameters

**Table 170–40** *FREENODE Procedure Parameters*

Parameter	Description
elem	Of type DOMELEMENT

## FREENODE Procedure

This procedure frees all resources associated with a DOMNODE.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.FREENODE (  
    n          IN      DOMNODE);
```

### Parameters

**Table 170–41** *FREENODE Procedure Parameters*

Parameter	Description
n	DOMNODE

## FREENODELIST Procedure

This procedure frees all resources associated with a nodelist.

**See Also:** [DOMNodeList Subprograms](#) on page 170-26

### Syntax

```
DBMS_XMLDOM.FREENODE(  
    nl    IN    DOMNodeList);
```

### Parameters

**Table 170–42** *FREENODE Procedure Parameters*

Parameter	Description
nl	Of type DOMNODELIST

## GETATTRIBUTE Functions

This function returns the value of an attribute of an `DOMELEMENT` by name.

**See Also:** [DOMElement Subprograms](#) on page 170-21

### Syntax

Returns the value of a `DOMELEMENT`'s attribute by name:

```
DBMS_XMLDOM.GETATTRIBUTE (  
    elem      IN      DOMELEMENT,  
    name      IN      VARCHAR2)  
RETURN VARCHAR2;
```

Returns the value of a `DOMELEMENT`'s attribute by name and namespace URI:

```
DBMS_XMLDOM.GETATTRIBUTE (  
    elem      IN      DOMELEMENT,  
    name      IN      VARCHAR2,  
    ns        IN      VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–43** *GETATTRIBUTE Function Parameters*

Parameter	Description
<code>elem</code>	The <code>DOMELEMENT</code>
<code>name</code>	Attribute name
<code>ns</code>	Namespace

## GETATTRIBUTENODE Functions

This function returns an attribute node from the `DOMELEMENT` by name. The function is overloaded. The specific forms of functionality are described along with the syntax declarations.

**See Also:** [DOMElement Subprograms](#) on page 170-21

### Syntax

Returns an attribute node from the `DOMELEMENT` by name:

```
DBMS_XMLDOM.GETATTRIBUTENODE (
    elem      IN      DOMELEMENT,
    name      IN      VARCHAR2)
RETURN DOMATTR;
```

Returns an attribute node from the `DOMELEMENT` by name and namespace URI:

```
DBMS_XMLDOM.GETATTRIBUTENODE (
    elem      IN      DOMELEMENT,
    name      IN      VARCHAR2,
    ns        IN      VARCHAR2)
RETURN DOMATTR;
```

### Parameters

**Table 170–44** *GETATTRIBUTENODE Function Parameters*

Parameter	Description
<code>elem</code>	The <code>DOMELEMENT</code>
<code>name</code>	Attribute name; * matches any attribute
<code>ns</code>	Namespace

## GETATTRIBUTES Function

This function retrieves a NAMEDNODEMAP containing the attributes of this node (if it is an Element) or NULL otherwise.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETATTRIBUTES(  
    n          IN          DOMNode)  
RETURN DOMNAMEDNODEMAP;
```

### Parameters

**Table 170–45** GETATTRIBUTES Function Parameters

Parameter	Description
n	DOMNode

## GETCHARSET Function

This function retrieves the character set of the DOM document.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETCHARSET(  
    doc IN    DOMDocument)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–46** *GETCHARSET Function Parameters*

Parameter	Description
doc	DOM document

### Usage Notes

For a newly parsed document, we return the database character set. Once the SETCHARSET Procedure is called with a non-NULL value for charset, that charset is returned.

## GETCHILDNODES Function

This function retrieves a DOMNODELIST that contains all children of this node. If there are no children, this is a DOMNODELIST containing no nodes.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETCHILDNODES(  
    n          IN      DOMNode)  
RETURN DOMNodeList;
```

### Parameters

**Table 170–47** *GETCHILDNODES Function Parameters*

Parameter	Description
n	DOMNode



## GETCHILDRENBYTAGNAME Functions

This function returns the children of the `DOMELEMENT`.

**See Also:** [DOMElement Subprograms](#) on page 170-21

### Syntax

Returns children of the `DOMELEMENT` given the tag name:

```
DBMS_XMLDOM.GETCHILDRENBYTAGNAME (
  elem      IN      DOMElement,
  name      IN      VARCHAR2)
RETURN DOMNODELIST;
```

Returns children of the `DOMELEMENT` given the tag name and namespace:

```
DBMS_XMLDOM.GETCHILDRENBYTAGNAME (
  elem      IN      DOMElement,
  name      IN      VARCHAR2,
  ns        IN      VARCHAR2)
RETURN DOMNODELIST;
```

### Parameters

**Table 170–48** *GETCHILDRENBYTAGNAME Function Parameters*

Parameter	Description
elem	DOMELEMENT
name	Tag name
ns	Namespace

## GETDATA Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Gets the character data of the node that implements this interface (See Also: [DOMCharacterData Subprograms](#) on page 170-15):

```
DBMS_XMLDOM.GETDATA (  
    cd      IN    DOMCHARACTERDATA)  
RETURN VARCHAR2;
```

Returns the content data of the DOMProcessingInstruction (See Also: [DOMProcessingInstruction Subprograms](#) on page 170-28):

```
DBMS_XMLDOM.GETDATA (  
    pi      IN    DOMPROCESSINGINSTRUCTION)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–49** *GETDATA Function Parameters*

Parameter	Description
cd	DOMCHARACTERDATA
pi	The DOMPROCESSINGINSTRUCTION

## GETDOCTYPE Function

This function returns the DTD associated to the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETDOCTYPE(  
    doc      IN      DOMDOCUMENT)  
RETURN DOMDOCUMENTTYPE;
```

### Parameters

**Table 170–50** *GETDOCTYPE Function Parameters*

Parameter	Description
doc	DOMDOCUMENT

## GETDOCUMENTELEMENT Function

This function returns the root element of the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETDOCUMENTELEMENT (  
    doc      IN      DOMDOCUMENT)  
RETURN DOMELEMENT;
```

### Parameters

**Table 170–51** *GETDOCUMENTELEMENT Function Parameters*

Parameter	Description
doc	DOMDOCUMENT

## GETELEMENTSBYTAGNAME Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Returns a DOMNODELIST of all the elements with a specified tagname (See Also: [DOMDocument Subprograms](#) on page 170-17):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME (
    doc          IN      DOMDOCUMENT,
    tagname      IN      VARCHAR2)
RETURN DOMNODELIST;
```

Returns the element children of the DOMELEMENT given the tag name (See Also: [DOMELEMENT Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME (
    elem        IN      DOMELEMENT,
    name        IN      VARCHAR2)
RETURN DOMNODELIST;
```

Returns the element children of the DOMELEMENT given the tag name and namespace (See Also: [DOMELEMENT Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME (
    elem        IN      DOMELEMENT,
    name        IN      VARCHAR2,
    ns          IN      VARCHAR2)
RETURN DOMNODELIST;
```

### Parameters

**Table 170-52** *GETELEMENTSBYTAGNAME Function Parameters*

Parameter	Description
doc	DOMDOCUMENT
tagname	Name of the tag to match on
elem	The DOMELEMENT
name	Tag name; using a wildcard(*) would match any tag
ns	Namespace

## GETENTITIES Function

This function retrieves a DOMNAMEDNODEMAP containing the general entities, both external and internal, declared in the DTD.

**See Also:** [DOMDocumentType Subprograms](#) on page 170-20

### Syntax

```
DBMS_XMLDOM.GETENTITIES(  
    dt          IN      DOMDocumentType)  
RETURN DOMNAMEDNODEMAP;
```

### Parameters

**Table 170–53** *GETENTITIES Function Parameters*

Parameter	Description
dt	DOMDOCUMENTTYPE

## GETEXPANDEDNAME Procedure and Functions

This subprogram is overloaded as a procedure and two functions. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Retrieves the expanded name of the `Node` if it is in an `Element` or `Attribute` type; otherwise, returns `NULL` (See Also: [DOMNode Subprograms](#) on page 170-11)

```
DBMS_XMLDOM.GETEXPANDEDNAME (
  n          IN          DOMNODE
  data      OUT          VARCHAR);
```

Returns the expanded name of the `DOMAttr` (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.GETEXPANDEDNAME (
  a          IN          DOMAttr)
RETURN VARCHAR2;
```

Returns the expanded name of the `DOMElement` (See Also: [DOMElement Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.GETEXPANDEDNAME (
  elem      IN          DOMELEMENT)
RETURN VARCHAR2;
```

### Parameters

**Table 170–54** *GETEXPANDEDNAME Procedure and Function Parameters*

Parameter	Description
n	DOMNODE
data	Returned expanded name of the <code>Node</code>
a	DOMATTR
elem	DOMELEMENT

## GETFIRSTCHILD Function

This function retrieves the first child of this node. If there is no such node, this returns NULL.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETFIRSTCHILD(  
    n          IN          DOMNODE)  
RETURN DOMNODE;
```

### Parameters

**Table 170–55** *GETFIRSTCHILD Function Parameters*

Parameter	Description
n	DOMNODE



## GETIMPLEMENTATION Function

This function returns the DOMIMPLEMENTATION object that handles this DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETIMPLEMENTATION(  
    doc      IN      DOMDOCUMENT)  
RETURN DOMIMPLEMENTATION;
```

### Parameters

**Table 170–56** *GETIMPLEMENTATION Function Parameters*

Parameter	Description
doc	DOMDOCUMENT

## GETLASTCHILD Function

This function retrieves the last child of this node. If there is no such node, this returns NULL.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETLASTCHILD(  
    n      IN  DOMNODE)  
RETURN DOMNODE;
```

### Parameters

**Table 170–57** GETLASTCHILD Function Parameters

Parameter	Description
n	DOMNODE

## GETLENGTH Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Gets the number of characters in the data. This may have the value zero, because CharacterData nodes may be empty (See Also: [DOMCharacterData Subprograms](#) on page 170-15):

```
DBMS_XMLDOM.GETLENGTH (
    cd      IN      DOMCHARACTERDATA)
RETURN NUMBER;
```

Gets the number of nodes in this map. The range of valid child node indexes is 0 to length-1, inclusive (See Also: [DOMNamedNodeMap Subprograms](#) on page 170-25):

```
DBMS_XMLDOM.GETLENGTH (
    nnm     IN      DOMNAMEDNODEMAP)
RETURN NUMBER;
```

Gets the number of nodes in the list. The range of valid child node indexes is 0 to length-1, inclusive (See Also: [DOMNodeList Subprograms](#) on page 170-26):

```
DBMS_XMLDOM.GETLENGTH (
    nl      IN      DOMNODELIST)
RETURN NUMBER;
```

### Parameters

**Table 170–58** GETLENGTH Function Parameters

Parameter	Description
cd	DOMCHARACTERDATA
nnm	DOMNAMEDNODEMAP
nl	DOMNODELIST

## GETLOCALNAME Procedure and Functions

This function is overloaded as a procedure and two functions. The specific forms of functionality are described alongside the syntax declarations.

### Syntax

Retrieves the local part of the node's qualified name (See Also: [DOMNode Subprograms](#) on page 170-11):

```
DBMS_XMLDOM.GETLOCALNAME (
    n          IN    DOMNODE,
    data      OUT   VARCHAR2);
```

Returns the local name of the DOMAttr (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.GETLOCALNAME (
    a          IN    DOMATTR)
RETURN VARCHAR2;
```

Returns the local name of the DOMELEMENT (See Also: [DOMELEMENT Subprograms](#) on page 170-21)

```
DBMS_XMLDOM.GETLOCALNAME (
    elem      IN    DOMELEMENT)
RETURN VARCHAR2;
```

### Parameters

**Table 170–59 GETLOCALNAME Procedure and Function Parameters**

Parameter	Description
n	DOMNode
data	Returned local name.
a	DOMAttr.
elem	DOMELEMENT.

## GETNAME Functions

This function is overloaded. The specific forms of functionality are described alongside the syntax declarations.

### Syntax

Returns the name of this attribute (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.GETNAME (  
    a          IN      DOMATTR)  
RETURN VARCHAR2;
```

Retrieves the name of DTD, or the name immediately following the DOCTYPE keyword (See Also: [DOMDocumentType Subprograms](#) on page 170-20):

```
DBMS_XMLDOM.GETNAME (  
    dt          IN      DOMDOCUMENTTYPE)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–60** GETNAME Function Parameters

Parameter	Description
a	DOMATTR
dt	DOMDOCUMENTTYPE

## GETNAMEDITEM Function

This function retrieves a node specified by name.

**See Also:** [DOMNamedNodeMap Subprograms](#) on page 170-25

### Syntax

Retrieves a node specified by name:

```
DBMS_XMLDOM.GETNAMEDITEM(  
  nnm    IN  DOMNAMEDNODEMAP,  
  name   IN  VARCHAR2)  
RETURN DOMNODE;
```

Retrieves a node specified by name and namespace URI:

```
DBMS_XMLDOM.GETNAMEDITEM(  
  nnm    IN  DOMNAMEDNODEMAP,  
  name   IN  VARCHAR2,  
  ns     IN  VARCHAR2)  
RETURN DOMNODE;
```

### Parameters

**Table 170–61** *GETNAMEDITEM Function Parameters*

Parameter	Description
nnm	DOMNAMEDNODEMAP
name	Name of the item to be retrieved
ns	Namespace

## GETNAMESPACE Procedure and Functions

This subprogram is overloaded as a procedure and two functions. The specific forms of functionality are described alongside the syntax declarations.

### Syntax

Retrieves the namespace URI associated with the node (See Also: [DOMNode Subprograms](#) on page 170-11):

```
DBMS_XMLDOM.GETNAMESPACE (
    n          IN    DOMNODE,
    data      OUT   VARCHAR2);
```

Retrieves the namespace of the DOMATTR (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.GETNAMESPACE (
    a          IN    DOMATTR)
RETURN VARCHAR2;
```

Retrieves the namespace of the DOMELEMENT (See Also: [DOMELEMENT Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.GETNAMESPACE (
    elem      IN    DOMELEMENT)
RETURN VARCHAR2;
```

### Parameters

**Table 170–62** *GETNAMESPACE Procedure and Function Parameters*

Parameter	Description
n	DOMNODE
data	Returned namespace URI
a	DOMATTR
elem	DOMELEMENT

## GETNEXTSIBLING Function

This function retrieves the node immediately following this node. If there is no such node, this returns `NULL`.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETNEXTSIBLING(  
    n          IN      DOMNODE)  
RETURN DOMNode;
```

### Parameters

**Table 170–63** *GETNEXTSIBLING Function Parameters*

Parameter	Description
n	DOMNODE



## GETNODETYPE Function

This function retrieves a code representing the type of the underlying object.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETNODETYPE (  
    n          IN      DOMNODE)  
RETURN NUMBER;
```

### Parameters

**Table 170–64** *GETNODETYPE Function Parameters*

Parameter	Description
n	DOMNODE

## GETNODENAME Function

This function gets the name of the node depending on its type.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETNODENAME (  
    n          IN      DOMNODE)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–65** *GETNODENAME Function Parameters*

Parameter	Description
n	DOMNODE

## GETNODEVALUE Function

This function gets the value of this node, depending on its type.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETNODEVALUE (  
    n          IN      DOMNODE)  
RETURN VARCHAR2;*
```

### Parameters

**Table 170–66** *GETNODEVALUE Function Parameters*

Parameter	Description
n	DOMNODE

## GETNODEVALUEASBINARYSTREAM Function & Procedure

The operation of these subprograms is described with each syntax implementation.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

This function returns an instance of the PL/SQL `XMLBinaryInputStream`. The node data type must be `RAW` or `BLOB` – if not an exception is raised.

```
DBMS_XMLDOM.GETNODEVALUEASBINARYSTREAM (
    n          IN      DOMNODE)
RETURN SYS.UTL_BINARYINPUTSTREAM;
```

Using this procedure, the application passes an implementation of `SYS.UTL_BINARYOUTPUTSTREAM` into which XDB writes the contents of the node. The data type of the node must be `RAW` or `CLOB` – if not an exception is raised.

```
DBMS_XMLDOM.GETNODEVALUEASBINARYSTREAM (
    n          in      DOMNODE,
    value      in      SYS.UTL_BINARYOUTPUTSTREAM);
```

### Parameters

**Table 170–67** *GETNODEVALUEASBINARYSTREAM Function & Procedure Parameters*

Parameter	Description
n	DOMNODE
value	BINARYOUTPUTSTREAM

## GETNODEVALUEASCHARACTERSTREAM Function & Procedure

The operation of these subprograms is described with each syntax implementation.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

This function returns an instance of the PL/SQL `XMLCharacterInputStream`. If the node data is character it is converted to the current session character set. If the node data is not character data, it is first converted to character data.

```
DBMS_XMLDOM.GETNODEVALUEASCHARACTERSTREAM (
    n          IN      DOMNODE)
RETURN SYS.UTL_CHARACTERINPUTSTREAM;
```

Using this procedure, the node data is converted, as necessary, to the session character set and then "pushed" into the `SYS.UTL_CHARACTEROUTPUTSTREAM`.

```
DBMS_XMLDOM.GETNODEVALUEASCHARACTERSTREAM (
    n          IN      DOMNODE,
    value      IN      SYS.UTL_CHARACTEROUTPUTSTREAM);
```

### Parameters

**Table 170–68** *GETNODEVALUEASCHARACTERSTREAM Function & Procedure Parameters*

Parameter	Description
n	DOMNODE
value	CHARACTEROUTPUTSTREAM

## GETNOTATIONNAME Function

This function returns the notation name of the `DOENTITY`.

**See Also:** [DOMEntity Subprograms](#) on page 170-22

### Syntax

```
DBMS_XMLDOM.GETNOTATIONNAME(  
    ent          IN      DOENTITY)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–69** *GETNOTATIONNAME Function Parameters*

Parameter	Description
ent	DOENTITY

## GETNOTATIONS Function

This function retrieves a DOMNAMEDNODEMAP containing the notations declared in the DTD.

**See Also:** [DOMDocumentType Subprograms](#) on page 170-20

### Syntax

```
DBMS_XMLDOM.GETNOTATIONS (  
    dt          IN      DOMDOCUMENTTYPE)  
RETURN DOMNAMEDNODEMAP;
```

### Parameters

**Table 170–70** *GETNOTATIONS Function Parameters*

Parameter	Description
dt	DOMDOCUMENTTYPE

## GETTARGET Function

This function returns the target of the DOMPROCESSINGINSTRUCTION.

**See Also:** [DOMProcessingInstruction Subprograms](#) on page 170-28

### Syntax

```
DBMS_XMLDOM.GETTARGET(  
    pi          IN      DOMPROCESSINGINSTRUCTION)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–71** *GETTARGET Function Parameters*

Parameter	Description
pi	DOMPROCESSINGINSTRUCTION



## GETOWNERDOCUMENT Function

This function retrieves the Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document or a Document Type that is not used with any Document yet, this is NULL.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETOWNERDOCUMENT(  
    n          IN      DOMNODE)  
RETURN DOMDOCUMENT;
```

### Parameters

**Table 170–72** *GETOWNERDOCUMENT Function Parameters*

Parameter	Description
n	DOMNODE

## GETOWNERELEMENT Function

This function retrieves the Element node to which the specified Attribute is attached.

**See Also:** [DOMAttr Subprograms](#) on page 170-13

### Syntax

```
DBMS_XMLDOM.GETOWNERELEMENT(  
    a          IN      DOMATTR)  
RETURN DOMELEMENT;
```

### Parameters

**Table 170–73** *GETOWNERELEMENT Function Parameters*

Parameter	Description
a	Attribute

## GETPARENTNODE Function

This function retrieves the parent of this node. All nodes, except `Attr`, `Document`, `DocumentFragment`, `Entity`, and `Notation` may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `NULL`.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETPARENTNODE (  
    n          IN      DOMNODE)  
RETURN DOMNODE;
```

### Parameters

**Table 170–74** *GETPARENTNODE Function Parameters*

Parameter	Description
n	DOMNODE

## GETPREFIX Function

This function retrieves the namespace prefix of the node.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETPREFIX(  
    n          IN      DOMNODE)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–75** *GETPREFIX Function Parameters*

Parameter	Description
n	DOMNODE

## GETPREVIOUSIBLING Function

This function retrieves the node immediately preceding this node. If there is no such node, this returns NULL.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETPREVIOUSIBLING (  
    n          IN      DOMNODE)  
RETURN DOMNODE;
```

### Parameters

**Table 170–76** *GETPREVIOUSIBLING Function Parameters*

Parameter	Description
n	DOMNODE

## GETPUBLICID Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Returns the public identifier of the specified DTD (See Also: [DOMDocumentType Subprograms](#) on page 170-20):

```
DBMS_XMLDOM.GETPUBLICID (
    dt          IN      DOMDOCUMENTTYPE)
RETURN VARCHAR2;
```

Returns the public identifier of the DOENTITY (See Also: [DOMEntity Subprograms](#) on page 170-22):

```
DBMS_XMLDOM.GETPUBLICID (
    ent        IN      DOENTITY)
RETURN VARCHAR2;
```

Returns the public identifier of the DOMNOTATION (See Also: [DOMNotation Subprograms](#) on page 170-27):

```
DBMS_XMLDOM.GETPUBLICID (
    n          IN      DOMNOTATION)
RETURN VARCHAR2;
```

### Parameters

**Table 170–77** *GETPUBLICID Function Parameters*

Parameter	Description
dt	The DTD
ent	DOENTITY
n	DOMNOTATION

## GETQUALIFIEDNAME Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Returns the qualified name of the DOMATTR (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.GETQUALIFIEDNAME (  
    a          IN      DOMATTR)  
RETURN VARCHAR2;
```

Returns the qualified name of the DOMELEMENT (See Also: [DOMELEMENT Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.GETQUALIFIEDNAME (  
    elem      IN      DOMELEMENT)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–78** GETQUALIFIEDNAME Functions Parameters

Parameter	Description
a	DOMATTR
elem	DOMELEMENT

## GETSCHEMANODE Function

This function retrieves the schema URI associated with the node.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.GETSCHEMANODE(  
    n          IN      DOMNODE)  
RETURN DOMNODE;
```

### Parameters

**Table 170–79** GETSCHEMANODE Function Parameters

Parameter	Description
n	DOMNODE



## GETSPECIFIED Function

If this attribute was explicitly specified, a value in the original document, this is true; otherwise, it is false.

**See Also:** [DOMAttr Subprograms](#) on page 170-13

### Syntax

```
DBMS_XMLDOM.GETSPECIFIED(  
    a          IN      DOMATTR)  
RETURN BOOLEAN;
```

### Parameters

**Table 170–80** *GETSPECIFIED Function Parameters*

Parameter	Description
a	DOMATTR

## GETSTANDALONE Function

This function returns the standalone property associated with the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETSTANDALONE (  
    doc          IN      DOMDOCUMENT)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–81** *GETSTANDALONE Function Parameters*

Parameter	Description
doc	DOMDOCUMENT.

## GETSYSTEMID Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Returns the system id of the specified DTD (See Also: [DOMDocumentType Subprograms](#) on page 170-20):

```
DBMS_XMLDOM.GETSYSTEMID(
    dt          IN      DOMDOCUMENTTYPE)
RETURN VARCHAR2;
```

Returns the system identifier of the DUMENTITY (See Also: [DOMEntity Subprograms](#) on page 170-22):

```
DBMS_XMLDOM.GETSYSTEMID(
    ent        IN      DUMENTITY)
RETURN VARCHAR2;
```

Returns the system identifier of the DOMNOTATION (See Also: [DOMNotation Subprograms](#) on page 170-27):

```
DBMS_XMLDOM.GETSYSTEMID(
    n          IN      DOMNOTATION)
RETURN VARCHAR2;
```

### Parameters

**Table 170–82 GETSYSTEMID Function Parameters**

Parameter	Description
dt	The DTD.
ent	DOMEntity.
n	DOMNotation.

## GETTAGNAME Function

This function returns the name of the DOMELEMENT.

**See Also:** [DOMELEMENT Subprograms](#) on page 170-21

### Syntax

```
DBMS_XMLDOM.GETTAGNAME(  
    elem          IN      DOMELEMENT)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–83** *GETTAGNAME Function Parameters*

Parameter	Description
elem	The DOMELEMENT

## GETVALUE Function

This function retrieves the value of the attribute.

**See Also:** [DOMAttr Subprograms](#) on page 170-13

### Syntax

```
DBMS_XMLDOM.GETVALUE (  
    a          IN      DOMATTR)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–84** GETVALUE Function Parameters

Parameter	Description
a	DOMATTR

## GETVERSION Function

This function returns the version of the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETVERSION(  
    doc          IN      DOMDOCUMENT)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–85** *GETVERSION Function Parameters*

Parameter	Description
doc	DOMDOCUMENT

## GETXMLTYPE Function

This function returns the XMLType associated with the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.GETXMLTYPE(  
    doc          IN      DOMDOCUMENT)  
RETURN SYS.XMLTYPE;
```

### Parameters

**Table 170–86** GETXMLTYPE Function Parameters

Parameter	Description
doc	DOMDOCUMENT

## HASATTRIBUTE Functions

Verifies whether an attribute has been defined for `DOMELEMENT`, or has a default value.

**See Also:** [DOMElement Subprograms](#) on page 170-21

### Syntax

Verifies whether an attribute with the specified name has been defined for `DOMElement`:

```
DBMS_XMLDOM.HASATTRIBUTE (
    elem      IN  DOMELEMENT,
    name      IN  VARCHAR2)
RETURN VARCHAR2;
```

Verifies whether an attribute with specified name and namespace URI has been defined for `DOMELEMENT`; namespace enabled:

```
DBMS_XMLDOM.HASATTRIBUTE (
    elem      IN  DOMELEMENT,
    name      IN  VARCHAR2,
    ns        IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 170–87 HASATTRIBUTE Function Parameters**

Parameter	Description
<code>elem</code>	The <code>DOMELEMENT</code>
<code>name</code>	Attribute name; * matches any attribute
<code>ns</code>	Namespace



## HASATTRIBUTES Function

This function returns whether this node has any attributes.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.HASATTRIBUTES (  
    n          IN      DOMNODE)  
RETURN BOOLEAN;
```

### Parameters

**Table 170–88** *HASATTRIBUTES Function Parameters*

Parameter	Description
n	DOMNODE

## HASCHILDNODES Function

This function determines whether this node has any children.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.HASCHILDNODES (  
    n          IN      DOMNODE)  
RETURN BOOLEAN;
```

### Parameters

**Table 170–89** HASCHILDNODES Function Parameters

Parameter	Description
n	DOMNODE

## HASFEATURE Function

This function tests if the DOMIMPLEMENTATION implements a specific feature.

**See Also:** [DOMImplementation Subprograms](#) on page 170-24

### Syntax

```
DBMS_XMLDOM.HASFEATURE (  
    di          IN      DOMIMPLEMENTATION,  
    feature     IN      VARCHAR2,  
    version     IN      VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 170–90 HASFEATURE Function Parameters**

Parameter	Description
di	DOMIMPLEMENTATION
feature	The feature to check for
version	The version of the DOM to check in

## IMPORTNODE Function

This function imports a node from an external document and returns this new node.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.IMPORTNODE(  
    doc           IN  DOMDOCUMENT,  
    importedNode IN  DOMNODE,  
    deep          IN  BOOLEAN)  
RETURN DOMNODE;
```

### Parameters

**Table 170–91** *IMPORTNODE Function Parameters*

Parameter	Description
doc	Document from which the node is imported
importedNode	Node to import
deep	Setting for recursive import. <ul style="list-style-type: none"><li>■ If this value is <code>TRUE</code>, the entire subtree of the node will be imported with the node.</li><li>■ If this value is <code>FALSE</code>, only the node itself will be imported.</li></ul>

### Usage Notes

Note that the [ADOPTNODE Function](#) removes the node from the source document while the [IMPORTNODE Function](#) clones the node in the source document.

## INSERTBEFORE Function

This function inserts the node `newchild` before the existing child node `refchild`. If `refchild` is `NULL`, insert `newchild` at the end of the list of children.

If `newchild` is a `DOCUMENTFRAGMENT` object, all of its children are inserted, in the same order, before `refchild`. If the `newchild` is already in the tree, it is first removed.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.INSERTBEFORE (
  n          IN      DOMNODE,
  newchild  IN      DOMNODE,
  refchild  IN      DOMNODE)
RETURN DOMNode;
```

### Parameters

**Table 170–92** *INSERTBEFORE Function Parameters*

Parameter	Description
<code>n</code>	DOMNODE
<code>newChild</code>	The child to be inserted in the DOMNODE
<code>refChild</code>	The reference node before which the <code>newchild</code> is to be inserted

## INSERTDATA Procedure

This procedure inserts a string at the specified character offset.

**See Also:** [DOMCharacterData Subprograms](#) on page 170-15

### Syntax

```
DBMS_XMLDOM.INSERTDATA(  
  cd      IN      DOMCHARACTERDATA,  
  offset  IN      NUMBER,  
  arg     IN      VARCHAR2);
```

### Parameters

**Table 170–93** *INSERTDATA Procedure Parameters*

Parameter	Description
cd	DOMCHARACTERDATA
offset	The offset at which to insert the data
arg	The value to be inserted

## ISNULL Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Checks if the specified DOMNODE is NULL. Returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMNode Subprograms](#) on page 170-11):

```
DBMS_XMLDOM.ISNULL (
    n          IN      DOMNODE)
RETURN BOOLEAN;
```

Checks that the specified DOMATTR is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.ISNULL (
    a          IN      DOMATTR)
RETURN BOOLEAN;
```

Checks that the specified DOMCDATASECTION is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMCDATASection Subprograms](#) on page 170-14):

```
DBMS_XMLDOM.ISNULL (
    cds       IN      DOMCDATASECTION)
RETURN BOOLEAN;
```

Checks that the specified DOMCHARACTERDATA is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMCharacterData Subprograms](#) on page 170-15):

```
DBMS_XMLDOM.ISNULL (
    cd        IN      DOMCHARACTERDATA)
RETURN BOOLEAN;
```

Checks that the specified DOMCOMMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMComment Subprograms](#) on page 170-16):

```
DBMS_XMLDOM.ISNULL (
    com       IN      DOMCOMMENT)
RETURN BOOLEAN;
```

Checks that the specified DOMDOCUMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMDocument Subprograms](#) on page 170-17):

```
DBMS_XMLDOM.ISNULL (
    doc       IN      DOMDOCUMENT)
RETURN BOOLEAN;
```

Checks that the specified DOMDOCUMENTFRAGMENT is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMDocumentFragment Subprograms](#) on page 170-19):

```
DBMS_XMLDOM.ISNULL (
    df        IN      DOMDOCUMENTFRAGMENT)
RETURN BOOLEAN;
```

Checks that the specified DOMDOCUMENTTYPE is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMDocumentType Subprograms](#) on page 170-20):

```
DBMS_XMLDOM.ISNULL (
    dt        IN      DOMDOCUMENTTYPE)
RETURN BOOLEAN;
```

```
RETURN BOOLEAN;
```

Checks that the specified `DOMELEMENT` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMElement Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.ISNULL (
    elem      IN      DOMELEMENT)
RETURN BOOLEAN;
```

Checks that the specified `DOMENTITY` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMEntity Subprograms](#) on page 170-22):

```
DBMS_XMLDOM.ISNULL (
    ent       IN      DUMENTITY)
RETURN BOOLEAN;
```

Checks that the specified `DOMENTITYREFERENCE` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMEntityReference Subprograms](#) on page 170-23):

```
DBMS_XMLDOM.ISNULL (
    EREF      IN      DUMENTITYREFERENCE)
RETURN BOOLEAN;
```

Checks that the specified `DOMIMPLEMENTATION` is NULL; returns TRUE if it is NULL (See Also: [DOMImplementation Subprograms](#) on page 170-24):

```
DBMS_XMLDOM.ISNULL (
    di        IN      DOMIMPLEMENTATION)
RETURN BOOLEAN;
```

Checks that the specified `DOMNAMEDNODEMAP` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMNamedNodeMap Subprograms](#) on page 170-25):

```
DBMS_XMLDOM.ISNULL (
    nnm       IN      DOMNAMEDNODEMAP)
RETURN BOOLEAN;
```

Checks that the specified `DOMNODELIST` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMNodeList Subprograms](#) on page 170-26):

```
DBMS_XMLDOM.ISNULL (
    nl        IN      DOMNODELIST)
RETURN BOOLEAN;
```

Checks that the specified `DOMNOTATION` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMNotation Subprograms](#) on page 170-27):

```
DBMS_XMLDOM.ISNULL (
    n         IN      DOMNOTATION)
RETURN BOOLEAN;
```

Checks that the specified `DOMPROCESSINGINSTRUCTION` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMProcessingInstruction Subprograms](#) on page 170-28):

```
DBMS_XMLDOM.ISNULL (
    pi        IN      DOMPROCESSINGINSTRUCTION)
RETURN BOOLEAN;
```

Checks that the specified `DOMTEXT` is NULL; returns TRUE if it is NULL, FALSE otherwise (See Also: [DOMText Subprograms](#) on page 170-29):

```
DBMS_XMLDOM.ISNULL (
```



```

t      IN      DOMTEXT)
RETURN BOOLEAN;

```

## Parameters

**Table 170–94 ISNULL Function Parameters**

Parameter	Description
n	DOMNODE to check
a	DOMATTR to check
cds	DOMCDATASECTION to check
cd	DOMCHARACTERDATA to check
com	DOMCOMMENT to check
doc	DOMDOCUMENT to check
dF	DOMDOCUMENTFRAGMENT to check
dt	DOMDOCUMENTTYPE to check
elem	DOMELEMENT to check
ent	DOMENTITY to check
eref	DOMENTITYREFERENCE to check
di	DOMIMPLEMENTATION to check
nnm	DOMNAMENODEMAP to check
nl	DOMNODELIST to check
n	DOMNOTATION to check
pi	DOMPROCESSINGINSTRUCTION to check
t	DOMTEXT to check

## ITEM Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Returns the item in the map which corresponds to the `INDEX` parameter. If `INDEX` is greater than or equal to the number of nodes in this map, this returns `NULL` (See Also: [DOMNamedNodeMap Subprograms](#) on page 170-25):

```
DBMS_XMLDOM.ITEM(
    nnm      IN      DOMNAMEDNODEMAP,
    index    IN      NUMBER)
RETURN DOMNODE;
```

Returns the item in the collection which corresponds to the `INDEX` parameter. If `index` is greater than or equal to the number of nodes in the list, this returns `NULL` (See Also: [DOMNodeList Subprograms](#) on page 170-26):

```
DBMS_XMLDOM.ITEM(
    nl       IN      DOMNODELIST,
    index    IN      NUMBER)
RETURN DOMNODE;
```

### Parameters

**Table 170–95** *ITEM Function Parameters*

Parameter	Description
<code>nnm</code>	<code>DOMNAMEDNODEMAP</code>
<code>index</code>	The index in the node map at which the item is to be retrieved
<code>nl</code>	<code>DOMNODELIST</code>
<code>index</code>	The index in the <code>NodeList</code> used to retrieve the item

## MAKEATTR Function

This function casts a specified DOMNODE to a DOMATTR, and returns the DOMATTR.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEATTR (  
    n          IN      DOMNODE)  
RETURN DOMATTR;
```

### Parameters

**Table 170–96** MAKEATTR Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKECDATASECTION Function

This function casts a specified DOMNODE to a DOMCDATASECTION.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKECDATASECTION (  
    n          IN      DOMNODE)  
RETURN DOMCDATASECTION;
```

### Parameters

**Table 170–97 MAKECDATASECTION Function Parameters**

Parameter	Description
n	DOMNODE to cast

## MAKECHARACTERDATA Function

This function casts a specified DOMNODE to a DOMCHARACTERDATA, and returns the DOMCHARACTERDATA.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKECHARACTERDATA (  
    n          IN          DOMNode)  
RETURN DOMCharacterData;
```

### Parameters

**Table 170–98** MAKECHARACTERDATA Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKECOMMENT Function

This function casts a specified DOMNODE to a DOMCOMMENT, and returns the DOMCOMMENT.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKECOMMENT (  
    n          IN      DOMNODE)  
RETURN DOMCOMMENT;
```

### Parameters

**Table 170–99** MAKECOMMENT Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKEDOCUMENT Function

This function casts a specified DOMNODE to a DOMDOCUMENT, and returns the DOMDOCUMENT.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEDOCUMENT (  
    n          IN      DOMNODE)  
RETURN DOMDocument;
```

### Parameters

**Table 170–100** *MAKEDOCUMENT Function Parameters*

Parameter	Description
n	DOMNODE to cast

## MAKEDOCUMENTFRAGMENT Function

This function casts a specified DOMNODE to a DOMDOCUMENTFRAGMENT, and returns the DOMDOCUMENTFRAGMENT.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEDOCUMENTFRAGMENT (  
    n          IN      DOMNODE)  
RETURN DOMDOCUMENTFRAGMENT;
```

### Parameters

**Table 170–101** *MAKEDOCUMENTFRAGMENT Function Parameters*

Parameter	Description
n	DOMNODE to cast



## MAKEDOCUMENTTYPE Function

This function casts a specified DOMNODE to a DOMDOCUMENTTYPE and returns the DOMDOCUMENTTYPE.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEDOCUMENTTYPE (  
    n          IN      DOMNODE)  
RETURN DOMDOCUMENTTYPE;
```

### Parameters

**Table 170–102** *MAKEDOCUMENTTYPE Function Parameters*

Parameter	Description
n	DOMNODE to cast.

## MAKEELEMENT Function

This function casts a specified DOMNODE to a DOMELEMENT, and returns the DOMELEMENT.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEELEMENT (  
    n          IN      DOMNODE)  
RETURN DOMELEMENT;
```

### Parameters

**Table 170–103** MAKEELEMENT Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKEENTITY Function

This function casts a specified DOMNODE to a DUMENTITY, and returns the DUMENTITY.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEENTITY (  
    n          IN      DOMNODE)  
RETURN DUMENTITY;
```

### Parameters

**Table 170–104** MAKEENTITY Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKEENTITYREFERENCE Function

This function casts a specified DOMNODE to a DUMENTITYREFERENCE, and returns the DUMENTITYREFERENCE.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEENTITYREFERENCE(  
    n          IN      DOMNODE)  
RETURN DUMENTITYREFERENCE;
```

### Parameters

**Table 170–105** MAKEENTITYREFERENCE Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKENODE Functions

This function is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Casts specified `DOMATTR` to a `DOMNODE`, and returns the `DOMNODE` (See Also: [DOMAttr Subprograms](#) on page 170-13):

```
DBMS_XMLDOM.MAKENODE (
    a          IN      DOMATTR)
RETURN DOMNODE;
```

Casts the `DOMCDATASECTION` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMCDATASection Subprograms](#) on page 170-14):

```
DBMS_XMLDOM.MAKENODE (
    cds        IN      DOMCDATASECTION)
RETURN DOMNODE;
```

Casts the specified `DOMCHARACTERDATA` as a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMCharacterData Subprograms](#) on page 170-15):

```
DBMS_XMLDOM.MAKENODE (
    cd         IN      DOMCHARACTERDATA)
RETURN DOMNODE;
```

Casts the specified `DOMCOMMENT` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMComment Subprograms](#) on page 170-16):

```
DBMS_XMLDOM.MAKENODE (
    com        IN      DOMCOMMENT)
RETURN DOMNODE;
```

Casts the `DOMDOCUMENT` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMDocument Subprograms](#) on page 170-17):

```
DBMS_XMLDOM.MAKENODE (
    doc        IN      DOMDOCUMENT)
RETURN DOMNODE;
```

Casts the specified `DOMDOCUMENTFRAGMENT` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMDocumentFragment Subprograms](#) on page 170-19):

```
DBMS_XMLDOM.MAKENODE (
    df         IN      DOMDOCUMENTFRAGMENT)
RETURN DOMNode;
```

Casts the specified `DOMDOCUMENTTYPE` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMDocumentType Subprograms](#) on page 170-20):

```
DBMS_XMLDOM.MAKENODE (
    dt         IN      DOMDOCUMENTTYPE)
RETURN DOMNODE;
```

Casts the specified `DOMELEMENT` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMElement Subprograms](#) on page 170-21):

```
DBMS_XMLDOM.MAKENODE (
    elem       IN      DOMELEMENT)
```

```
RETURN DOMNODE;
```

Casts specified `DOMENTITY` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMEntity Subprograms](#) on page 170-22):

```
DBMS_XMLDOM.MAKENODE (
    ent      IN      DOMENTITY)
RETURN DOMNODE;
```

Casts the `DOMENTITYREFERENCE` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMEntityReference Subprograms](#) on page 170-23):

```
DBMS_XMLDOM.MAKENODE (
   eref     IN      DOMENTITYREFERENCE)
RETURN DOMNODE;
```

Casts the `DOMNOTATION` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMNotation Subprograms](#) on page 170-27):

```
DBMS_XMLDOM.MAKENODE (
    n       IN      DOMNOTATION)
RETURN DOMNODE;
```

Casts the `DOMPROCESSINGINSTRUCTION` to a `DOMNODE`, and returns the `DOMNODE` (See Also: [DOMProcessingInstruction Subprograms](#) on page 170-28):

```
DBMS_XMLDOM.MAKENODE (
    pi      IN      DOMPROCESSINGINSTRUCTION)
RETURN DOMNODE;
```

Casts the `DOMTEXT` to a `DOMNODE`, and returns that `DOMNODE` (See Also: [DOMText Subprograms](#) on page 170-29):

```
DBMS_XMLDOM.MAKENODE (
    t       IN      DOMTEXT)
RETURN DOMNODE;
```

## Parameters

**Table 170–106** *MAKENODE Function Parameters*

Parameter	Description
a	DOMATTR to cast
cds	DOMCDATASECTION to cast
cd	DOMCHARACTERDATA to cast
com	DOMCOMMENT to cast
doc	DOMDOCUMENT to cast
df	DOMDOCUMENTFRAGMENT to cast
dt	DOMDOCUMENTTYPE to cast
elem	DOMELEMENT to cast
ent	DOMENTITY to cast
eref	DOMENTITYREFERENCE to cast
n	DOMNOTATION to cast
pi	DOMPROCESSINGINSTRUCTION to cast

**Table 170–106 (Cont.) MAKENODE Function Parameters**

<b>Parameter</b>	<b>Description</b>
t	DOMTEXT to cast

## MAKENOTATION Function

This function casts a specified DOMNODE to a DOMNOTATION, and returns the DOMNOTATION.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKENOTATION(  
    n          IN      DOMNODE)  
RETURN DOMNOTATION;
```

### Parameters

**Table 170–107** *MAKENOTATION Function Parameters*

Parameter	Description
n	DOMNODE to cast



## MAKEPROCESSINGINSTRUCTION Function

This function casts a specified DOMNODE to a DOMPROCESSINGINSTRUCTION, and returns the Domprocessinginstruction.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKEPROCESSINGINSTRUCTION (  
    n          IN      DOMNODE)  
RETURN DOMPROCESSINGINSTRUCTION;
```

### Parameters

**Table 170–108** MAKEPROCESSINGINSTRUCTION Function Parameters

Parameter	Description
n	DOMNODE to cast

## MAKETEXT Function

This function casts a specified DOMNODE to a DOMTEXT, and returns the DOMTEXT.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.MAKETEXT (  
    n          IN      DOMNODE)  
RETURN DOMTEXT;
```

### Parameters

**Table 170–109** *MAKETEXT Function Parameters*

Parameter	Description
n	DOMNODE to cast

## NEWDOMDOCUMENT Functions

This function returns a new DOMDOCUMENT instance.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

Returns a new DOMDOCUMENT instance:

```
DBMS_XMLDOM.NEWDOMDOCUMENT
RETURN DOMDOCUMENT;
```

Returns a new DOMDOCUMENT instance created from the specified XMLType object:

```
DBMS_XMLDOM.NEWDOMDOCUMENT (
xmlDoc    IN SYS.XMLTYPE)
RETURN DOMDOCUMENT;
```

Returns a new DOMDOCUMENT instance created from the specified CLOB:

```
DBMS_XMLDOM.NEWDOMDOCUMENT (
c1        IN CLOB)
RETURN DOMDOCUMENT;
```

### Parameters

**Table 170–110** *NEWDOMDOCUMENT Function Parameters*

Parameter	Description
xmlDoc	XMLType source for the DOMDOCUMENT
c1	CLOB source for the DOMDOCUMENT

## NORMALIZE Procedure

This procedure normalizes the text children of the DOMELEMENT.

**See Also:** [DOMELEMENT Subprograms](#) on page 170-21

### Syntax

```
DBMS_XMLDOM.NORMALIZE(  
    elem          IN      DOMELEMENT);
```

### Parameters

**Table 170–111** *NORMALIZE Procedure Parameters*

Parameter	Description
elem	The DOMELEMENT

## REMOVEATTRIBUTE Procedures

This procedure removes an attribute from the `DOMELEMENT` by name.

**See Also:** [DOMElement Subprograms](#) on page 170-21

### Syntax

Removes the value of a `DOMELEMENT`'s attribute by name:

```
DBMS_XMLDOM.REMOVEATTRIBUTE (
  elem      IN   DOMELEMENT,
  name      IN   VARCHAR2);
```

Removes the value of a `DOMELEMENT`'s attribute by name and namespace URI.

```
DBMS_XMLDOM.REMOVEATTRIBUTE (
  elem      IN   DOMELEMENT,
  name      IN   VARCHAR2,
  ns        IN   VARCHAR2);
```

### Parameters

**Table 170–112 REMOVEATTRIBUTE Procedure Parameters**

Parameter	Description
<code>elem</code>	The <code>DOMELEMENT</code>
<code>name</code>	Attribute name
<code>ns</code>	Namespace

## REMOVEATTRIBUTENODE Function

This function removes the specified attribute node from the `DOMELEMENT`. The method returns the removed node.

**See Also:** [DOMElement Subprograms](#) on page 170-21

### Syntax

```
DBMS_XMLDOM.REMOVEATTRIBUTENODE(  
    elem      IN      DOMELEMENT,  
    oldAttr   IN      DOMATTR)  
RETURN DOMAttr;
```

### Parameters

**Table 170–113 REMOVEATTRIBUTENODE Function Parameters**

Parameter	Description
<code>elem</code>	The <code>DOMELEMENT</code> .
<code>oldAttr</code>	The old <code>DOMATTR</code> .

## REMOVECHILD Function

This function removes the child node indicated by `oldchild` from the list of children, and returns it.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.REMOVECHILD(  
    n           IN      DOMNode,  
    oldchild   IN      DOMNode)  
RETURN DOMNODE;
```

### Parameters

**Table 170–114 REMOVECHILD Function Parameters**

Parameter	Description
<code>n</code>	DOMNode
<code>oldChild</code>	The child of the node <code>n</code> to be removed

## REMOVENAMEDITEM Function

This function removes from the map a node specified by name, and returns this node. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

**See Also:** [DOMNamedNodeMap Subprograms](#) on page 170-25

### Syntax

Removes a node specified by name:

```
DBMS_XMLDOM.REMOVENAMEDITEM (
  nnm      IN      DOMNamedNodeMap,
  name     IN      VARCHAR2)
RETURN DOMNode;
```

Removes a node specified by name and namespace URI:

```
DBMS_XMLDOM.REMOVENAMEDITEM (
  nnm      IN      DOMNamedNodeMap,
  name     IN      VARCHAR2,
  ns       IN      VARCHAR2)
RETURN DOMNode;
```

### Parameters

**Table 170–115** *REMOVENAMEDITEM Function Parameters*

Parameter	Description
nnm	DOMNamedNodeMap
name	The name of the item to be removed from the map
ns	Namespace



## REPLACECHILD Function

This function replaces the child node `oldchild` with `newchild` in the list of children, and returns the `oldchild` node. If `newchild` is a `DocumentFragment` object, `oldchild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newchild` is already in the tree, it is first removed.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.REPLACECHILD(
    n           IN      DOMNode,
    newchild   IN      DOMNode,
    oldchild   IN      DOMNode)
RETURN DOMNode;
```

### Parameters

**Table 170–116** *REPLACECHILD Function Parameters*

Parameter	Description
<code>n</code>	DOMNode
<code>newchild</code>	The new child which is to replace the old child
<code>oldchild</code>	The child of the node <code>n</code> which is to be replaced

## REPLACEDATA Procedure

This procedure changes a range of characters in the node. Upon success, data and length reflect the change.

**See Also:** [DOMCharacterData Subprograms](#) on page 170-15

### Syntax

```
DBMS_XMLDOM.REPLACEDATA (  
  cd          IN      DOMCHARACTERDATA,  
  offset      IN      NUMBER,  
  cnt         IN      NUMBER,  
  arg         IN      VARCHAR2);
```

### Parameters

**Table 170–117** *REPLACEDATA Procedure Parameters*

Parameter	Description
cd	DOMCHARACTERDATA
offset	The offset at which to replace
cnt	The number of characters to replace
arg	The value to replace with

## RESOLVENAMESPACEPREFIX Function

This function resolves the specified namespace prefix, and returns the resolved namespace.

**See Also:** [DOMELEMENT Subprograms](#) on page 170-21

### Syntax

```
DBMS_XMLDOM.RESOLVENAMESPACEPREFIX(  
    elem      IN      DOMELEMENT,  
    prefix    IN      VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–118** *RESOLVENAMESPACEPREFIX Function Parameters*

Parameter	Description
elem	The DOMELEMENT
prefix	Namespace prefix

## SETATTRIBUTE Procedures

Sets the value of a DOMELEMENT's attribute by name.

**See Also:** [DOMELEMENT Subprograms](#) on page 170-21

### Syntax

Sets the value of a DOMELEMENT's attribute by name:

```
DBMS_XMLDOM.SETATTRIBUTE (  
    elem      IN  DOMELEMENT,  
    name      IN  VARCHAR2,  
    newvalue  IN  VARCHAR2);
```

Sets the value of a DOMELEMENT's attribute by name and namespace URI:

```
DBMS_XMLDOM.SETATTRIBUTE (  
    elem      IN  DOMELEMENT,  
    name      IN  VARCHAR2,  
    newvalue  IN  VARCHAR2,  
    ns        IN  VARCHAR2);
```

### Parameters

**Table 170–119** SETATTRIBUTE Procedure Parameters

Parameter	Description
elem	The DOMELEMENT
name	Attribute name
newvalue	Attribute value
ns	Namespace

## SETATTRIBUTENODE Functions

Adds a new attribute node to the DOMELEMENT.

**See Also:** [DOMELEMENT Subprograms](#) on page 170-21

### Syntax

Adds a new attribute node to the DOMELEMENT:

```
DBMS_XMLDOM.SETATTRIBUTENODE (
    elem      IN  DOMELEMENT,
    newAttr   IN  DOMATTR)
RETURN DOMATTR;
```

Adds a new attribute node to the DOMELEMENT; namespace enabled:

```
DBMS_XMLDOM.SETATTRIBUTENODE (
    elem      IN  DOMELEMENT,
    newAttr   IN  DOMATTR,
    ns        IN  VARCHAR2)
RETURN DOMATTR;
```

### Parameters

**Table 170–120 SETATTRIBUTENODE Function Parameters**

Parameter	Description
elem	The DOMELEMENT
newAttr	The new DOMATTR
ns	The namespace

## SETCHARSET Procedure

This function sets the character set of the DOM document.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.SETCHARSET(  
  doc      IN      DOMDocument,  
  charset  IN      VARCHAR2);
```

### Parameters

**Table 170–121** SETCHARSET Procedure Parameters

Parameter	Description
doc	DOM document
charset	Character set

### Usage Notes

This is used for [WRITETOFILE Procedures](#) if not explicitly specified at that time.

## SETDATA Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Sets the character data of the node that implements this interface (See Also: [DOMCharacterData Subprograms](#) on page 170-15):

```
DBMS_XMLDOM.SETDATA (
  cd      IN      DOMCHARACTERDATA,
  data    IN      VARCHAR2);
```

Sets the content data of the DOMPROCESSINGINSTRUCTION (See Also: [DOMProcessingInstruction Subprograms](#) on page 170-15):

```
DBMS_XMLDOM.SETDATA (
  pi      IN      DOMPROCESSINGINSTRUCTION,
  data    IN      VARCHAR2);
```

### Parameters

**Table 170–122 SETDATA Procedure Parameters**

Parameter	Description
cd	DOMCHARACTERDATA
data	The data to which the node is set
pi	DOMPROCESSINGINSTRUCTION
data	New processing instruction content data

## SETDOCTYPE Procedure

Given a DOM document, this procedure creates a new DTD with the specified name, system id and public id and sets it in the document. This DTD can later be retrieved using the [GETDOCTYPE Function](#).

### Syntax

```
DBMS_XMLDOM.SETDOCTYPE(  
  doc      IN   DOMDocument,  
  name     IN   VARCHAR2,  
  sysid   IN   VARCHAR2,  
  pubid   IN   VARCHAR2);
```

### Parameters

**Table 170–123** *SETDOCTYPE Procedure Parameters*

Parameter	Description
doc	The document whose DTD has to be set
name	The name that the doctype needs to be initialized with
sysid	The system ID that the doctype needs to be initialized with
pubid	The public ID that the doctype needs to be initialized with



## SETNAMEDITEM Function

This function adds a node using its `nodeName` attribute. If a node with that name is already present in this map, it is replaced by the new one. The old node is returned on replacement; if no replacement is made, `NULL` is returned.

As the `nodeName` attribute is used to derive the name under which the node must be stored, multiple nodes of certain types, those that have a "special" string value, cannot be stored because the names would clash. This is seen as preferable to allowing nodes to be aliased.

**See Also:** [DOMNamedNodeMap Subprograms](#) on page 170-25

### Syntax

Adds a node using its `nodeName` attribute:

```
DBMS_XMLDOM.SETNAMEDITEM (
    nnm      IN      DOMNAMEDNODEMAP,
    arg      IN      DOMNODE)
RETURN DOMNode;
```

Adds a node using its `nodeName` attribute and namespace URI:

```
DBMS_XMLDOM.SETNAMEDITEM (
    nnm      IN      DOMNAMEDNODEMAP,
    arg      IN      DOMNODE,
    ns       IN      VARCHAR2)
RETURN DOMNode;
```

### Parameters

**Table 170–124 SETNAMEDITEM Function Parameters**

Parameter	Description
<code>nnm</code>	<code>DOMNAMEDNODEMAP</code>
<code>arg</code>	The Node to be added using its <code>nodeName</code> attribute
<code>ns</code>	Namespace

## SETNODEVALUE Procedure

This procedure sets the value of this node, depending on its type. When it is defined to be NULL, setting it has no effect.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.SETNODEVALUE (  
    n          IN      DOMNODE,  
    nodeValue IN      VARCHAR2);
```

### Parameters

**Table 170–125** SETNODEVALUE Procedure Parameters

Parameter	Description
n	DOMNode
nodeValue	The value to which node is set

## SETNODEVALUEASBINARYSTREAM Function & Procedure

The operation of these subprograms is described with each syntax implementation.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

This function returns an instance of the PL/SQL XMLBINARYOUTPUTSTREAM into which the caller can write the node value. The data type of the node must be RAW or BLOB – if not, an exception is raised.

```
DBMS_XMLDOM.SETNODEVALUEASBINARYSTREAM (
    n      IN      DOMNODE)
RETURN SYS.UTL_BINARYOUTPUTSTREAM;
```

Using this procedure, the application passes in an implementation of `sys.utl_BinaryInputStream` from which XDB reads data to populate the node. The data type of the node must be RAW or BLOB – if not an exception is raised.

```
DBMS_XMLDOM.SETNODEVALUEASBINARYSTREAM (
    n      in      DOMNODE,
    value  in      SYS.UTL_BINARYINPUTSTREAM);
```

### Parameters

**Table 170–126 SETNODEVALUEASBINARYSTREAM Function & Procedure Parameters**

Parameter	Description
n	DOMNODE
value	BINARYINPUTSTREAM

## SETNODEVALUEASCHARACTERSTREAM Function & Procedure

The operation of these subprograms is described with each syntax implementation.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

This function returns an instance of the PL/SQL XMLCHARACTEROUTPUTSTREAM type into which the caller can write the node value. The data type of the node can be any valid XDB data type. If the type is not character or CLOB, the character data written to the stream is converted to the node data type. If the data type of the node is character or CLOB, then the character data written to the stream is converted from PL/SQL session character set to the character set of the node.

```
DBMS_XMLDOM.SETNODEVALUEASCHARACTERSTREAM (
    n          IN      DOMNODE)
RETURN SYS.UTL_CHARACTEROUTPUTSTREAM;
```

Using this procedure, the application passes in an implementation of SYS.UTL\_CHARACTERINPUTSTREAM from which XDB reads to populate the node. The data type of the node may be any valid type supported by XDB. If a non-character data type, the character data read from the stream is converted to the data type of the node. If the data type of the node is either character or CLOB, then no conversion occurs and the character set of the node becomes the character set of the PL/SQL session.

```
DBMS_XMLDOM.SETNODEVALUEASCHARACTERSTREAM (
    n          IN      DOMNODE,
    value      IN      SYS.UTL_CHARACTERINPUTSTREAM);
```

### Parameters

**Table 170–127 SETNODEVALUEASCHARACTERSTREAM Function & Procedure Parameters**

Parameter	Description
n	DOMNODE
value	CHARACTERINPUTSTREAM

## SETPREFIX Procedure

This procedure sets the namespace prefix for this node to the specified value.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.SETPREFIX(  
    n          IN      DOMNODE,  
    prefix     IN      VARCHAR2);
```

### Parameters

**Table 170–128** *SETPREFIX Procedure Parameters*

Parameter	Description
n	DOMNODE
prefix	The value for the namespace prefix of the node

## SETSTANDALONE Procedure

This procedure sets the standalone property of the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.SETSTANDALONE(  
    doc          IN      DOMDOCUMENT,  
    newvalue     IN      VARCHAR2);
```

### Parameters

**Table 170–129** SETSTANDALONE Procedure Parameters

Parameter	Description
doc	DOMDOCUMENT
newvalue	Value of the standalone property of the document

## SETVALUE Procedure

This procedure sets the value of the attribute.

**See Also:** [DOMAttr Subprograms](#) on page 170-13

### Syntax

```
DBMS_XMLDOM.SETVALUE (  
  a          IN      DOMATTR,  
  value     IN      VARCHAR2);
```

### Parameters

**Table 170–130 SETVALUE Procedure Parameters**

Parameter	Description
a	DOMATTR
value	The value to which to set the attribute

## SETVERSION Procedure

This procedure sets the version of the DOMDOCUMENT.

**See Also:** [DOMDocument Subprograms](#) on page 170-17

### Syntax

```
DBMS_XMLDOM.SETVERSION(  
    doc          IN      DOMDOCUMENT,  
    version     IN      VARCHAR2);
```

### Parameters

**Table 170–131** SETVERSION Procedure Parameters

Parameter	Description
doc	DOMDOCUMENT
version	The version of the document



## SPLITTEXT Function

This function breaks this DOMTEXT node into two DOMTEXT nodes at the specified offset.

**See Also:** [DOMText Subprograms](#) on page 170-29

### Syntax

```
DBMS_XMLDOM.SPLITTEXT(  
    t          IN      DOMTEXT,  
    offset     IN      NUMBER)  
RETURN DOMText;
```

### Parameters

**Table 170–132** *SPLITTEXT Function Parameters*

Parameter	Description
t	DOMTEXT
offset	Offset at which to split

## SUBSTRINGDATA Function

This function extracts a range of data from the node.

**See Also:** [DOMCharacterData Subprograms](#) on page 170-15

### Syntax

```
DBMS_XMLDOM.SUBSTRINGDATA (  
    cd          IN      DOMCHARACTERDATA,  
    offset     IN      NUMBER,  
    cnt        IN      NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 170–133 SUBSTRINGDATA Function Parameters**

Parameter	Description
cd	DOMCHARACTERDATA
offset	The starting offset of the data from which to get the data
cnt	The number of characters (from the offset) of the data to get

## USEBINARYSTREAM Function

This function returns TRUE if the data type of the node is RAW or BLOB so that the node value may be read or written using an UTL\_BINARYINPUTSTREAM or UTL\_BINARYOUTPUTSTREAM. If a value of FALSE is returned, the node value may only be accessed through an UTL\_CHARACTERINPUTSTREAM or UTL\_CHARACTEROUTPUTSTREAM.

**See Also:** [DOMNode Subprograms](#) on page 170-11

### Syntax

```
DBMS_XMLDOM.USEBINARYSTREAM (
    n          IN      DOMNODE)
RETURN BOOLEAN;
```

### Parameters

**Table 170–134** *USEBINARYSTREAM Function Parameters*

Parameter	Description
n	DOMNODE

## WRITETOBUFFER Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Writes XML node to specified buffer using the database character set (See Also: [DOMNode Subprograms](#) on page 170-11):

```
DBMS_XMLDOM.WRITETOBUFFER (
    n          IN          DOMNODE,
    buffer     IN OUT     VARCHAR2);
```

Writes XML document to a specified buffer using database character set (See Also: [DOMDocument Subprograms](#) on page 170-17):

```
DBMS_XMLDOM.WRITETOBUFFER (
    doc        IN          DOMDOCUMENT,
    buffer     IN OUT     VARCHAR2);
```

Writes the contents of the specified document fragment into a buffer using the database character set (See Also: [DOMDocumentFragment Subprograms](#) on page 170-19):

```
DBMS_XMLDOM.WRITETOBUFFER (
    df         IN          DOMDOCUMENTFRAGMENT,
    buffer     IN OUT     VARCHAR2);
```

### Parameters

**Table 170–135** *WRITETOBUFFER Procedure Parameters*

Parameter	Description
n	DOMNODE
buffer	Buffer to which to write
doc	DOMDOCUMENT
df	DOM document fragment

## WRITETOCLOB Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Writes XML node to specified CLOB using the database character set (See Also: [DOMNode Subprograms](#) on page 170-11):

```
DBMS_XMLDOM.WRITETOCLOB (
  n          IN          DOMNODE,
  c1         IN OUT     CLOB);
```

Writes XML document to a specified CLOB using database character set (See Also: [DOMDocument Subprograms](#) on page 170-17):

```
DBMS_XMLDOM.WRITETOCLOB (
  doc       IN          DOMDOCUMENT,
  c1        IN OUT     CLOB);
```

### Parameters

**Table 170–136** *WRITETOCLOB Procedure Parameters*

Parameter	Description
n	DOMNODE
c1	CLOB to which to write
doc	DOMDOCUMENT

## WRITETOFILE Procedures

This procedure is overloaded. The specific forms of functionality are described along with the syntax declarations.

### Syntax

Writes XML node to specified file using the database character set (See Also: [DOMNode Subprograms](#)):

```
DBMS_XMLDOM.WRITETOFILE (
  n          IN      DOMNODE,
  fileName   IN      VARCHAR2);
```

Writes XML node to specified file using the specified character set, which is passed in as a separate parameter (See Also: [DOMNode Subprograms](#)):

```
DBMS_XMLDOM.WRITETOFILE (
  n          IN      DOMNODE,
  fileName   IN      VARCHAR2,
  charset    IN      VARCHAR2);
```

Writes an XML document to a specified file using database character set (See Also: [DOMDocument Subprograms](#)):

```
DBMS_XMLDOM.WRITETOFILE (
  doc        IN      DOMDOCUMENT,
  filename   IN      VARCHAR2);
```

Writes an XML document to a specified file using specified character set (See Also: [DOMDocument Subprograms](#)):

```
DBMS_XMLDOM.WRITETOFILE (
  doc        IN      DOMDOCUMENT,
  fileName   IN      VARCHAR2,
  charset    IN      VARCHAR2);
```

### Parameters

**Table 170–137 WRITETOFILE Procedure Parameters**

Parameter	Description
n	DOMNODE
fileName	File to which to write
charset	specified character set
doc	DOMDOCUMENT
charset	Character set

The `DBMS_XMLGEN` package converts the results of a SQL query to a canonical XML format. The package takes an arbitrary SQL query as input, converts it to XML format, and returns the result as a `CLOB`. This package is similar to the `DBMS_XMLQUERY` package, except that it is written in C and compiled into the kernel. This package can only be run on the database.

This chapter contains the following topic:

- [Using DBMS\\_XMLGEN](#)
- [Summary of DBMS\\_XMLGEN Subprograms](#)

**See Also:** *Oracle XML DB Developer's Guide*, for more information on XML support and on examples of using `DBMS_XMLGEN`

---

## Using DBMS\_XMLGEN

- [Security Model](#)



## Security Model

Owned by XDB, the DBMS\_XMLGEN package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XMLGEN Subprograms

**Table 171–1 Summary of DBMS\_XMLGEN Package Subprograms**

Subprogram	Description
<a href="#">CLOSECONTEXT Procedure</a> on page 171-5	Closes the context and releases all resources
<a href="#">CONVERT Functions</a> on page 171-6	Converts the XML into the escaped or unescaped XML equivalent
<a href="#">GETNUMROWSPROCESSED Function</a> on page 171-7	Gets the number of SQL rows that were processed in the last call to <a href="#">GETXML Functions</a>
<a href="#">GETXML Functions</a> on page 171-8	Gets the XML document
<a href="#">GETXMLTYPE Functions</a> on page 171-9	Gets the XML document and returns it as XMLType
<a href="#">NEWCONTEXT Functions</a> on page 171-10	Creates a new context handle
<a href="#">NEWCONTEXTFROMHIERARCHY Function</a> on page 171-11	Obtains a handle to use in the <a href="#">GETXML Functions</a> and other functions to get a hierarchical XML with recursive elements from the result
<a href="#">RESTARTQUERY Procedure</a> on page 171-12	Restarts the query to start fetching from the beginning
<a href="#">SETCONVERTSPECIALCHARS Procedure</a> on page 171-13	Sets whether special characters such as \$, which are non-XML characters, should be converted or not to their escaped representation
<a href="#">SETMAXROWS Procedure</a> on page 171-14	Sets the maximum number of rows to be fetched each time
<a href="#">SETNULLHANDLING Procedure</a> on page 171-15	Sets NULL handling options
<a href="#">SETROWSETTAG Procedure</a> on page 171-16	Sets the name of the element enclosing the entire result
<a href="#">SETROWTAG Procedure</a> on page 171-17	Sets the name of the element enclosing each row of the result
<a href="#">SETSKIPROWS Procedure</a> on page 171-18	Sets the number of rows to skip every time before generating the XML.
<a href="#">USEITEMTAGSFORCOLL Procedure</a> on page 171-19	Forces the use of the collection column name appended with the tag <code>_ITEM</code> for collection elements
<a href="#">USENULLATTRIBUTEINDICATOR Procedure</a> on page 171-20	Specifies whether to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document.

## CLOSECONTEXT Procedure

This procedure closes a given context and releases all resources associated with it, including the SQL cursor and bind and define buffers. After this call, the handle cannot be used for a subsequent function call.

### Syntax

```
DBMS_XMLGEN.CLOSECONTEXT (  
    ctx IN ctxHandle);
```

### Parameters

**Table 171–2** *CLOSECONTEXT Procedure Parameters*

Parameter	Description
ctx	The context handle to close.

## CONVERT Functions

This function converts the XML data into the escaped or unescapes XML equivalent, and returns XML CLOB data in encoded or decoded format. There are several version of the function.

### Syntax

Uses XMLDATA in string form (VARCHAR2):

```
DBMS_XMLGEN.CONVERT (
    xmlData IN VARCHAR2,
    flag    IN NUMBER := ENTITY_ENCODE)
RETURN VARCHAR2;
```

Uses XMLDATA in CLOB form:

```
DBMS_XMLGEN.CONVERT (
    xmlData IN CLOB,
    flag    IN NUMBER := ENTITY_ENCODE)
RETURN CLOB;
```

### Parameters

**Table 171–3** *CONVERT Function Parameters*

Parameter	Description
xmlData	The XML CLOB data to be encoded or decoded.
flag	The flag setting; ENTITY_ENCODE (default) for encode, and ENTITY_DECODE for decode.

### Usage Notes

This function escapes the XML data if the ENTITY\_ENCODE is specified. For example, the escaped form of the character < is &lt; ;. Unescaping is the reverse transformation.

## GETNUMROWSPROCESSED Function

This function retrieves the number of SQL rows processed when generating the XML using the [GETXML Functions](#) call. This count does not include the number of rows skipped before generating the XML. Note that [GETXML Functions](#) always generates an XML document, even if there are no rows present.

### Syntax

```
DBMS_XMLGEN.GETNUMROWSPROCESSED (  
    ctx      IN      ctxHandle)  
RETURN NUMBER;
```

### Parameters

**Table 171-4** *GETNUMROWSPROCESSED Function Parameters*

Parameter	Description
ctx	The context handle obtained from the <a href="#">NEWCONTEXT Functions</a> call on page 171-10.

### Usage Notes

This function is used to determine the terminating condition if calling [GETXML Functions](#) in a loop.

## GETXML Functions

This function gets the XML document. The function is overloaded.

### Syntax

Gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in. Use this version of [GETXML Functions](#) to avoid any extra CLOB copies and to reuse the same CLOB for subsequent calls. Because of the CLOB reuse, this [GETXML Functions](#) call is potentially more efficient:

```
DBMS_XMLGEN.GETXML (
    ctx          IN ctxHandle,
    tmpclob      IN OUT NCOPY CLOB,
    dtdOrSchema IN number := NONE)
RETURN BOOLEAN;
```

Generates the XML document and returns it as a temporary CLOB. The temporary CLOB obtained from this function must be freed using the `DBMS_LOB.FREETEMPORARY` call:

```
DBMS_XMLGEN.GETXML (
    ctx          IN ctxHandle,
    dtdOrSchema IN number := NONE)
RETURN CLOB;
```

Converts the results from the SQL query string to XML format, and returns the XML as a temporary CLOB, which must be subsequently freed using the `DBMS_LOB.FREETEMPORARY` call:

```
DBMS_XMLGEN.GETXML (
    sqlQuery     IN VARCHAR2,
    dtdOrSchema  IN number := NONE)
RETURN CLOB;
```

### Parameters

**Table 171–5** GETXML Function Parameters

Parameter	Description
<code>ctx</code>	The context handle obtained from the <code>newContext</code> call.
<code>tmpclob</code>	The CLOB to which the XML document is appended.
<code>sqlQuery</code>	The SQL query string.
<code>dtdOrSchema</code>	Generate a DTD or a schema? Only <code>NONE</code> is supported.

### Usage Notes

When the rows indicated by the [SETSKIPROWS Procedure](#) call are skipped, the maximum number of rows as specified by the [SETMAXROWS Procedure](#) call (or the entire result if not specified) is fetched and converted to XML. Use the [GETNUMROWSPROCESSED Function](#) to check if any rows were retrieved.

## GETXMLTYPE Functions

This function gets the XML document and returns it as an `XMLTYPE`. `XMLTYPE` operations can be performed on the results.

This function is overloaded.

### Syntax

Generates the XML document and returns it as a `sys.XMLType`:

```
DBMS_XMLGEN.GETXMLTYPE (
  ctx          IN ctxhandle,
  dtdOrSchema IN number := NONE)
RETURN sys.XMLType;
```

Converts the results from the SQL query string to XML format, and returns the XML as a `sys.XMLType`:

```
DBMS_XMLGEN.GETXMLTYPE (
  sqlQuery     IN VARCHAR2,
  dtdOrSchema IN number := NONE)
RETURN sys.XMLType
```

### Parameters

**Table 171–6** *GETXMLTYPE Function Parameters*

Parameter	Description
<code>ctx</code>	The context handle obtained from the <code>newContext</code> call.
<code>sqlQuery</code>	The SQL query string.
<code>dtdOrSchema</code>	Generate a DTD or a schema? Only <code>NONE</code> is supported.

## NEWCONTEXT Functions

This function generates and returns a new context handle. This context handle is used in [GETXML Functions](#) and other functions to get XML back from the result. There are several version of the function.

### Syntax

Generates a new context handle from a query:

```
DBMS_XMLGEN.NEWCONTEXT (  
    query      IN VARCHAR2)  
RETURN ctxHandle;
```

Generates a new context handle from a query string in the form of a PL/SQL ref cursor:

```
DBMS_XMLGEN.NEWCONTEXT (  
    queryString IN SYS_REFCURSOR)  
RETURN ctxHandle;
```

### Parameters

**Table 171–7** NEWCONTEXT Function Parameters

Parameter	Description
query	The query, in the form of a VARCHAR, the result of which must be converted to XML.
queryString	The query string in the form of a PL/SQL ref cursor, the result of which must be converted to XML.



## NEWCONTEXTFROMHIERARCHY Function

This function obtains a handle to use in the [GETXML Functions](#) and other functions to get a hierarchical XML with recursive elements from the result.

### Syntax

```
DBMS_XMLGEN.NEWCONTEXTFROMHIERARCHY (  
    queryString IN VARCHAR2)  
RETURN ctxHandle;
```

### Parameters

**Table 171–8** *NEWCONTEXTFROMHIERARCHY Function Parameters*

Parameter	Description
queryString	The query string, the result of which must be converted to XML. The query is a hierarchical query typically formed using a <code>CONNECT BY</code> clause, and the result must have the same property as the result set generated by a <code>CONNECT BY</code> query. The result set must have only two columns, the level number and an XML value. The level number is used to determine the hierarchical position of the XML value within the result XML document.

## RESTARTQUERY Procedure

This procedure restarts the query and generates the XML from the first row. It can be used to start executing the query again, without having to create a new context.

### Syntax

```
DBMS_XMLGEN.RESTARTQUERY (  
  ctx IN ctxHandle);
```

### Parameters

**Table 171–9** *RESTARTQUERY Procedure Parameters*

Parameter	Description
ctx	The context handle corresponding to the current query.

## SETCONVERTSPECIALCHARS Procedure

This procedure sets whether or not special characters in the XML data must be converted into their escaped XML equivalent. For example, the < sign is converted to &lt;. The default is to perform conversions. This function improves performance of XML processing when the input data cannot contain any special characters such as <, >, " , ' , which must be escaped. It is expensive to scan the character data to replace the special characters, particularly if it involves a lot of data.

### Syntax

```
DBMS_XMLGEN.SETCONVERTSPECIALCHARS (  
  ctx   IN ctxHandle,  
  conv  IN BOOLEAN);
```

### Parameters

**Table 171–10** SETCONVERTSPECIALCHARS Procedure Parameters

Parameter	Description
ctx	The context handle obtained from one of the <a href="#">NEWCONTEXT Functions</a> call on page 171-10.
conv	TRUE indicates that conversion is needed.

## SETMAXROWS Procedure

This procedure sets the maximum number of rows to fetch from the SQL query result for every invocation of the [GETXML Functions](#) call. It is used when generating paginated results. For example, when generating a page of XML or HTML data, restrict the number of rows converted to XML or HTML by setting the `maxrows` parameter.

### Syntax

```
DBMS_XMLGEN.SETMAXROWS (  
  ctx      IN ctxHandle,  
  maxRows  IN NUMBER);
```

### Parameters

**Table 171–11** *SETMAXROWS Procedure Parameters*

Parameter	Description
<code>ctx</code>	The context handle corresponding to the query executed.
<code>maxRows</code>	The maximum number of rows to get for each call to <a href="#">GETXML Functions</a>

## SETNULLHANDLING Procedure

This procedure sets NULL handling options, handled through the `flag` parameter setting.

### Syntax

```
DBMS_XMLGEN.SETNULLHANDLING(  
  ctx IN ctx,  
  flag IN NUMBER);
```

### Parameters

**Table 171–12** *SETNULLHANDLING Procedure Parameters*

Parameter	Description
<code>ctx</code>	The context handle corresponding to the query executed.
<code>flag</code>	The NULL handling option set. <ul style="list-style-type: none"><li>■ <code>DROP_NULLS CONSTANT NUMBER := 0;</code> (Default) Leaves out the tag for NULL elements.</li><li>■ <code>NULL_ATTR CONSTANT NUMBER := 1;</code> Sets <code>xsi:nil="true"</code>.</li><li>■ <code>EMPTY_TAG CONSTANT NUMBER := 2;</code> Sets, for example, <code>&lt;foo/&gt;</code>.</li></ul>

## SETROWSETTAG Procedure

This procedure sets the name of the root element of the document. The default name is ROWSET .

### Syntax

```
DBMS_XMLGEN.SETROWSETTAG (  
  ctx          IN ctxHandle,  
  rowSetTagName IN VARCHAR2);
```

### Parameters

**Table 171–13 SETROWSETTAG Procedure Parameters**

Parameter	Description
ctx	The context handle obtained from the <a href="#">NEWCONTEXT Functions</a> call on page 171-10.
rowSetTagName	The name of the document element. Passing NULL indicates that you do not want the ROWSET element present.

### Usage Notes

The user can set the rowSetTag to NULL to suppress the printing of this element. However, an error is produced if both the row and the rowset are NULL and there is more than one column or row in the output . This is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

## SETROWTAG Procedure

This procedure sets the name of the element separating all the rows. The default name is ROW.

### Syntax

```
DBMS_XMLGEN.SETROWTAG (  
  ctx          IN ctxHandle,  
  rowTagName  IN VARCHAR2);
```

### Parameters

**Table 171–14** *SETROWTAG Procedure Parameters*

Parameter	Description
ctx	The context handle obtained from the <a href="#">NEWCONTEXT Functions</a> call on page 171-10.
rowTagName	The name of the ROW element. Passing NULL indicates that you do not want the ROW element present.

### Usage Notes

The user can set the name of the element to NULL to suppress the ROW element itself. However, an error is produced if both the row and the rowset are NULL and there is more than one column or row in the output. This is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

## SETSKIPROWS Procedure

This procedure skips a given number of rows before generating the XML output for every call to the [GETXML Functions](#). It is used when generating paginated results for stateless Web pages using this utility. For example, when generating the first page of XML or HTML data, set `skiprows` to zero. For the next set, set the `skiprows` to the number of rows obtained in the first case. See [GETNUMROWSPROCESSED Function](#) on page 171-7.

### Syntax

```
DBMS_XMLGEN.SETSKIPROWS (  
  ctx          IN ctxHandle,  
  skipRows    IN NUMBER);
```

### Parameters

**Table 171–15** *SETSKIPROWS Procedure Parameters*

Parameter	Description
<code>ctx</code>	The context handle corresponding to the query executed.
<code>skipRows</code>	The number of rows to skip for each call to <code>getXML</code> .



## USEITEMTAGSFORCOLL Procedure

This procedure overrides the default name of the collection elements. The default name for collection elements is the type name itself.

### Syntax

```
DBMS_XMLGEN.USEITEMTAGSFORCOLL (  
    ctx IN ctxHandle);
```

### Parameters

**Table 171–16** *USEITEMTAGSFORCOLL Procedure Parameters*

Parameter	Description
ctx	The context handle.

### Usage Notes

Using this procedure, you can override the default to use the name of the column with the `_ITEM` tag appended to it. If there is a collection of `NUMBER`, the default tag name for the collection elements is `NUMBER`.

## USENULLATTRIBUTEINDICATOR Procedure

This procedure specifies whether to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document. It is used as a shortcut for the [SETNULLHANDLING Procedure](#).

### Syntax

```
DBMS_XMLGEN.USENULLATTRIBUTEINDICATOR(  
  ctx          IN   ctxType,  
  attrind     IN   BOOLEAN := TRUE);
```

### Parameters

**Table 171–17** *USENULLATTRIBUTEINDICATOR Procedure Parameters*

Parameter	Description
ctx	Context handle.
attrind	Use attribute to indicate NULL?

The DBMS\_XMLINDEX package provides an interface to implement asynchronous indexing.

**See Also:** *Oracle XML DB Developer's Guide* for more information about "XMLIndex"

This chapter contains the following topics:

- [Using DBMS\\_XMLINDEX](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_XMLINDEX Subprograms](#)

## Using DBMS\_XMLINDEX

- [Overview](#)
- [Security Model](#)

## Overview

### **Asynchronous Index Maintenance**

The basic XMLIndex is maintained on every DML operation. However, given the computing costs, in many cases the availability of stale result is adequate. In such situations, it is desirable to defer index updates to a convenient time, for example to a time when the load on the database is low. Thus a mechanism for asynchronous index maintenance is provided.

## Security Model

Owned by XDB, the DBMS\_XMLINDEX package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XMLINDEX Subprograms

This table lists the package subprograms in alphabetical order.

**Table 172-1 DBMS\_XMLINDEX Package Subprograms**

Subprogram	Description
<a href="#">CREATEDATEINDEX Procedure</a> on page 172-6	Creates a secondary index for date values in the VALUE column of a PATH TABLE which is the storage table of an XMLIndex
<a href="#">CREATENUMBERINDEX Procedure</a> on page 172-7	Creates a secondary index for number values in the VALUE column of a PATH TABLE which is the storage table of an XMLIndex
<a href="#">DROPPARAMETER Procedure</a> on page 172-8	Drops the XMLIndex parameter string that is associated with a given parameter identifier.
<a href="#">MODIFYPARAMETER Procedure</a> on page 172-9	Modifies the XMLIndex parameter string that is associated with a given parameter name
<a href="#">REGISTERPARAMETER Procedure</a> on page 172-10	Registers a parameter string and XMLIndex parameter string pair in XDB
<a href="#">SYNCINDEX Procedure</a> on page 172-11	Synchronizes the index manually

## CREATEDATEINDEX Procedure

This procedure creates a secondary index for date values in the VALUE column of a PATH TABLE which is the storage table of an XMLIndex. The second form of the procedure allows for the date\_index\_clause to be set to an empty string.

### Syntax

```
DBMS_XMLINDEX.CREATEDATEINDEX (
  xml_index_schema IN VARCHAR2,
  xml_index_name   IN VARCHAR2,
  date_index_name  IN VARCHAR2,
  xmltypename     IN VARCHAR2,
  date_index_clause IN VARCHAR2);
```

```
DBMS_XMLINDEX.CREATEDATEINDEX (
  xml_index_schema IN VARCHAR2,
  xml_index_name   IN VARCHAR2,
  date_index_name  IN VARCHAR2,
  xmltypename     IN VARCHAR2);
```

### Parameters

**Table 172–2** *CREATEDATEINDEX Procedure Parameters*

Parameter	Description
xml_index_schema	Name of the owner of the XMLIndex
xml_index_name	Name of the XMLIndex
date_index_name	Name of the secondary index to be created for date values in the VALUE column of the PATH TABLE of XMLIndex named xml_index_name and owned by xml_index_schema
xmltypename	The type to which values in the VALUE column of the path table are to be cast. Acceptable values are the following strings: DATETIME, TIME, DATE, GDAY, GMONTH, GYEAR, GYEARMONTH, GMONTHDAY.
date_index_clause	Storage clause to be applied to the date index during its creation. This is a string argument appended to the CREATE INDEX statement for creating the date index



## CREATENUMBERINDEX Procedure

This procedure creates a secondary index for number values in the `VALUE` column of a `PATH TABLE` which is the storage table of an `XMLIndex`.

### Syntax

```
DBMS_XMLINDEX.CREATENUMBERINDEX (
  xml_index_schema IN VARCHAR2,
  xml_index_name   IN VARCHAR2,
  num_index_name   IN VARCHAR2,
  num_index_clause IN VARCHAR2,
  xmltypename     IN VARCHAR2);
```

### Parameters

**Table 172–3** *CREATENUMBERINDEX Procedure Parameters*

Parameter	Description
<code>xml_index_schema</code>	Name of the owner of the <code>XMLIndex</code>
<code>xml_index_name</code>	Name of the <code>XMLIndex</code>
<code>num_index_name</code>	Name of the secondary index to be created for number values in the <code>VALUE</code> column of the <code>PATH TABLE</code> of <code>XMLIndex</code> named <code>xml_index_name</code> and owned by <code>xml_index_schema</code>
<code>num_index_clause</code>	Storage clause to be applied to the number index during its creation. This is a string argument appended to the <code>CREATE INDEX</code> statement for creating the number index.
<code>xmltypename</code>	The type to which values in the <code>VALUE</code> column of the path table are to be cast. Acceptable values are the following strings: <code>FLOAT</code> , <code>DOUBLE</code> , <code>DECIMAL</code> , <code>INTEGER</code> , <code>NONPOSITIVEINTEGER</code> , <code>NEGATIVEINTEGER</code> , <code>LONG</code> , <code>INT</code> , <code>SHORT</code> , <code>BYTE</code> , <code>NONNEGATIVEINTEGER</code> , <code>UNSIGNEDLONG</code> , <code>UNSIGNEDINT</code> , <code>UNSIGNEDSHORT</code> , <code>UNSIGNEDBYTE</code> , <code>POSITIVEINTEGER</code> .

## DROPPARAMETER Procedure

This procedure drops the XMLIndex parameter string that is associated with a given parameter identifier.

### Syntax

```
DBMS_XMLINDEX.DROPPARAMETER (  
    name          IN          VARCHAR2);
```

### Parameters

**Table 172–4** DROPPARAMETER Procedure Parameters

Parameter	Description
name	Identifier for parameter string

### Examples

```
DBMS_XMLINDEX.DROPPARAMETER (  
    'myIndexParam');
```

## MODIFYPARAMETER Procedure

This procedure modifies the XMLIndex parameter string that is associated with a given parameter identifier.

### Syntax

```
DBMS_XMLINDEX.MODIFYPARAMETER (
  name          IN      VARCHAR2,
  parameter     IN      CLOB);
```

### Parameters

**Table 172–5** *MODIFYPARAMETER Procedure Parameters*

Parameter	Description
name	Identifier for parameter string
parameter	XMLIndex parameter clause that can appear in a CREATE INDEX or an ALTER INDEX statement

### Examples

```
DBMS_XMLINDEX.MODIFYPARAMETER (
  'myIndexParam',
  'PATH TABLE po_ptab
  PATH ID INDEX po_pidx
  ORDER KEY INDEX po_oidx
  VALUE INDEX po_vidx');
```

## REGISTERPARAMETER Procedure

This procedure registers a parameter identifier and XMLIndex parameter string pair in XDB.

### Syntax

```
DBMS_XMLINDEX.REGISTERPARAMETER (
  name          IN          VARCHAR2,
  parameter     IN          CLOB);
```

### Parameters

**Table 172–6 REGISTERPARAMETER Procedure Parameters**

Parameter	Description
name	Identifier for parameter string
parameter	XMLIndex parameter clause that can appear in a CREATE INDEX or an ALTER INDEX statement

### Examples

```
DBMS_XMLINDEX.REGISTERPARAMETER (
  'myIndexParam',
  'PATH TABLE po_ptab
  PATH ID INDEX po_pidx
  ORDER KEY INDEX po_oidx
  VALUE INDEX po_vidx
  PATHS (NAMESPACE MAPPING (xmlns:p="http://www.example.com/IPO"))
  GROUP MASTERGROUP XMLTABLE PO_TAB
  ('/p:PurchaseOrder'
   COLUMNS
     REFERENCE VARCHAR2(30) PATH 'p:Reference',
     REQUESTOR VARCHAR2(30) PATH 'p:Requestor' )
  GROUP ITEMGROUP XMLTABLE ITEMGROUP_TAB
  ('/p:PurchaseOrder/p:LineItems/p:LineItem'
   COLUMNS
     LINENUMBER NUMBER(38) PATH '@p:ItemNumber',
     QUANTITY NUMBER(38) PATH '@p:Quantity',
     DESCRIPTION VARCHAR2(256) PATH 'p:Description' ));
```

## SYNCINDEX Procedure

This function synchronizes an asynchronously maintained XMLIndex. It applies to the XMLIndex changes that are logged in the pending table, and brings the path table up-to-date with the base XMLTYPE column.

### Syntax

```
DBMS_XMLINDEX.SYNCINDEX (
  xml_index_schema    IN VARCHAR2,
  xml_index_name      IN VARCHAR2,
  partition_name      IN VARCHAR2 DEFAULT NULL,
  reindex             IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 172–7 SYNCINDEX Procedure Parameters**

Parameter	Description
xml_index_schema	Name of the owner of the XMLIndex
xml_schema_name	Name of the XMLIndex
partition_name	[Currently not supported]
reindex	Default is FALSE. If set to TRUE, this drops the secondary indexes and recreates them later so that they can be bulkloaded.

### Examples

```
EXEC DBMS_XMLINDEX.SYNCINDEX('USER1', 'SS_TAB_XMLI', REINDEX=>TRUE);
```



---

---

## DBMS\_XMLPARSER

Using `DBMS_XMLPARSER`, you can access the contents and structure of XML documents. XML describes a class of data XML document objects. It partially describes the behavior of computer programs which process them. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application. This PL/SQL implementation of the XML processor (or parser) follows the W3C XML specification REC-xml-19980210 and includes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The default behavior for this PL/SQL XML parser is to build a parse tree that can be accessed by DOM APIs, validate it if a DTD is found (otherwise, it is non-validating), and record errors if an error log is specified. If parsing fails, an application error is raised.

This chapter contains the following topics:

- [Using DBMS\\_XMLPARSER](#)
- [Summary of DBMS\\_XMLPARSER Subprograms](#)

**See Also:** *Oracle XML DB Developer's Guide*

## Using DBMS\_XMLPARSER

- [Security Model](#)



## Security Model

Owned by XDB, the DBMS\_XDB\_XMLPARSER package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XMLPARSER Subprograms

**Table 173–1 DBMS\_XMLPARSER Package Subprograms**

<b>Method</b>	<b>Description</b>
<a href="#">FREEPARSER</a> on page 173-5	Frees a parser object.
<a href="#">GETDOCTYPE</a> on page 173-6	Gets parsed DTD.
<a href="#">GETDOCUMENT</a> on page 173-7	Gets DOM document.
<a href="#">GETRELEASEVERSION</a> on page 173-7	Returns the release version of Oracle XML Parser for PL/SQL.
<a href="#">GETVALIDATIONMODE</a> on page 173-9	Returns validation mode.
<a href="#">NEWPARSER</a> on page 173-10	Returns a new parser instance
<a href="#">PARSE</a> on page 173-11	Parses XML stored in the given url/file.
<a href="#">PARSEBUFFER</a> on page 173-12	Parses XML stored in the given buffer
<a href="#">PARSECLOB</a> on page 173-12	Parses XML stored in the given clob
<a href="#">PARSEDTD</a> on page 173-14	Parses DTD stored in the given url/file
<a href="#">PARSEDTDBUFFER</a> on page 173-15	Parses DTD stored in the given buffer
<a href="#">PARSEDTDCLOB</a> on page 173-16	Parses DTD stored in the given clob
<a href="#">SETBASEDIR</a> on page 173-17	Sets base directory used to resolve relative URLs.
<a href="#">SETDOCTYPE</a> on page 173-18	Sets DTD.
<a href="#">SETERRORLOG</a> on page 173-19	Sets errors to be sent to the specified file
<a href="#">SETPRESERVEWHITESPACE</a> on page 173-20	Sets white space preserve mode
<a href="#">SETVALIDATIONMODE</a> on page 173-21	Sets validation mode.
<a href="#">SHOWWARNINGS</a> on page 173-22	Turns warnings on or off.

## FREEPARSER

Frees a parser object.

### Syntax

```
PROCEDURE freeParser(  
    p Parser);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

## GETDOCTYPE

Returns the parsed DTD; this function must be called only after a DTD is parsed.

### Syntax

```
FUNCTION getDoctype(  
  p Parser)  
RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

## GETDOCUMENT

Returns the document node of a DOM tree document built by the parser; this function must be called only after a document is parsed.

### Syntax

```
FUNCTION GETDOCUMENT (  
  p Parser)  
RETURN DOMDocument;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

## GETRELEASEVERSION

Returns the release version of the Oracle XML parser for PL/SQL.

### Syntax

```
FUNCTION getReleaseVersion  
RETURN VARCHAR2;
```

## GETVALIDATIONMODE

Retrieves validation mode; TRUE for validating, FALSE otherwise.

### Syntax

```
FUNCTION GETVALIDATIONMODE(  
  p Parser)  
RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

## NEWPARSER

Returns a new parser instance. This function must be called before the default behavior of Parser can be changed and if other parse methods need to be used.

### **Syntax**

```
FUNCTION newParser  
RETURN Parser;
```



## PARSE

Parses XML stored in the given URL or file. An application error is raised if parsing fails. There are several versions of this method.

Syntax	Description
<pre>FUNCTION parse(     url VARCHAR2) RETURN DOMDocument;  PROCEDURE parse(     p    Parser,     url VARCHAR2);</pre>	<p>Returns the built DOM Document.</p> <p>This is meant to be used when the default parser behavior is acceptable and just a url/file needs to be parsed.</p> <p>Any changes to the default parser behavior should be effected before calling this procedure.</p>

Parameter	IN / OUT	Description
url	(IN)	Complete path of the url/file to be parsed.
p	(IN)	Parser instance.

## PARSEBUFFER

Parses XML stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

### Syntax

```
PROCEDURE PARSEBUFFER(  
  p Parser,  
  doc VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

## PARSECLOB

Parses XML stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

### Syntax

```
PROCEDURE PARSECLOB(  
  p  Parser,  
  doc CLOB);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

## PARSEDTD

Parses the DTD stored in the given URL or file. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

### Syntax

```
PROCEDURE PARSEDTD(  
  p      Parser,  
  url    VARCHAR2,  
  root   VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
url	(IN)	Complete path of the URL or file to be parsed.
root	(IN)	Name of the root element.

## PARSEDTDBUFFER

Parses the DTD stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

### Syntax

```
PROCEDURE PARSEDTDBUFFER(  
  p    Parser,  
  dtd  VARCHAR2,  
  root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD buffer to parse.
root	(IN)	Name of the root element.

## PARSEDTCLOB

Parses the DTD stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

### Syntax

```
PROCEDURE PARSEDTCLOB(  
  p    Parser,  
  dtd  CLOB,  
  root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD Clob to parse.
root	(IN)	Name of the root element.

## SETBASEDIR

Sets base directory used to resolve relative URLs. An application error is raised if parsing fails.

### Syntax

```
PROCEDURE setBaseDir(  
  p Parser,  
  dir VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dir	(IN)	Directory used as a base directory.

## SETDOCTYPE

Sets a DTD to be used by the parser for validation. This call should be made before the document is parsed.

### Syntax

```
PROCEDURE setDoctype(  
  p Parser,  
  dtd DOMDocumentType);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD to set.



## SETERRORLOG

Sets errors to be sent to the specified file.

### Syntax

```
PROCEDURE setErrorLog(  
p          Parser,  
fileName VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
fileName	(IN)	Complete path of the file to use as the error log.

## SETPRESERVEWHITESPACE

Sets whitespace preserving mode.

### Syntax

```
PROCEDURE setPreserveWhitespace(  
  p Parser,  
  yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - preserve, FALSE - don't preserve.

## SETVALIDATIONMODE

Sets validation mode.

### Syntax

```
PROCEDURE setValidationMode(  
  p  Parser,  
  yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - validate, FALSE - don't validate.

## SHOWWARNINGS

Turns warnings on or off.

### Syntax

```
PROCEDURE showWarnings(  
  p Parser,  
  yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - show warnings, FALSE - don't show warnings.

---

---

## DBMS\_XMLQUERY

DBMS\_XMLQUERY provides database-to-XMLType functionality. Whenever possible, use DBMS\_XMLGEN, a built-in package in C, instead of DBMS\_XMLQUERY.

**See Also:** *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_XMLQUERY](#)
  - Security Model
  - Constants
  - Types
- [Summary of DBMS\\_XMLQUERY Subprograms](#)

## Using DBMS\_XMLQUERY

- [Security Model](#)
- [Constants](#)
- [Types](#)

## Security Model

Owned by XDB, the DBMS\_XMLQUERY package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Constants

**Table 174–1 Constants of DBMS\_XMLQUERY**

Constant	Description
DB_ENCODING	Used to signal that the DB character encoding is to be used.
DEFAULT_ROWSETTAG	The tag name for the element enclosing the XML generated from the result set (that is, for most cases the root node tag name) -- ROWSET.
DEFAULT_ERRORTAG	The default tag to enclose raised errors -- ERROR.
DEFAULT_ROWIDATTR	The default name for the cardinality attribute of XML elements corresponding to <code>db.records</code> -- NUM
DEFAULT_ROWTAG	The default tag name for the element corresponding to <code>db.records</code> -- ROW
DEFAULT_DATE_FORMAT	Default date mask -- 'MM/dd/yyyy HH:mm:ss'
ALL_ROWS	Indicates that all rows are needed in the output.
NONE	Used to specifies that the output should not contain any XML metadata (for example, no DTD).
DTD	Used to specify that the generation of the DTD is desired.
SCHEMA	Used to specify that the generation of the XML Schema is desired.
LOWER_CASE	Use lower case tag names.
UPPER_CASE	Use upper case tag names.



## Types

**Table 174–2** *Types of DBMS\_XMLQUERY*

Type	Description
ctxType	The type of the query context handle. This is the return type of <a href="#">NEWCONTEXT</a>

---

## Summary of DBMS\_XMLQUERY Subprograms

**Table 174–3 DBMS\_XMLQUERY Package Subprograms**

Method	Description
<a href="#">CLOSECONTEXT</a> on page 174-8	Closes or deallocates a particular query context.
<a href="#">GETDTD</a> on page 174-9	Generates the DTD.
<a href="#">GETEXCEPTIONCONTENT</a> on page 174-10	Returns the thrown exception's error code and error message.
<a href="#">GETNUMROWSPROCESSED</a> on page 174-11	Returns the number of rows processed for the query.
<a href="#">GETVERSION</a> on page 174-12	Prints the version of the XSU in use.
<a href="#">GETXML</a> on page 174-13	Generates the XML document.
<a href="#">NEWCONTEXT</a> on page 174-14	Creates a query context and it returns the context handle.
<a href="#">PROPAGATEORIGINALEXCEPTION</a> on page 174-15	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather than, wrapping it with an <code>OracleXMLSQLException</code> .
<a href="#">REMOVEXSLTPARAM</a> on page 174-16	Removes a particular top-level stylesheet parameter.
<a href="#">SETBINDVALUE</a> on page 174-17	Sets a value for a particular bind name.
<a href="#">SETCOLLIDATTRNAME</a> on page 174-18	Sets the name of the id attribute of the collection element's separator tag.
<a href="#">SETDATAHEADER</a> on page 174-18	Sets the XML data header.
<a href="#">SETDATEFORMAT</a> on page 174-20	Sets the format of the generated dates in the XML document.
<a href="#">SETENCODINGTAG</a> on page 174-21	Sets the encoding processing instruction in the XML document.
<a href="#">SETERRORTAG</a> on page 174-22	Sets the tag to be used to enclose the XML error documents.
<a href="#">SETMAXROWS</a> on page 174-23	Sets the maximum number of rows to be converted to XML.
<a href="#">SETMETAHEADER</a> on page 174-24	Sets the XML meta header.
<a href="#">SETRAISEEXCEPTION</a> on page 174-25	Tells the XSU to throw the raised exceptions.
<a href="#">SETRAISENOROWSEXCEPTION</a> on page 174-26	Tells the XSU to throw or not to throw an <code>OracleXMLNoRowsException</code> in the case when for one reason or another, the XML document generated is empty.
<a href="#">SETROWIDATTRNAME</a> on page 174-27	Sets the name of the id attribute of the row enclosing tag.
<a href="#">SETROWIDATTRVALUE</a> on page 174-28	Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.
<a href="#">SETROWSETTAG</a> on page 174-29	Sets the tag to be used to enclose the XML dataset.
<a href="#">SETROWTAG</a> on page 174-30	Sets the tag to be used to enclose the XML element.

**Table 174-3 (Cont.) DBMS\_XMLQUERY Package Subprograms**

<b>Method</b>	<b>Description</b>
<a href="#">SETSKIPROWS</a> on page 174-31	Sets the number of rows to skip.
<a href="#">SETSQLTOXMLNAMEESCAPING</a> on page 174-32	This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.
<a href="#">SETSTYLESHEETHEADER</a> on page 174-33	Sets the stylesheet header.
<a href="#">SETTAGCASE</a> on page 174-34	Specifies the case of the generated XML tags.
<a href="#">SETXSLT</a> on page 174-35	Registers a stylesheet to be applied to generated XML.
<a href="#">SETXSLTPARAM</a> on page 174-36	Sets the value of a top-level stylesheet parameter.
<a href="#">USENULLATTRIBUTEINDICATOR</a> on page 174-37	Specifies whether to use an XML attribute to indicate NULLness.
<a href="#">USETYPEFORCOLLELEMTAG</a> on page 174-38	Tells the XSU to use the collection element's type name as the collection element tag name.

## CLOSECONTEXT

Closes or deallocates a particular query context

### Syntax

```
PROCEDURE CLOSECONTEXT(  
  ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## GETDTD

Generates and returns the DTD based on the SQL query used to initialize the context. The options are described in the following table.

Syntax	Description
<pre>FUNCTION GETDTD(     ctxHdl IN ctxType,     withVer IN BOOLEAN := false) RETURN CLOB;</pre>	Function that generates the DTD based on the SQL query used to initialize the context.
<pre>PROCEDURE GETDTD(     ctxHdl IN ctxType,     xDoc IN CLOB,     withVer IN BOOLEAN := false);</pre>	Procedure that generates the DTD based on the SQL query used to initialize the context; specifies the output CLOB for XML document result.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
withVer	(IN)	Generate the version information? TRUE for yes.
xDoc	(IN)	CLOB into which to write the generated XML document.

## GETEXCEPTIONCONTENT

Returns the thrown exception's SQL error code and error message through the procedure's OUT parameters. This procedure is a work around the JVM functionality that obscures the original exception by its own exception, rendering PL/SQL unable to access the original exception content.

### Syntax

```
PROCEDURE GETEXCEPTIONCONTENT(  
  ctxHdl IN ctxType,  
  errNo OUT NUMBER,  
  errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(OUT)	Error number.
errMsg	(OUT)	Error message.

## GETNUMROWSPROCESSED

Return the number of rows processed for the query.

### Syntax

```
FUNCTION GETNUMROWSPROCESSED(  
  ctxHdl IN ctxType)  
RETURN NUMBER;
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## GETVERSION

Prints the version of the XSU in use.

### **Syntax**

```
PROCEDURE GETVERSION();
```



## GETXML

Creates the new context, executes the query, gets the XML back and closes the context. This is a convenience function. The context doesn't have to be explicitly opened or closed. The options are described in the following table.

Syntax	Description
<pre>FUNCTION GETXML(     sqlQuery IN VARCHAR2,     metaType IN NUMBER := NONE) RETURN CLOB;</pre>	This function uses a SQL query in string form.
<pre>FUNCTION GETXML(     sqlQuery IN CLOB,     metaType IN NUMBER := NONE) RETURN CLOB;</pre>	This function uses a SQL query in CLOB form.
<pre>FUNCTION GETXML(     ctxHdl IN ctxType,     metaType IN NUMBER := NONE) RETURN CLOB;</pre>	This function generates the XML document based on a SQL query used to initialize the context.
<pre>PROCEDURE GETXML(     ctxHdl IN ctxType,     xDoc IN CLOB,     metaType IN NUMBER := NONE);</pre>	This procedure generates the XML document based on the SQL query used to initialize the context.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
metaType	(IN)	XML metadata type (NONE, DTD, or SCHEMA).
sqlQuery	(IN)	SQL query.
xDoc	(IN)	CLOB into which to write the generated XML document.

## NEWCONTEXT

Creates a query context and it returns the context handle. The options are described in the following table.

<b>Syntax</b>	<b>Description</b>
<pre>FUNCTION NEWCONTEXT(     sqlQuery IN VARCHAR2) RETURN ctxType;  FUNCTION NEWCONTEXT(     sqlQuery IN CLOB) RETURN ctxType;</pre>	<p>Creates a query context from a string.</p> <p>Creates a query context from a CLOB.</p>

<b>Parameter</b>	<b>IN / OUT</b>	<b>Description</b>
<code>sqlQuery</code>	(IN)	SQL query, the results of which to convert to XML.

## PROPAGATEORIGINALEXCEPTION

Specifies whether to throw every original exception raised or to wrap it in an OracleXMLSQLException.

### Syntax

```
PROCEDURE PROPAGATEORIGINALEXCEPTION(  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	TRUE if want to propagate original exception, FALSE to wrap in OracleXMLException.

## REMOVEXSLTPARAM

Removes the value of a top-level stylesheet parameter. If no stylesheet is registered, this method is not operational.

### Syntax

```
PROCEDURE REMOVEXSLTPARAM(  
  ctxHdl IN ctxType,  
  name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.

## SETBINDVALUE

Sets a value for a particular bind name.

### Syntax

```
PROCEDURE SETBINDVALUE(  
  ctxHdl IN ctxType,  
  bindName IN VARCHAR2,  
  bindValue IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
bindName	(IN)	Bind name.
bindValue	(IN)	Bind value.

## SETCOLLIDATTRNAME

Sets the name of the id attribute of the collection element's separator tag. Passing `NULL` or an empty string for the tag causes the row id attribute to be omitted.

### Syntax

```
PROCEDURE SETCOLLIDATTRNAME(  
  ctxHdl IN ctxType,  
  attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
attrName	(IN)	Attribute name.

## SETDATAHEADER

Sets the XML data header. The data header is an XML entity that is appended at the beginning of the query-generated XML entity, the rowset. The two entities are enclosed by the docTag argument. The last data header specified is used. Passing in NULL for the header parameter unsets the data header.

### Syntax

```
PROCEDURE SETDATAHEADER(  
  ctxHdl IN ctxType,  
  header IN CLOB := null,  
  tag IN VARCHAR2 := null);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
header	(IN)	Header.
tag	(IN)	Tag used to enclose the data header and the rowset.

## SETDATEFORMAT

Sets the format of the generated dates in the XML document. The syntax of the date format pattern, the date mask, should conform to the requirements of the `java.text.SimpleDateFormat` class. Setting the mask to `NULL` or an empty string sets the default mask -- `DEFAULT_DATE_FORMAT`.

### Syntax

```
PROCEDURE SETDATEFORMAT(  
  ctxHdl IN ctxType,  
  mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>mask</code>	(IN)	The date mask.



## SETENCODINGTAG

Sets the encoding processing instruction in the XML document.

### Syntax

```
PROCEDURE SETENCODINGTAG(  
  ctxHdl IN ctxType,  
  enc IN VARCHAR2 := DB_ENCODING);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
enc	(IN)	The encoding to use.

## SETERRORTAG

Sets the tag to be used to enclose the XML error documents.

### Syntax

```
PROCEDURE SETERRORTAG(  
  ctxHdl IN ctxType,  
  tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

## SETMAXROWS

Sets the maximum number of rows to be converted to XML. By default, there is no set maximum.

### Syntax

```
PROCEDURE SETMAXROWS (  
  ctxHdl IN ctxType,  
  rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to generate.

## SETMETAHEADER

Sets the XML meta header. When set, the header is inserted at the beginning of the metadata part (DTD or XMLSchema) of each XML document generated by this object. The last meta header specified is used. Passing in `NULL` for the `header` parameter unsets the meta header.

### Syntax

```
PROCEDURE SETMETAHEADER(  
  ctxHdl IN ctxType,  
  header IN CLOB := null);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>Header</code>	(IN)	Header.

## SETRAISEEXCEPTION

Specifies whether to throw raised exceptions. If this call isn't made or if `FALSE` is passed to the `flag` argument, the XSU catches the SQL exceptions and generates an XML document from the exception message.

### Syntax

```
PROCEDURE SETRAISEEXCEPTION(  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN:=true);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>flag</code>	(IN)	Throw raised exceptions? <code>TRUE</code> for yes, otherwise <code>FALSE</code> .

## SETRAISENOROWSEXCEPTION

Specifies whether to throw an `OracleXMLNoRowsException` when the generated XML document is empty. By default, the exception is not thrown.

### Syntax

```
PROCEDURE SETRAISENOROWSEXCEPTION(  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN:=false);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Throws an <code>OracleXMLNoRowsException</code> if set to <code>TRUE</code> .

## SETROWIDATTRNAME

Sets the name of the id attribute of the row enclosing tag. Passing `NULL` or an empty string for the tag causes the row id attribute to be omitted.

### Syntax

```
PROCEDURE SETROWIDATTRNAME(  
  ctxHdl IN ctxType,  
  attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
attrName	(IN)	Attribute name.

## SETROWIDATTRVALUE

Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag. Passing `NULL` or an empty string for the `colName` assigns the row count value (0, 1, 2 and so on) to the row id attribute.

### Syntax

```
PROCEDURE SETROWIDATTRVALUE(  
  ctxHdl IN ctxType,  
  colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>colName</code>	(IN)	Column whose value is to be assigned to the row id attribute.



## SETROWSETTAG

Sets the tag to be used to enclose the XML dataset.

### Syntax

```
PROCEDURE SETROWSETTAG(  
  ctxHdl IN ctxType,  
  tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

## SETROWTAG

Sets the tag to be used to enclose the XML element corresponding to a `db.record`.

### Syntax

```
PROCEDURE SETROWTAG (  
  ctxHdl IN ctxType,  
  tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>tag</code>	(IN)	Tag name.

## SETSKIPROWS

Sets the number of rows to skip. By default, 0 rows are skipped.

### Syntax

```
PROCEDURE SETSKIPROWS(  
  ctxHdl IN ctxType,  
  rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to skip.

## SETSQLTOXMLNAMEESCAPING

This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

### Syntax

```
PROCEDURE SETSQLTOXMLNAMEESCAPING (  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping? TRUE for yes, otherwise FALSE.

## SETSTYLESHEETHEADER

Sets the stylesheet header (the stylesheet processing instructions) in the generated XML document. Passing `NULL` for the `uri` argument will unset the stylesheet header and the stylesheet type.

### Syntax

```
PROCEDURE SETSTYLESHEETHEADER(  
  ctxHdl IN ctxType,  
  uri IN VARCHAR2,  
  type IN VARCHAR2 := 'text/xsl');
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>uri</code>	(IN)	Stylesheet URI.
<code>type</code>	(IN)	Stylesheet type; defaults to "text/xsl".

## SETTAGCASE

Specifies the case of the generated XML tags.

### Syntax

```
PROCEDURE SETTAGCASE (  
  ctxHdl IN ctxType,  
  tCase IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tCase	(IN)	The tag's case: <ul style="list-style-type: none"><li>■ 0 for as are</li><li>■ 1 for lower case</li><li>■ 2 for upper case</li></ul>

## SETXSLT

Registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it is replaced by the new one. The options are described in the following table. Passing `NULL` for the `uri` argument or an empty string for the `stylesheet` argument will unset the stylesheet header and type.

Syntax	Description
<pre>PROCEDURE SETXSLT(   ctxHdl IN ctxType,   uri IN VARCHAR2,   ref IN VARCHAR2 := null);</pre>	To un-register the stylesheet pass in a null for the uri.
<pre>PROCEDURE SETXSLT(   ctxHdl IN ctxType,   stylesheet CLOB,   ref IN VARCHAR2 := null);</pre>	To un-register the stylesheet pass in a null or an empty string for the stylesheet.

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>uri</code>	(IN)	Stylesheet URI.
<code>stylesheet</code>	(IN)	Stylesheet.
<code>ref</code>	(IN)	URL to include, imported and external entities.

## SETXSLTPARAM

Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression; the string literal values would therefore have to be quoted explicitly. If no stylesheet is registered, this method is not operational.

### Syntax

```
PROCEDURE SETXSLTPARAM(  
  ctxHdl IN ctxType,  
  name IN VARCHAR2,  
  value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.
value	(IN)	Value to be assigned to the stylesheet parameter.



## USENULLATTRIBUTEINDICATOR

Specifies whether to use an XML attribute to indicate NULLness, or to do this by omitting the particular entity in the XML document.

### Syntax

```
PROCEDURE SETNULLATTRIBUTEINDICATOR(  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Sets attribute to NULL if TRUE, omits from XML document if FALSE.

## USETYPEFORCOLLELEMTAG

Specifies whether to use the collection element's type name as its element tag name. By default, the tag name for elements of a collection is the collection's tag name followed by `_item`.

### Syntax

```
PROCEDURE USETYPEFORCOLLELEMTAG(  
  ctxHdl IN ctxType,  
  flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>flag</code>	(IN)	Turn on use of the type name?

DBMS\_XMLSAVE provides XML to database-type functionality.

**See Also:** *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_XMLSAVE](#)
  - Security Model
  - Constants
  - Types
- [Summary of DBMS\\_XMLSAVE Subprograms](#)

## Using DBMS\_XMLSAVE

- [Security Model](#)
- [Constants](#)
- [Types](#)

## Security Model

Owned by XDB, the DBMS\_XMLSAVE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Constants

**Table 175–1 Constants of DBMS\_XMLSAVE**

<b>Constant</b>	<b>Description</b>
DEFAULT_ROWTAG	The default tag name for the element corresponding to database records -- ROW
DEFAULT_DATE_FORMAT	Default date mask: 'MM/dd/yyyy HH:mm:ss'
MATCH_CASE	Used to specify that when mapping XML elements to database entities; the XSU should be case sensitive.
IGNORE_CASE	Used to specify that when mapping XML elements to database entities the XSU should be case insensitive.

## Types

**Table 175–2** *Types of DBMS\_XMLSAVE*

Type	Description
ctxType	The type of the query context handle. The type of the query context handle. This the return type of <a href="#">NEWCONTEXT</a> .

---

## Summary of DBMS\_XMLSAVE Subprograms

**Table 175-3 DBMS\_XMLSAVE Package Subprograms**

Method	Description
<a href="#">CLEARKEYCOLUMNLIST</a> on page 175-7	Clears the key column list.
<a href="#">CLEARUPDATECOLUMNLIST</a> on page 175-8	Clears the update column list.
<a href="#">CLOSECONTEXT</a> on page 175-9	It closes/deallocates a particular save context.
<a href="#">DELETEXML</a> on page 175-10	Deletes records specified by data from the XML document, from the table specified at the context creation time.
<a href="#">GETEXCEPTIONCONTENT</a> on page 175-11	Returns the thrown exception's error code and error message.
<a href="#">INSERTXML</a> on page 175-12	Inserts the XML document into the table specified at the context creation time.
<a href="#">NEWCONTEXT</a> on page 175-13	Creates a save context, and returns the context handle.
<a href="#">PROPAGATEORIGINALEXCEPTION</a> on page 175-14	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather than, wrapping it with an <code>OracleXMLSQLException</code> .
<a href="#">REMOVEXSLTPARAM</a> on page 175-15	Removes the value of a top-level stylesheet parameter
<a href="#">SETBATCHSIZE</a> on page 175-16	Changes the batch size used during DML operations.
<a href="#">SETCOMMITBATCH</a> on page 175-18	Sets the commit batch size.
<a href="#">SETDATEFORMAT</a> on page 175-18	Sets the format of the generated dates in the XML document.
<a href="#">SETIGNORECASE</a> on page 175-19	The XSU does mapping of XML elements to database.
<a href="#">SETKEYCOLUMN</a> on page 175-20	This methods adds a column to the key column list.
<a href="#">SETPRESERVEWHITESPACE</a> on page 175-21	Tells the XSU whether to preserve whitespace or not.
<a href="#">SETROWTAG</a> on page 175-22	Names the tag used in the XML document to enclose the XML elements corresponding to database.
<a href="#">SETSQLTOXMLNAMEESCAPING</a> on page 175-23	This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.
<a href="#">SETUPDATECOLUMN</a> on page 175-24	Adds a column to the update column list.
<a href="#">SETXSLT</a> on page 175-25	Registers a XSL transform to be applied to the XML to be saved.
<a href="#">SETXSLTPARAM</a> on page 175-26	Sets the value of a top-level stylesheet parameter.
<a href="#">UPDATEXML</a> on page 175-27	Updates the table given the XML document.



## CLEARKEYCOLUMNLIST

Clears the key column list.

### Syntax

```
PROCEDURE clearKeyColumnList(  
    ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## CLEARUPDATECOLUMNLIST

Clears the update column list.

### Syntax

```
PROCEDURE clearUpdateColumnList(  
    ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## CLOSECONTEXT

Closes/deallocates a particular save context.

### Syntax

```
PROCEDURE closeContext(  
    ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## DELETXML

Deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted. The options are described in the following table.

Syntax	Description
<pre>FUNCTION deleteXML(     ctxHdl IN ctxPType,     xDoc IN VARCHAR2) RETURN NUMBER;</pre>	Uses a VARCHAR2 type for the xDoc parameter.
<pre>FUNCTION deleteXML(     ctxHdl IN ctxType,     xDoc IN CLOB) RETURN NUMBER;</pre>	Uses a CLOB type for the xDoc parameter.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

## GETEXCEPTIONCONTENT

Through its arguments, this method returns the thrown exception's error code and error message, SQL error code. This is to get around the fact that the JVM throws an exception on top of whatever exception was raised; thus, rendering PL/SQL unable to access the original exception.

### Syntax

```
PROCEDURE getExceptionContent(  
    ctxHdl IN ctxType,  
    errNo OUT NUMBER,  
    errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(IN)	Error number.
errMsg	(IN)	Error message.

## INSERTXML

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted. The options are described in the following table.

<b>Syntax</b>	<b>Description</b>
<pre>FUNCTION insertXML(     ctxHdl IN ctxType,     xDoc IN VARCHAR2) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a VARCHAR2.
<pre>FUNCTION insertXML(     ctxHdl IN ctxType,     xDoc IN CLOB) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a CLOB.

<b>Parameter</b>	<b>IN / OUT</b>	<b>Description</b>
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

## NEWCONTEXT

Creates a save context, and returns the context handle.

### Syntax

```
FUNCTION newContext(  
    targetTable IN VARCHAR2)  
RETURN ctxType;
```

Parameter	IN / OUT	Description
targetTable	(IN)	The target table into which to load the XML document.

## PROPAGATEORIGINALEXCEPTION

Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather than, wrapping it with an `OracleXMLSQLException`.

### Syntax

```
PROCEDURE propagateOriginalException(  
    ctxHdl IN ctxType,  
    flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Propagate the original exception? 0=FALSE, 1=TRUE.



## REMOVEXSLTPARAM

Removes the value of a top-level stylesheet parameter.

### Syntax

```
PROCEDURE removeXSLTParam(  
    ctxHdl IN ctxType,  
    name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.

## SETBATCHSIZE

Changes the batch size used during DML operations. When performing inserts, updates or deletes, it is better to batch the operations so that they get executed in one shot rather than as separate statements. The flip side is that more memory is needed to buffer all the bind values. Note that when batching is used, a commit occurs only after a batch is executed. So if one of the statement inside a batch fails, the whole batch is rolled back. This is a small price to pay considering the performance gain; nevertheless, if this behavior is unacceptable, then set the batch size to 1.

### Syntax

```
PROCEDURE setBatchSize(  
    ctxHdl IN ctxType,  
    batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
batchSize	(IN)	Batch size.

## SETCOMMITBATCH

Sets the commit batch size. The commit batch size refers to the number of records inserted after which a commit should follow. If `batchSize` is less than 1 or the session is in "auto-commit" mode, using the XSU does not make any explicit commits. By default, `commitBatch` is 0.

### Syntax

```
PROCEDURE setCommitBatch(  
    ctxHdl IN ctxType,  
    batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>batchSize</code>	(IN)	Commit batch size.

## SETDATEFORMAT

Sets the format of the generated dates in the XML document. The syntax of the date format pattern, the date mask, should conform to the requirements of the class `java.text.SimpleDateFormat`. Setting the mask to `<code>null</code>` or an empty string unsets the date mask.

### Syntax

```
PROCEDURE setDateFormat(  
    ctxHdl IN ctxType,  
    mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
mask	(IN)	Syntax of the date format pattern..

## SETIGNORECASE

The XSU does mapping of XML elements to db columns/attributes based on the element names (XML tags). This function tells the XSU to do this match case insensitive.

### Syntax

```
PROCEDURE setIgnoreCase(  
    ctxHdl IN ctxType,  
    flag IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Ignore tag case in the XML doc? 0=FALSE, 1=TRUE.

## SETKEYCOLUMN

This method adds a column to the "key column list". The value for the column cannot be `NULL`. In case of update or delete, the columns in the key column list make up the `WHERE` clause of the statement. The key columns list must be specified before updates can complete; this is optional for delete operations.

### Syntax

```
PROCEDURE setKeyColumn(  
    ctxHdl IN ctxType,  
    colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the key column list; cannot be <code>NULL</code> .

## SETPRESERVEWHITESPACE

Tells the XSU whether or not to preserve whitespace.

### Syntax

```
PROCEDURE setPreserveWhitespace(  
    ctxHdl IN ctxType,  
    flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Should XSU preserve whitespace?

## SETROWTAG

Names the tag used in the XML document to enclose the XML elements corresponding to db. records.

### Syntax

```
PROCEDURE setRowTag(  
    ctxHdl IN ctxType,  
    tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.



## SETSQLTOXMLNAMEESCAPING

Turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

### Syntax

```
PROCEDURE setSQLToXMLNameEscaping(  
    ctxHdl IN ctxType,  
    flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping?

## SETUPDATECOLUMN

Adds a column to the update column list. In case of insert, the default is to insert values to all the columns in the table; on the other hand, in case of updates, the default is to only update the columns corresponding to the tags present in the ROW element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

### Syntax

```
PROCEDURE setUpdateColumn(  
    ctxHdl IN ctxType,  
    colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the update column list.

## SETXSLT

Registers an XSL transform to be applied to the XML to be saved. If a stylesheet was already registered, it gets replaced by the new one. To un-register the stylesheet, pass in null for the URI. The options are described in the following table.

Syntax	Description
<pre>PROCEDURE setXSLT(     ctxHdl IN ctxType,     uri IN VARCHAR2,     ref IN VARCHAR2 := null);</pre>	Passes in the stylesheet through a URI.
<pre>PROCEDURE setXSLT(     ctxHdl IN ctxType,     stylesheet IN CLOB,     ref IN VARCHAR2 := null);</pre>	Passes in the stylesheet through a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
uri	(IN)	URI to the stylesheet to register.
ref	(IN)	URL for include, import, and external entities.
stylesheet	(IN)	CLOB containing the stylesheet to register

## SETXSLTPARAM

Sets the value of a top-level stylesheet parameter. The parameter is expected to be a valid XPath expression; literal values would therefore have to be explicitly quoted.

### Syntax

```
PROCEDURE setXSLTParam(  
    ctxHdl IN ctxType,  
    name IN VARCHAR2,  
    value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.
value	(IN)	Parameter value as an XPath expression

## UPDATEXML

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated. The options are described in the following table.

Syntax	Description
<pre>FUNCTION updateXML(     ctxHdl IN ctxType,     xDoc IN VARCHAR2) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a VARCHAR2.
<pre>FUNCTION updateXML(     ctxHdl IN ctxType,     xDoc IN CLOB) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.



---

---

## DBMS\_XMLSCHEMA

DBMS\_XMLSCHEMA package provides procedures to manage XML schemas. It is created by script `dbmsxsch.sql` during Oracle database installation.

**See Also:** *Oracle XML DB Developer's Guide*

This chapter contains the following topics:

- [Using DBMS\\_XMLSCHEMA](#)
  - Overview
  - Security Model
  - Constants
  - Views
  - Operational Notes
- [Summary of DBMS\\_XMLSCHEMA Subprograms](#)

---

## Using DBMS\_XMLSCHEMA

This section contains topics which relate to using the DBMS\_XMLSCHEMA package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Views](#)
- [Operational Notes](#)



## Overview

This package provides subprograms to

- Register an XML schema
- Delete a previously registered XML schema
- Re-compile a previously registered XML schema
- Generate an XML schema
- Evolves an XML schema

## Security Model

Owned by XDB, the DBMS\_XMLSCHEMA package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Constants

The DBMS\_XMLSCHEMA package uses the constants shown in following tables.

- [DBMS\\_XMLSCHEMA Constants - Delete Option](#)
- [DBMS\\_XMLSCHEMA Constants - Enable Hierarchy](#)
- [DBMS\\_XMLSCHEMA Constants - Register CSID](#)

**Table 176–1 DBMS\_XMLSCHEMA Constants - Delete Option**

Constant	Type	Value	Description
DELETE_RESTRICT	NUMBER	1	Deletion of an XML schema fails if there are any tables or XML schemas that depend on it
DELETE_INVALIDATE	NUMBER	2	Deletion of an XML schema does not fail if there are tables or XML schemas that depend on it. All dependent tables and schemas are invalidated.
DELETE_CASCADE	NUMBER	3	Deletion of an XML schema also drops all SQL types and default tables associated with it. SQL types are dropped only if <code>gentypes</code> argument was set to <code>TRUE</code> during registration of the XML schema. However, deletion of the XML schema fails if there are any instance documents conforming to the schema or any dependent XML schemas.
DELETE_CASCADE_FORCE	NUMBER	4	This option is similar to <code>DELETE_CASCADE</code> except that it does not check for any stored instance documents conforming to the schema or any dependent XML schemas. Also, it ignores any errors.

**Table 176–2 DBMS\_XMLSCHEMA Constants - Enable Hierarchy**

Constant	Type	Value	Description
ENABLE_HIERARCHY_NONE	PLS_INTEGER	1	The <code>ENABLE_HIERARCHY</code> procedure of the <code>DBMS_XDBZ</code> package will not be called on any tables created while registering that schema
ENABLE_HIERARCHY_CONTENTS	PLS_INTEGER	2	The <code>ENABLE_HIERARCHY</code> procedure of the <code>DBMS_XDBZ</code> package will be called for all tables created during schema registration with <code>hierarchy_type</code> as <code>DBMS_XDBZ.ENABLE_CONTENTS</code>

**Table 176–2 (Cont.) DBMS\_XMLSCHEMA Constants - Enable Hierarchy**

Constant	Type	Value	Description
ENABLE_HIERARCHY_ RESMETADATA	PLS_INTEGER	3	The ENABLE_HIERARCHY procedure of the DBMS_XDBZ package will be called on all tables created during schema registration with hierarchy_type as DBMS_XDBZ.ENABLE_RESMETADATA. Users should pass in DBMS_XMLSCHEMA.ENABLE_RESMETADATA for schemas they intend to use as resource metadata tables.

**Table 176–3 DBMS\_XMLSCHEMA Constants - Register CSID**

Constant	Type	Value	Description
REGISTER_NODOCID	NUMBER	1	If a schema is registered for metadata use (using the value ENABLE_HIER_RESMETADATA for parameter enablehierarchy during registration), a column named DOCID is added to all tables created during schema registration. This constant can be used in the options argument of REGISTERSCHEMA to prevent the creation of this column if the user wishes to optimize on storage
REGISTER_CSID_NULL	NUMBER	-1	If user wishes to not specify the character set of the input schema document when invoking REGISTERSCHEMA, this value can be used for the csid parameter

## Views

The DBMS\_XMLSCHEMA package uses the views shown in [Table 176–4](#). The columns of these views are described in detail in the *Oracle Database Reference*.

**Table 176–4 Summary of Views used by DBMS\_XMLSCHEMA**

Schema	Description
USER_XML_SCHEMAS	All registered XML Schemas owned by the user
ALL_XML_SCHEMAS	All registered XML Schemas usable by the current user
DBA_XML_SCHEMAS	All registered XML Schemas in the database
DBA_XML_TABLES	All XMLType tables in the system
USER_XML_TABLES	All XMLType tables owned by the current user
ALL_XML_TABLES	All XMLType tables usable by the current user
DBA_XML_TAB_COLS	All XMLType table columns in the system
USER_XML_TAB_COLS	All XMLType table columns in tables owned by the current user
ALL_XML_TAB_COLS	All XMLType table columns in tables usable by the current user
DBA_XML_VIEWS	All XMLType views in the system
USER_XML_VIEWS	All XMLType views owned by the current user
ALL_XML_VIEWS	All XMLType views usable by the current user
DBA_XML_VIEW_COLS	All XMLType view columns in the system
USER_XML_VIEW_COLS	All XMLType view columns in views owned by the current user
ALL_XML_VIEW_COLS	All XMLType view columns in views usable by the current user

## Operational Notes

### **Guidelines for Using In-Place XML Schema Evolution**

Before you perform an in-place XML-schema evolution, you should follow these preparatory steps:

1. Back up all existing data (instance documents) for the XML schema that will be evolved.
2. Perform a dry run using trace only, that is, without actually evolving the XML schema or updating any instance documents, to produce a trace of the update operations that would be performed during evolution. To do this, set the flag parameter value to only `INPLACE_TRACE`. Do not also use `INPLACE_EVOLVE`. After performing the dry run, examine the trace file, verifying that the listed DDL operations are in fact those that you intend.

---

## Summary of DBMS\_XMLSCHEMA Subprograms

**Table 176–5 DBMS\_XMLSCHEMA Package Subprograms**

Method	Description
<a href="#">COMPILESHEMA Procedure</a> on page 176-10	Used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state.
<a href="#">COPYEVOLVE Procedure</a> on page 176-11	Evolves registered schemas so that existing XML instances remain valid
<a href="#">DELETESHEMA Procedure</a> on page 176-13	Removes the schema from the database
<a href="#">GENERATESHEMA Function</a> on page 176-14	Generates an XML schema from an oracle type name
<a href="#">GENERATESHEMAS Function</a> on page 176-15	Generates several XML schemas from an oracle type name
<a href="#">INPLACEEVOLVE Procedure</a> on page 176-16	Evolves registered schemas by propagating schema changes to object types and tables
<a href="#">PURGESHEMA Procedure</a> on page 176-18	Removes the XML schema
<a href="#">REGISTERSHEMA Procedures</a> on page 176-19	Registers the specified schema for use by Oracle. This schema can then be used to store documents conforming to this.
<a href="#">REGISTERURI Procedure</a> on page 176-23	Registers an XML schema specified by a URI name

## COMPILESCHEMA Procedure

This procedure can be used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state. Can result in a `ORA-31001` exception: invalid resource handle or path name.

### Syntax

```
DBMS_XMLSCHEMA.COMPILESCHEMA(  
    schemaur1 IN VARCHAR2);
```

### Parameters

**Table 176–6** *COMPILESCHEMA Procedure Parameters*

Parameter	Description
<code>schemaur1</code>	URL identifying the schema



## COPYEVOLVE Procedure

This procedure evolves registered schemas so that existing XML instances remain valid.

This procedure is accomplished in according to the following basic scenario (alternative actions are controlled by the procedure's parameters):

- copies data in schema based XMLType tables to temporary table storage
- drops old tables
- deletes old schemas
- registers new schemas
- creates new XMLType tables
- Populates new tables with data in temporary storage; auxiliary structures (constraints, triggers, indexes, and others) are not preserved
- drops temporary tables

### See Also:

- "Schema Evolution" chapter of the *Oracle XML DB Developer's Guide* for examples on how to evolve existing schemas
- *Oracle Database Error Messages* for information on exceptions specific to schema evolution, ORA-30142 through ORA-30946.

## Syntax

```
DBMS_XMLSCHEMA.COPYEVOLVE(
  schemaurls      IN  XDB$STRUBG_LIST_T,
  newschemas      IN  XMLSequenceType,
  transforms      IN  XMLSequenceType :=NULL,
  preserveolddocs IN  BOOLEAN :=FALSE,
  maptablename    IN  VARCHAR2 :=NULL,
  generatetables IN  BOOLEAN :=TRUE,
  force           IN  BOOLEAN :=FALSE,
  schemaowners    IN  XDB$STRING_LIST_T :=NULL,
  parallelDegree  IN  PLS_INTEGER := 0,
  options         IN  PLS_INTEGER := 0);
```

## Parameters

**Table 176–7 COPYEVOLVE Procedure Parameters**

Parameter	Description
schemaurls	VARRAY of URLs of all schemas to be evolved. Should include the dependent schemas. Unless the FORCE parameter is TRUE, URLs should be in the order of dependency.
newschemas	VARRAY of new schema documents. Should be specified in same order as the corresponding URLs.
transforms	VARRAY of transforming XSL documents to be applied to schema-based documents. Should be specified in same order as the corresponding URLs. Optional if no transformations are required.

**Table 176–7 (Cont.) COPYEVOLVE Procedure Parameters**

Parameter	Description
preserveolddocs	Default is <code>FALSE</code> , and temporary tables with old data are dropped. If <code>TRUE</code> , these table are still available after schema evolution is complete.
maptabname	Specifies the name of the table mapping permanent to temporary tables during the evolution process. Valid columns are: <ul style="list-style-type: none"> <li>▪ <code>SCHEMA_URL</code> - <code>VARCHAR2 (700)</code> - URL of schema to which this table conforms</li> <li>▪ <code>SCHEMA_OWNER</code> - <code>VARCHAR2 (30)</code> - Owner of the schema</li> <li>▪ <code>ELEMENT_NAME</code> - <code>VARCHAR2 (256)</code> - Element to which this table conforms</li> <li>▪ <code>TAB_NAME</code> - <code>VARCHAR2 (65)</code> - Qualified table name: <code>&lt;owner_name&gt;.&lt;table_name&gt;</code></li> <li>▪ <code>COL_NAME</code> - <code>VARCHAR2 (4000)</code> - Name of the column (NULL for XMLType tables)</li> <li>▪ <code>TEMP_TABNAME</code> - <code>VARCHAR2 (30)</code> - Name of temporary tables which holds data for this table.</li> </ul>
generatetables	Default is <code>TRUE</code> , and new tables will be generated. If <code>FALSE</code> : <ul style="list-style-type: none"> <li>▪ new tables will not be generated after registration of new schemas</li> <li>▪ <code>preserveolddocs</code> must be <code>TRUE</code></li> <li>▪ <code>maptabname</code> must be non-NULL</li> </ul>
force	Default is <code>FALSE</code> . If <code>TRUE</code> , ignores errors generated during schema evolution. Used when there are circular dependencies among schemas to ensure that all schemas are stored despite possible errors in registration.
schemaowners	VARRAY of names of schema owners. Should be specified in same order as the corresponding URLs. Default is <code>NULL</code> , assuming that all schemas are owned by the current user.
paralleldegree	Specifies the degree of parallelism to be used in a <code>PARALLEL</code> hint during the data copy stage of the evolution. If this is 0 (default), the <code>PARALLEL</code> hint will not be given in the data copy statements.
options	Currently, the only supported option is <code>COPYEVOLVE_BINARY_XML</code> which lets you register the new schemas for binary XML and create the new tables/columns with binary XML as the storage type.

## Usage Notes

You should back up all schemas and documents prior to invocation because [COPYEVOLVE Procedure](#) deletes all conforming documents prior to implementing the schema evolution.

## DELETESCHEMA Procedure

This procedure deletes the XML Schema specified by the URL.

### Syntax

```
DBMS_XMLSCHEMA.DELETESCHEMA (
  schemaur1      IN  VARCHAR2,
  delete_option  IN  PLS_INTEGER := DELETE_RESTRICT);
```

**See Also:** "XMLSCHEMA Storage and Query: Basic" chapter of the *Oracle XML DB Developer's Guide*

### Parameters

**Table 176–8 DELETESCHEMA Procedure Parameters**

Parameter	Description
schemaur1	URL identifying the schema to be deleted
delete_option	Delete options: <ul style="list-style-type: none"> <li>▪ DELETE_RESTRICT - Schema deletion fails if there are any tables or schemas that depend on this schema</li> <li>▪ DELETE_INVALIDATE - Schema deletion does not fail if there are any dependencies. Instead, it simply invalidates all dependent objects.</li> <li>▪ DELETE_CASCADE - Schema deletion will also drop all default SQL types and default tables. However the deletion fails if there are any stored instances conforming to this schema.</li> <li>▪ DELETE_CASCADE_FORCE - Similar to DELETE_CASCADE except that it does not check for any stored instances conforming to this schema. Also, it ignores any errors.</li> </ul>

### Exceptions

**Table 176–9 DELETESCHEMA Procedure Exceptions**

Exception	Description
ORA-31001	Invalid resource handle or path name

## GENERATESCHEMA Function

This function generates XML schema(s) from an Oracle type name. It inlines all in one schema (XMLType).

**See Also:** "XMLSCHEMA Storage and Query: Advanced" chapter of the *Oracle XML DB Developer's Guide*

### Syntax

```
DBMS_XMLSCHEMA.GENERATESCHEMA (
  schemaname      IN  VARCHAR2,
  typename        IN  VARCHAR2,
  elementname     IN  VARCHAR2 := NULL,
  recurse         IN  BOOLEAN  := TRUE,
  annotate        IN  BOOLEAN  := TRUE,
  embedcoll       IN  BOOLEAN  := TRUE)
RETURN SYS.XMLTYPE;
```

### Parameters

**Table 176–10 GENERATESCHEMA Function Parameters**

Parameter	Description
schemaname	Name of the database schema containing the type
typename	Name of the Oracle type
elementname	The name of the top level element in the XML Schema. Defaults to typename.
recurse	Whether or not to also generate schema for all types referred to by the type specified
annotate	Whether or not to put the SQL annotations in the XML Schema
embedcoll	Determines whether the collections should be embedded in the type which refers to them, or create a complextype. Cannot be FALSE if annotations are turned on

### Exceptions

**Table 176–11 GENERATESCHEMA Function Exceptions**

Exception	Description
ORA-31001	Invalid resource handle or path name

## GENERATESCHEMAS Function

This function generates XML schema(s) from an Oracle type name. It returns a collection of `XMLTypes`, one XML Schema document for each database schema.

**See Also:** "XMLSCHEMA Storage and Query: Advanced" chapter of the *Oracle XML DB Developer's Guide*

### Syntax

```
DBMS_XMLSCHEMA.GENERATESCHEMAS (
  schemaname   IN  VARCHAR2,
  typename     IN  VARCHAR2,
  elementname  IN  VARCHAR2 := NULL,
  schemaurl    IN  VARCHAR2 := NULL,
  annotate     IN  BOOLEAN := TRUE,
  embedcoll   IN  BOOLEAN := TRUE )
RETURN SYS.XMLTYPE;
```

### Parameters

**Table 176–12** *GENERATESCHEMAS Function Parameters*

Parameter	Description
schemaname	Name of the database schema containing the type
typename	Name of the Oracle type
elementname	The name of the top level element in the XML Schema defaults to <code>typeName</code>
schemaurl	Specifies base URL where schemas will be stored, needed by top level schema for import statement
annotate	Whether or not to put the SQL annotations in the XML Schema
embedcoll	Determines whether the collections be embedded in the type which refers to them, or create a <code>complextype</code> . Cannot be <code>FALSE</code> if annotations are turned on

### Exceptions

**Table 176–13** *GENERATESCHEMAS Function Exceptions*

Exception	Description
ORA-31001	Invalid resource handle or path name

## INPLACEEVOLVE Procedure

This procedure evolves registered schemas by propagating schema changes to object types and tables.

### Syntax

```
DBMS_XMLSCHEMA.INPLACEEVOLVE (
  schemaURL    IN   VARCHAR2,
  diffXML      IN   XMLType,
  flags        IN   NUMBER);
```

### Parameters

**Table 176–14** *INPLACEEVOLVE Procedure Parameters*

Parameter	Description
schemaurl	URL of the schema to evolve
diffXML	Changes to be applied to the schema. This is an XML document conforming to the XDIFF schema and specifies what changes need to be applied and the locations in the schema document where the changes are to be applied.
flags	<p>The following bits may be set in this parameter to control the behavior of this procedure:</p> <ul style="list-style-type: none"> <li>▪ INPLACE_EVOLVE (value 1, meaning that bit 1 is on) – Perform in-place XML schema evolution: construct a new XML schema and validate it (against the XML schema for XML schemas); construct the DDL statements needed to evolve the instance-document disk structures, execute the DDL statements, and replace the old XML schema with the new.</li> <li>▪ INPLACE_TRACE (value 2, meaning that bit 2 is on) – Perform all steps necessary for in-place evolution, except executing the DDL statements and overwriting the old XML schema with the new, then write both the DDL statements and the new XML schema to a trace file.</li> </ul> <p>That is, each of the bits constructs the new XML schema, validates it, and determines the steps needed to evolve the disk structures underlying the instance documents. In addition:</p> <ul style="list-style-type: none"> <li>▪ Bit INPLACE_EVOLVE carries out those evolution steps and replaces the old XML schema with the new.</li> <li>▪ Bit INPLACE_TRACE saves the evolution steps and the new XML schema in a trace file (it does not carry out the evolution steps)</li> </ul>

### Exceptions

The procedure raises exceptions in the following cases:

- An error will be raised for invalid XPATH expressions and for XDIFF documents that do not conform to the xdiff schema.
- Path expressions that are syntactically correct but result in an invalid node in the schema document will result in an error.
- If the schema change makes the schema an ill-formed XML document or an invalid XML schema, this will raise an error.

- Any errors resulting from `CREATE TYPE`, `ALTER TYPE` and like commands will generate error messages.

### Usage Notes

- Users are required to backup all their data before attempting in-place evolution, as there is no rollback with this operation.
- A user must register their new XML schema with the database using the [REGISTERSHEMA Procedures](#) and the [REGISTERURI Procedure](#) at a schema URL that is different from that of the one to be evolved. If the new schema registers successfully and is usable, only then should the user attempt to evolve the existing schema to the new schema by means of this subprogram. If the registration of the new schema is successful, then the user must delete this schema (and all its dependent objects) before attempting to evolve the schema at the old schema URL.

## PURGESCHEMA Procedure

This procedure removes the XML schema.

**See Also:** "XMLSCHEMA Storage and Query: Advanced" chapter of the *Oracle XML DB Developer's Guide*

### Syntax

```
DBMS_XMLSCHEMA.PURGESCHEMA(  
    schemaid IN RAW);
```

### Parameters

**Table 176–15** *PURGESCHEMA Procedure Parameters*

Parameter	Description
schemaid	ID of the schema to be purged

### Usage Notes

- The schema should have been originally registered for binary encoding and should have been deleted in the `HIDE` mode.
- Once a schema has been deleted in `HIDE` mode, it continues to exist in the XML DB dictionary and is used for decoding already encoded documents. The user invokes this interface when there are no stored instances encoded with this schema.
- Once the schema is purged, any space used by that schema will be reclaimed and documents encoded using the schema will raise an error if an attempt is made to decode them.
- The Schema ID can be obtained from the catalog views.



## REGISTERSCHEMA Procedures

This procedure registers the specified schema for use by the database. The procedure is overloaded. The different functionality of each form of syntax is presented along with the definition.

---



---

**Note:** As of release 11.2.0.2 the `genbean` parameter is deprecated. Oracle recommends that you do not use this parameter in new applications. Support for this feature is for backward compatibility only.

---



---

**See Also:** "XMLSCHEMA Storage and Query: Basic" chapter of the *Oracle XML DB Developer's Guide*

### Syntax

Registers a schema specified as a `VARCHAR2`:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  VARCHAR2,
  local          IN  BOOLEAN := TRUE,
  gentypes       IN  BOOLEAN := TRUE,
  genbean        IN  BOOLEAN := FALSE,
  gentables      IN  BOOLEAN := TRUE,
  force          IN  BOOLEAN := FALSE,
  owner          IN  VARCHAR2 := NULL,
  enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options        IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `BFILE`. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  BFILE,
  local          IN  BOOLEAN := TRUE,
  gentypes       IN  BOOLEAN := TRUE,
  genbean        IN  BOOLEAN := FALSE,
  force          IN  BOOLEAN := FALSE,
  owner          IN  VARCHAR2 := NULL,
  enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options        IN  PLS_INTEGER := 0);
```

Registers the schema specified as a `BFILE` and identifies the character set id of the schema document:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  BFILE,
  local          IN  BOOLEAN := TRUE,
  gentypes       IN  BOOLEAN := TRUE,
  genbean        IN  BOOLEAN := TRUE,
  gentables      IN  BOOLEAN := TRUE,
  force          IN  BOOLEAN := TRUE,
  owner          IN  VARCHAR2 := '',
  csid           IN  NUMBER,
  enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
```

```
options          IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BLOB. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  BLOB,
  local          IN  BOOLEAN := TRUE,
  genTypes       IN  BOOLEAN := TRUE,
  genBean        IN  BOOLEAN := FALSE,
  force          IN  BOOLEAN := FALSE,
  owner          IN  VARCHAR2 := NULL,
  enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options        IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BLOB and identifies the character set id of the schema document:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  BLOB,
  local          IN  BOOLEAN := TRUE,
  gentypes       IN  BOOLEAN := TRUE,
  genbean        IN  BOOLEAN := TRUE,
  gentables      IN  BOOLEAN := TRUE,
  force          IN  BOOLEAN := TRUE,
  owner          IN  VARCHAR2 := '',
  csid           IN  NUMBER,
  enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options        IN  PLS_INTEGER := 0);
```

Registers the schema specified as a CLOB

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  CLOB,
  local          IN  BOOLEAN := TRUE,
  gentypes       IN  BOOLEAN := TRUE,
  genbean        IN  BOOLEAN := FALSE,
  force          IN  BOOLEAN := FALSE,
  owner          IN  VARCHAR2 := NULL,
  options        IN  PLS_INTEGER := 0);
```

Registers the schema specified as an XMLTYPE.

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
  schemadoc      IN  SYS.XMLTYPE,
  local          IN  BOOLEAN := TRUE,
  gentypes       IN  BOOLEAN := TRUE,
  genbean        IN  BOOLEAN := FALSE,
  force          IN  BOOLEAN := FALSE,
  owner          IN  VARCHAR2 := NULL,
  enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
  options        IN  PLS_INTEGER := 0);
```

Registers the schema specified as a BLOB. The contents of the schema document must be in the database character set:

```
DBMS_XMLSCHEMA.REGISTERSCHEMA (
  schemaur1      IN  VARCHAR2,
```

```

schemadoc      IN SYS.URIType,
local          IN BOOLEAN := TRUE,
gentypes      IN BOOLEAN := TRUE,
genbean       IN BOOLEAN := FALSE,
force         IN BOOLEAN := FALSE,
owner         IN VARCHAR2 := NULL,
enablehierarchy IN PLS_INTEGER := DBMS_XMLSCHEMA.ENABLE_CONTENTS,
options       IN PLS_INTEGER := 0);

```

## Parameters

**Table 176–16 REGSITERSchema Procedure Parameters**

Parameter	Description
schemaur1	URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the database hierarchy. Can be used inside <code>schemalocation</code> attribute of XML Schema import element.
schemadoc	A valid XML schema document
local	Is this a local or global schema? <ul style="list-style-type: none"> <li>■ By default, all schemas are registered as local schemas, under <code>/sys/schemas/&lt;username&gt;/...</code></li> <li>■ If a schema is registered as global, it is added under <code>/sys/schemas/PUBLIC/...</code></li> </ul> <p>You need write privileges on the directory to be able to register a schema as global.</p>
gentypes	Determines whether the schema compiler generates object types. By default, <code>TRUE</code> . If you use binary XML, you must be set <code>gentypes</code> to <code>FALSE</code> .
genbean	Determines whether the schema compiler generates Java beans. By default, <code>FALSE</code> . Oracle recommends that this parameter always be set to <code>FALSE</code> .
gentables	Determines whether the schema compiler generates default tables. By default, <code>TRUE</code>
force	If this parameter is set to <code>TRUE</code> , the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is <code>FALSE</code> .
owner	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.
csid	Identifies the character set of the input schema document. If this value is 0, the schema document's encoding is determined by the current rule for "text/xml" MIME type.

**Table 176–16 (Cont.) REGSITERSchema Procedure Parameters**

Parameter	Description
enablehierarchy	<ul style="list-style-type: none"> <li>■ <code>ENABLE_HIERARCHY_NONE</code> - enable hierarchy will not be called on any tables created while registering that schema</li> <li>■ <code>ENABLE_HIERARCHY_CONTENTS</code> - enable hierarchy will be called for all tables created during schema registration with <code>hierarchy_type</code> as <code>DBMS_XDBZ.ENABLE_CONTENTS</code>. This is the default.</li> <li>■ <code>ENABLE_HIERARCHY_RESMETADATA</code> - enable hierarchy will be called on all tables created during schema registration with <code>hierarchy_type</code> as <code>DBMS_XDBZ.ENABLE_RESMETADATA</code>. Users should pass in <code>DBMS_XMLSCHEMA.ENABLE_RESMETADATA</code> for schemas they intend to use as resource metadata tables.</li> </ul>
options	<p>Additional options to specify how the schema should be registered. The various options are represented as bits of an integer and the options parameter should be constructed by doing a <code>BITOR</code> of the desired bits. Possible bits:</p> <ul style="list-style-type: none"> <li>■ <code>REGISTER_NODOCID</code> - this will suppress the creation of the <code>DOCID</code> column for out of line tables. This is a storage optimization which might be desirable when we do not need to join back to the document table (for example if we do not care about rewriting certain queries that could be rewritten by making use of the <code>DOCID</code> column)</li> <li>■ <code>REGISTER_BINARYXML</code> - Register the schema for Binary XML</li> <li>■ <code>REGISTER_NT_AS_IOT</code> - Store nested tables created during schema registration as index organized tables. The default is to store nested tables as heap tables</li> </ul>

## REGISTERURI Procedure

This procedure registers an XML Schema specified by a URI name.

---

**Note:** As of release 11.2.0.2 the `genbean` parameter is deprecated. Oracle recommends that you do not use this parameter in new applications. Support for this feature is for backward compatibility only.

---

### Syntax

```
DBMS_XMLSCHEMA.REGISTERURI (
  schemaur1      IN  VARCHAR2,
  schemadocuri   IN  VARCHAR2,
  local          IN  BOOLEAN := TRUE,
  gentypes      IN  BOOLEAN := TRUE,
  genbean       IN  BOOLEAN := FALSE,
  gentables     IN  BOOLEAN := TRUE,
  force         IN  BOOLEAN := FALSE,
  owner         IN  VARCHAR2 := NULL,
  options       IN  PLS_INTEGER := 0);
```

### Parameters

**Table 176–17 REGISTERURI Procedure Parameters**

Parameter	Description
<code>schemaur1</code>	Uniquely identifies the schema document. Can be used inside <code>schemaLocation</code> attribute of XML Schema import element.
<code>schemadocuri</code>	Pathname (URI) corresponding to the physical location of the schema document. The URI path could be based on HTTP, FTP, DB or Oracle XML DB protocols. This function constructs a <code>URIType</code> instance using the <code>urifactory</code> , and invokes the <a href="#">REGISTERSchema Procedures</a> .
<code>local</code>	Determines whether this is a local or global schema. By default, all schemas are registered as local schemas, under <code>/sys/schemas/&lt;username&gt;/...</code> . If a schema is registered as global, it is added under <code>/sys/schemas/PUBLIC/...</code> . The user needs write privileges on the directory to register a global schema.
<code>gentypes</code>	Determines whether the compiler generate object types. By default, <code>TRUE</code> .
<code>genbean</code>	Determines whether the compiler generate Java beans. By default, <code>FALSE</code> .
<code>gentables</code>	Determines whether the compiler generate default tables. <code>TRUE</code> by default.
<code>force</code>	<code>TRUE</code> : schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is <code>FALSE</code> .
<code>owner</code>	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.

**Table 176–17 (Cont.) REGISTERURI Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
options	<p>Additional options to specify how the schema should be registered. The various options are represented as bits of an integer and the options parameter should be constructed by doing a BITOR of the desired bits. Possible bits:</p> <ul style="list-style-type: none"><li>REGISTER_NODOCID - this will suppress the creation of the DOCID column for out of line tables. This is a storage optimization which might be desirable when we do not need to join back to the document table (for example if we do not care about rewriting certain queries that could be rewritten by making use of the DOCID column)</li></ul>

DBMS\_XMLSTORE provides the ability to store XML data in relational tables.

**See Also:** *Oracle XML DB Developer's Guide*

This chapter contains the following sections:

- [Using DBMS\\_XMLSTORE](#)
  - Security Model
  - Types
- [Summary of DBMS\\_XMLSTORE Subprograms](#)

## Using DBMS\_XMLSTORE

- [Security Model](#)
- [Types](#)



## Security Model

Owned by XDB, the DBMS\_XMLSTORE package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Types

**Table 177-1** *Types of DBMS\_XMLSTORE*

Type	Description
ctxType	The type of the query context handle. This is the return type of <a href="#">NEWCONTEXT</a> .

---

## Summary of DBMS\_XMLSTORE Subprograms

**Table 177-2 DBMS\_XMLSTORE Package Subprograms**

Method	Description
<a href="#">CLEARKEYCOLUMNLIST</a> on page 177-6	Clears the key column list.
<a href="#">CLEARUPDATECOLUMNLIST</a> on page 177-7	Clears the update column list.
<a href="#">CLOSECONTEXT</a> on page 177-8	It closes/deallocates a particular save context.
<a href="#">DELETEXML</a> on page 177-9	Deletes records specified by data from the XML document, from the table specified at the context creation time.
<a href="#">INSERTXML</a> on page 177-10	Inserts the XML document into the table specified at the context creation time.
<a href="#">NEWCONTEXT</a> on page 177-11	Creates a save context, and returns the context handle.
<a href="#">SETKEYCOLUMN</a> on page 177-12	This method adds a column to the key column list.
<a href="#">SETROWTAG</a> on page 177-13	Names the tag used in the XML document., to enclose the XML elements corresponding to the database.
<a href="#">SETUPDATECOLUMN</a> on page 177-14	Adds a column to the "update column list".
<a href="#">UPDATEXML</a> on page 177-15	Updates the table given the XML document.

## CLEARKEYCOLUMNLIST

Clears the key column list.

### Syntax

```
PROCEDURE clearKeyColumnList(  
    ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## CLEARUPDATECOLUMNLIST

Clears the update column list.

### Syntax

```
PROCEDURE clearUpdateColumnList(  
    ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## CLOSECONTEXT

Closes/deallocates a particular save context.

### Syntax

```
PROCEDURE closeContext(  
    ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

## DELETXML

Deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted.

Syntax	Description
<pre>FUNCTION deleteXML(     ctxHdl IN ctxPType,     xDoc IN VARCHAR2) RETURN NUMBER;</pre>	Uses a VARCHAR2 type for the xDoc parameter.
<pre>FUNCTION deleteXML(     ctxHdl IN ctxType,     xDoc IN CLOB) RETURN NUMBER;</pre>	Uses a CLOB type for the xDoc parameter.
<pre>FUNCTION deleteXML(     ctxHdl IN ctxType,     xDoc IN XMLType) RETURN NUMBER;</pre>	Uses an XMLType type for the xDoc parameter.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

## INSERTXML

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted.

Note that if a user passes an XML file for insertXML to DBMS\_XMLSTORE which contains extra elements (ones that do not match up to any columns in the table), Oracle will try to insert into those columns unless [SETUPDATECOLUMN](#) is used. The use of setUpdateColumn is optional only if the elements in the XML file match up to the columns in the table.

Syntax	Description
<pre>FUNCTION insertXML(   ctxHdl IN ctxType,   xDoc IN VARCHAR2) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a VARCHAR2.
<pre>FUNCTION insertXML(   ctxHdl IN ctxType,   xDoc IN CLOB) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a CLOB.
<pre>FUNCTION insertXML(   ctxHdl IN ctxType,   xDoc IN XMLType) RETURN NUMBER;</pre>	Passes in the xDoc parameter as an XMLType.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.



## NEWCONTEXT

Creates a save context, and returns the context handle.

### Syntax

```
FUNCTION newContext(  
    targetTable IN VARCHAR2)  
RETURN ctxType;
```

Parameter	IN / OUT	Description
targetTable	(IN)	The target table into which to load the XML document.

## SETKEYCOLUMN

This method adds a column to the "key column list". The value for the column cannot be `NULL`. In case of update or delete, the columns in the key column list make up the `WHERE` clause of the statement. The key columns list must be specified before updates can complete; this is optional for delete operations

### Syntax

```
PROCEDURE setKeyColumn(  
    ctxHdl IN ctxType,  
    colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the key column list; cannot be <code>NULL</code> .

## SETROWTAG

Names the tag used in the XML document, to enclose the XML elements corresponding to database records.

### Syntax

```
PROCEDURE setRowTag(  
    ctxHdl IN ctxType,  
    tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

## SETUPDATECOLUMN

Adds a column to the update column list. In case of insert, the default is to insert values to all the columns in the table; on the other hand, in case of updates, the default is to only update the columns corresponding to the tags present in the `ROW` element of the XML document. When the update column list is specified, the columns making up this list alone will get updated or inserted into.

Note that if a user passes an XML file for [INSERTXML](#) to `DEMS_XMLSTORE` which contains extra elements (ones that do not match up to any columns in the table), Oracle will try to insert into those columns unless `setUpdateColumn` is used. The use of `setUpdateColumn` is optional only if the elements in the XML file match up to the columns in the table.

### Syntax

```
PROCEDURE setUpdateColumn(  
    ctxHdl IN ctxType,  
    colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
<code>ctxHdl</code>	(IN)	Context handle.
<code>colName</code>	(IN)	Column to be added to the update column list.

## UPDATEXML

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated. The options are described in the following table.

Syntax	Description
<pre>FUNCTION updateXML(     ctxHdl IN ctxType,     xDoc IN VARCHAR2) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a VARCHAR2.
<pre>FUNCTION updateXML(     ctxHdl IN ctxType,     xDoc IN CLOB) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a CLOB.
<pre>FUNCTION updateXML(     ctxHdl IN ctxType,     xDoc IN XMLType) RETURN NUMBER;</pre>	Passes in the xDoc parameter as a XMLType.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.



---

---

## DBMS\_XMLTRANSLATIONS

The DBMS\_XMLTRANSLATIONS package provides an interface to perform translations so that strings can be searched or displayed in various languages.

**See Also:** For more information, see the *Oracle XML DB Developer's Guide*

This chapter contains the following sections:

- [Using DBMS\\_XMLTRANSLATIONS](#)
  - Security Model
- [Summary of DBMS\\_XMLTRANSLATIONS Subprograms](#)

---

## Using DBMS\_XMLTRANSLATIONS

- [Security Model](#)



## Security Model

Owned by XDB, the DBMS\_XMLTRANSLATIONS package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

## Summary of DBMS\_XMLTRANSLATIONS Subprograms

**Table 178–1 DBMS\_XMLSTORE Package Subprograms**

<b>Method</b>	<b>Description</b>
<a href="#">DISABLETRANSLATION Procedure</a> on page 178-5	Disables translations in the current session so that query or retrieval will take place on the base document ignoring session language values
<a href="#">ENABLETRANSLATION Procedure</a> on page 178-6	Enables translations in the current session
<a href="#">EXTRACTXLIFF Function &amp; Procedure</a> on page 178-7	Extracts the translations in XLIFF format from either an XMLTYPE or a resource in the XDB Repository
<a href="#">GETBASEDOCUMENT Function</a> on page 178-10	Returns the base document with all the translations
<a href="#">MERGEXLIFF Functions</a> on page 178-11	Merges the translations in XLIFF format into either an XMLTYPE or a resource in the XDB Repository
<a href="#">SETSOURCELANG Function</a> on page 178-14	Sets the source language to a particular language at the specified XPATH
<a href="#">TRANSLATEXML Function</a> on page 178-16	Returns the document in the specified language
<a href="#">UPDATETRANSLATION Function</a> on page 178-5	Updates the translation in a particular language at the specified XPATH

## **DISABLETRANSLATION Procedure**

This procedure disables translations in the current session so that query or retrieval will take place on the base document ignoring session language values.

### **Syntax**

```
DBMS_XMLTRANSLATIONS.DISABLETRANSLATION;
```

## ENABLETRANSLATION Procedure

This procedure enables translations in the current session. This is the default behavior.

### Syntax

```
DBMS_XMLTRANSLATIONS.ENABLETRANSLATION;
```

## EXTRACTXLIFF Function & Procedure

This function and procedure extracts the translations in XLIFF format from either an XMLTYPE or a resource in the XDB Repository.

### Syntax

```
DBMS_XMLTRANSLATIONS.EXTRACTXLIFF (
  doc      IN  XMLTYPE,
  xpath    IN  VARCHAR2,
  namespace IN VARCHAR2 := NULL)
RETURN XMLTYPE;
```

```
DBMS_XMLTRANSLATIONS.EXTRACTXLIFF (
  abspath  IN  XMLTYPE,
  xpath    IN  VARCHAR2,
  namespace IN VARCHAR2 := NULL)
RETURN XMLTYPE;
```

### Parameters

**Table 178–2 EXTRACTXLIFF Function Parameters**

Parameter	Description
doc	XMLTYPE from which the XLIFF is to be extracted
xpath	XPATH at which specifies the location of the element that needs to be translated. If no XPATH is specified, the entire document is processed.
namespace	Namespace
abspath	Absolute path of the resource from which the XLIFF is to be extracted

### Return Values

The translations in the XLIFF format

### Examples

#### Extracting the Translation from an XMLTYPE

Let doc =

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <title xml:lang="es">
    Security Class Example - ES
```

```

    </title>
    <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>

```

Let the `xpath = '/securityClass/title'`. The output of `EXTRACTXLIFF` will be as follows:

```

<xliff version='1.1'>
  <file original='' source-language='en' datatype='xml'>
    <body>
      <trans-unit id='/securityClass/title'>
        <source>Security Class Example</source>
        <alt-trans>
          <target xml:lang='fr'>Security Class Example - FR</target>
          <target xml:lang='es'>Security Class Example - ES</target>
        </alt-trans>
      </trans-unit>
    </body>
  </file>
</xliff>

```

### Extracting the Translation from a Resource

Let the resource `'/public/security.xml' =`

```

<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="es">
    Security Class Example - ES
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
  <privlist>
  <privilege name="privilege1"/>
  <aggregatePrivilege name="iStorePOApprover">
    <title>
      iStore Purchase Order Approver
    </title>
    <privilegeRef name="is:privilege1"/>
    <privilegeRef name="oa:submitPO"/>
    <privilegeRef name="oa:privilege3"/>
  </aggregatePrivilege>
  <privilege name="privilege2">
    <title xml:lang="en">
      secondary privilege
    </title>
    <title xml:lang="fr" xdb:srclang="true">
      secondary privilege - FR
    </title>
    <columnRef schema="APPS" table="PurchaseOrder" column="POId"/>
    <columnRef schema="APPS" table="PurchaseOrder" column="Amount"/>
  </privilege>

```

```

    </privilege>
  </privlist>
</securityClass>

```

And let XPATH = "", then the extracted XLIFF is

```

<xliff version='1.1'>
  <file original='/public/security.xml' source-language='en' datatype='xml'>
    <body>
      <trans-unit id='/securityClass/title'>
        <source>Security Class Example</source>
        <alt-trans>
          <target xml:lang='fr'>Security Class Example - FR</target>
          <target xml:lang='es'>Security Class Example - ES</target>
        </alt-trans>
      </trans-unit>
    </body>
  </file>
  <file original='/public/security.xml' source-language='fr' datatype='xml'>
    <body>
      <trans-unit id='/securityClass/privilege[@name="privilege2"]/title'>
        <source>secondary privilege - FR</source>
        <alt-trans>
          <target xml:lang='en'>secondary privilege</target>
        </alt-trans>
      </trans-unit>
    </body>
  </file>
</xliff>

```

## GETBASEDOCUMENT Function

This function returns the base document with all the translations.

### Syntax

```
DBMS_XMLTRANSLATIONS.GETBASEDOCUMENT (
    doc    IN XMLTYPE)
RETURN XMLTYPE;
```

### Parameters

**Table 178–3** *GETBASEDOCUMENT Function Parameters*

Parameter	Description
doc	Input XMLTYPE

### Return Values

The XMLTYPE which contains the base document with all the translations

### Examples

For example, for doc =

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```

For the above document, this subprogram will return:

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```



## MERGEXLIFF Functions

This function merges the translations in XLIFF format into either an XMLTYPE or a resource in the XDB Repository.

### Syntax

```
DBMS_XMLTRANSLATIONS.MERGEXLIFF(
  doc      IN XMLTYPE,
  xliiff   IN XMLTYPE)
RETURN XMLTYPE;
```

```
DBMS_XMLTRANSLATIONS.MERGEXLIFF(
  xliiff   IN XMLTYPE);
```

### Parameters

**Table 178–4 MERGEXLIFF Function & Procedure Parameters**

Parameter	Description
doc	XMLTYPE from which the XLIFF is to be merged
xliiff	Translations in the XLIFF format

### Return Values

The result of merging 'xliiff' into 'doc' at 'xpath'

### Examples

#### Merge Translations into an XMLTYPE

Consider the following input XMLTYPE:

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="es">
    Security Class Example - ES
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
  <privlist>
  <privilege name="privilege1"/>
  <aggregatePrivilege name="iStorePOApprover">
    <title>
      iStore Purchase Order Approver
    </title>
    <privilegeRef name="is:privilege1"/>
    <privilegeRef name="oa:submitPO"/>
  </aggregatePrivilege>
</securityClass>
```

```

        <privilegeRef name="oa:privilege3"/>
    </aggregatePrivilege>
    <privilege name="privilege2">
        <title xml:lang="en">
            secondary privilege
        </title>
        <title xml:lang="fr" xdb:srclang="true">
            secondary privilege - FR
        </title>
        <columnRef schema="APPS" table="PurchaseOrder" column="POId"/>
        <columnRef schema="APPS" table="PurchaseOrder" column="Amount"/>
    </privilege>
</privlist>
</securityClass>

```

Let the input XLIFF be as follows:

```

<xliff version='1.1'>
  <file original='/public/security.xml' source-language='en' datatype='xml'>
    <body>
      <trans-unit id='/securityClass/title'>
        <source>Security Class Example Modified</source>
        <alt-trans>
          <target xml:lang='fr'>Security Class Example Mod - FR</target>
          <target xml:lang='es'>Security Class Example Mod - ES</target>
        </alt-trans>
      </trans-unit>
      <trans-unit id='/securityClass/privilege[@name="privilege2"]/title'>
        <source>secondary privilege modified</source>
        <alt-trans>
          <target xml:lang='fr'>secondary privilege mod - FR</target>
        </alt-trans>
      </trans-unit>
    </body>
  </xliff>

```

The output of merge will be as follows:

```

<securityClass xmlns="http://xmlns.oracle.com/xdm/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example Modified
  </title>
  <title xml:lang="es">
    Security Class Example Mod - ES
  </title>
  <title xml:lang="fr">
    Security Class Example Mod - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
  <privlist>
    <privilege name="privilege1"/>
    <aggregatePrivilege name="iStorePOApprover">
      <title>
        iStore Purchase Order Approver
      </title>

```

```
<privilegeRef name="is:privilege1"/>
<privilegeRef name="oa:submitPO"/>
<privilegeRef name="oa:privilege3"/>
</aggregatePrivilege>
<privilege name="privilege2">
  <title xml:lang="en" xdb:srclang="true">
    secondary privilege modified
  </title>
  <title xml:lang="fr">
    secondary privilege mod - FR
  </title>
  <columnRef schema="APPS" table="PurchaseOrder" column="POId"/>
  <columnRef schema="APPS" table="PurchaseOrder" column="Amount"/>
</privilege>
</privlist>
</securityClass>
```

### **Merge XLIFF Translations into a Resource**

If the input document in the above example were to be stored in the repository at '/public/security.xml', then merging the above XLIFF will have the same effect.

## SETSOURCELANG Function

This function sets the source language to a particular language at the specified XPATH.

### Syntax

```
DBMS_XMLTRANSLATIONS.SETSOURCELANG (
  doc      IN  XMLTYPE,
  xpath    IN  VARCHAR2,
  lang     IN  VARCHAR2,
  namespace IN VARCHAR2 := NULL)
RETURN XMLTYPE;
```

### Parameters

**Table 178–5 SETSOURCELANG Function Parameters**

Parameter	Description
doc	XMLTYPE for which the source language is to be set
xpath	XPATH at which the source language is to be set
lang	Source language
namespace	Namespace

### Return Values

The updated document

### Examples

For example, if doc =

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```

the statement

```
setSourceLang ( doc, '/securityClass/title', 'fr' )
```

produces

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
```

```
<name>
  securityClassExample
</name>
<title xml:lang="en">
  Security Class Example
</title>
<title xml:lang="fr" xdb:srclang="true">
  Security Class Example - FR
</title>
<inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```

## TRANSLATEXML Function

This function returns the document in the specified language.

### Syntax

```
DBMS_XMLTRANSLATIONS.TRANSULATEXML (
  doc    IN  XMLTYPE,
  lang   IN  VARCHAR2)
RETURN XMLTYPE;
```

### Parameters

**Table 178–6 TRANSLATEXML Function Parameters**

Parameter	Description
doc	Input XMLTYPE
lang	Language

### Return Values

The XMLTYPE which contains the document in the specified language

### Examples

For example, for doc =

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```

TRANSLATEXML (doc, 'fr') will return:

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="fr">
    Security Class Example - FR
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```

## UPDATETRANSLATION Function

This function updates the translation in a particular language at the specified XPATH.

### Syntax

```
DBMS_XMLTRANSLATIONS.UPDATETRANSLATION(
  doc      IN  XMLTYPE,
  xpath    IN  VARCHAR2,
  lang     IN  VARCHAR2,
  value    IN  VARCHAR2,
  namespace IN VARCHAR2 := NULL)
RETURN XMLTYPE;
```

### Parameters

**Table 178–7** UPDATETRANSLATION Function Parameters

Parameter	Description
doc	XMLTYPE for which the translation is to be updated
xpath	XPATH at which the translation is to be updated
lang	Language for which the translation is to be updated
value	New translation
namespace	Namespace

### Return Values

The updated document

### Examples

For example,

```
updateTranslation ( doc, '/securityClass/title/text()', 'fr', 'Oracle' );
```

produces

```
<securityClass xmlns="http://xmlns.oracle.com/xdb/security.xsd"
  xmlns:is="xmlns.oracle.com/iStore"
  xmlns:oa="xmlns.oracle.com/OracleApps"
  targetNamespace="xmlns.oracle.com/example">
  <name>
    securityClassExample
  </name>
  <title xml:lang="en" xdb:srclang="true">
    Security Class Example
  </title>
  <title xml:lang="fr">
    Oracle
  </title>
  <inherits-from>is:iStorePurchaseOrder</inherits-from>
</securityClass>
```





The `DBMS_XPLAN` package provides an easy way to display the output of the `EXPLAIN PLAN` command in several, predefined formats. You can also use the `DBMS_XPLAN` package to display the plan of a statement stored in the Automatic Workload Repository (AWR) or stored in a SQL tuning set. It further provides a way to display the SQL execution plan and SQL execution runtime statistics for cached SQL cursors based on the information stored in the `V$SQL_PLAN` and `V$SQL_PLAN_STATISTICS_ALL` fixed views. Finally, it displays plans from a SQL plan baseline.

**See Also:**

- For more information on the `EXPLAIN PLAN` command, the AWR, and SQL tuning set, see *Oracle Database Performance Tuning Guide*.
- For more information on the `V$SQL_PLAN` and `V$SQL_PLAN_STATISTICS` fixed views, see *Oracle Database Reference*.

This chapter contains the following topics:

- [Using DBMS\\_XPLAN](#)
  - Overview
  - Security Model
  - Examples
- [Summary of DBMS\\_XPLAN Subprograms](#)

## Using DBMS\_XPLAN

- [Overview](#)
- [Security Model](#)
- [Examples](#)

## Overview

The DBMS\_XPLAN package supplies five table functions:

- DISPLAY - to format and display the contents of a plan table.
- DISPLAY\_AWR - to format and display the contents of the execution plan of a stored SQL statement in the AWR.
- DISPLAY\_CURSOR - to format and display the contents of the execution plan of any loaded cursor.
- DISPLAY\_SQL\_PLAN\_BASELINE - to display one or more execution plans for the SQL statement identified by SQL handle
- DISPLAY\_SQLSET - to format and display the contents of the execution plan of statements stored in a SQL tuning set.

## Security Model

This package runs with the privileges of the calling user, not the package owner (SYS). The table function `DISPLAY_CURSOR` requires to have select privileges on the following fixed views: `V$SQL_PLAN`, `V$SESSION` and `V$SQL_PLAN_STATISTICS_ALL`.

Using the [DISPLAY\\_AWR Function](#) requires the user to have `SELECT` privileges on `DBA_HIST_SQL_PLAN`, `DBA_HIST_SQLTEXT`, and `V$DATABASE`.

Using the [DISPLAY\\_SQLSET Function](#) requires the user to have the `SELECT` privilege on `ALL_SQLSET_STATEMENTS` and `ALL_SQLSET_PLANS`.

Using [DISPLAY\\_SQL\\_PLAN\\_BASELINE Function](#) requires the user to have the `SELECT` privilege on `DBA_SQL_PLAN_BASELINES` as well as the privileges to execute the SQL statement for which the user is trying to get the plan.

All these privileges are automatically granted as part of the `SELECT_CATALOG` role.

## Examples

### Displaying a Plan Table Using DBMS\_XPLAN.DISPLAY

Execute an explain plan command on a `SELECT` statement:

```
EXPLAIN PLAN FOR
SELECT * FROM emp e, dept d
      WHERE e.deptno = d.deptno
      AND e.ename='benoit';
```

Display the plan using the `DBMS_XPLAN.DISPLAY` table function

```
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

This query produces the following output:

Plan hash value: 3693697075

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	57	6 (34)	00:00:01
* 1	HASH JOIN		1	57	6 (34)	00:00:01
* 2	TABLE ACCESS FULL	EMP	1	37	3 (34)	00:00:01
3	TABLE ACCESS FULL	DEPT	4	80	3 (34)	00:00:01

Predicate Information (identified by operation id):

```
1 - access("E"."DEPTNO"="D"."DEPTNO")
2 - filter("E"."ENAME"='benoit')
```

15 rows selected.

### Displaying a Cursor Execution Plan Using DBMS\_XPLAN.DISPLAY\_CURSOR

By default, the table function `DISPLAY_CURSOR` formats the execution plan for the last SQL statement executed by the session. For example:

```
SELECT ename FROM emp e, dept d
      WHERE e.deptno = d.deptno
      AND e.empno=7369;
```

ENAME

-----

SMITH

To display the execution plan of the last executed statement for that session:

```
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR);
```

This query produces the following output:

Plan hash value: 3693697075, SQL hash value: 2096952573, child number: 0

```
SELECT ename FROM emp e, dept d WHERE e.deptno = d.deptno
AND e.empno=7369
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT					
* 1	HASH JOIN		1	16	6 (34)	00:00:01
* 2	TABLE ACCESS FULL	EMP	1	13	3 (34)	00:00:01
3	TABLE ACCESS FULL	DEPT	4	12	3 (34)	00:00:01

Predicate Information (identified by operation id):

- 1 - access("E"."DEPTNO"="D"."DEPTNO")
- 2 - filter("E"."EMPNO"=7369)

21 rows selected.

You can also use the table function `DISPLAY_CURSOR` to display the execution plan for any loaded cursor stored in the cursor cache. In that case, you must supply a reference to the child cursor to the table function. This includes the SQL ID of the statement and optionally the child number.

Run a query with a distinctive comment:

```
SELECT /* TOTO */ ename, dname
FROM dept d join emp e USING (deptno);
```

Get `sql_id` and `child_number` for the preceding statement:

```
SELECT sql_id, child_number
FROM v$sql
WHERE sql_text LIKE '%TOTO%';
```

SQL_ID	CHILD_NUMBER
gwp663cqh5qbf	0

Display the execution plan for the cursor:

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR('gwp663cqh5qbf',0));
```

Plan hash value: 3693697075, SQL ID: gwp663cqh5qbf, child number: 0

```
SELECT /* TOTO */ ename, dname
FROM dept d JOIN emp e USING (deptno);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				7 (100)	
1	SORT GROUP BY		4	64	7 (43)	00:00:01
* 2	HASH JOIN		14	224	6 (34)	00:00:01
3	TABLE ACCESS FULL	DEPT	4	44	3 (34)	00:00:01
4	TABLE ACCESS FULL	EMP	14	70	3 (34)	00:00:01

Predicate Information (identified by operation id):

- 2 - access("E"."DEPTNO"="D"."DEPTNO")

Instead of issuing two queries, one to get the `sql_id` and `child_number` pair and one to display the plan, you can combine these in a single query:

Display the execution plan of all cursors matching the string 'TOTO':

```
SELECT t.*
FROM v$sql s, table(DBMS_XPLAN.DISPLAY_CURSOR(s.sql_id, s.child_number)) t WHERE
sql_text LIKE '%TOTO%';
```

### Displaying a Plan Table with Parallel Information

By default, only relevant information is reported by the display and display\_cursor table functions. In [Displaying a Plan Table Using DBMS\\_XPLAN.DISPLAY](#) on page 179-5, the query does not execute in parallel. Hence, information related to the parallelization of the plan is not reported. As shown in the following example, parallel information is reported only if the query executes in parallel.

```
ALTER TABLE emp PARALLEL;
EXPLAIN PLAN for
SELECT * FROM emp e, dept d
WHERE e.deptno = d.deptno
AND e.ename = 'hermann'
ORDER BY e.empno;
```

### Display the plan using the DBMS\_XPLAN.DISPLAY table function

```
SET LINESIZE 130
SET PAGESIZE 0
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
Plan hash value: 3693697345
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	TQ	INOUT	PQ Distrib
0	SELECT STATEMENT		1	117	6 (50)	00:00:01			
1	PX COORDINATOR								
2	PX SEND QC (ORDER)	:TQ10003	1	117	6 (50)	00:00:01	Q1,03	P->S	QC (ORDER)
3	SORT ORDER BY		1	117	6 (50)	00:00:01	Q1,03	PCWP	
4	PX RECEIVE		1	117	5 (40)	00:00:01	Q1,03	PCWP	
5	PX SEND RANGE	:TQ10002	1	117	5 (40)	00:00:01	Q1,02	P->P	RANGE
* 6	HASH JOIN		1	117	5 (40)	00:00:01	Q1,02	PCWP	
7	PX RECEIVE		1	87	2 (50)	00:00:01	Q1,02	PCWP	
8	PX SEND HASH	:TQ10001	1	87	2 (50)	00:00:01	Q1,01	P->P	HASH
9	PX BLOCK ITERATOR		1	87	2 (50)	00:00:01	Q1,01	PCWC	
* 10	TABLE ACCESS FULL	EMP	1	87	2 (50)	00:00:01	Q1,01	PCWP	
11	BUFFER SORT						Q1,02	PCWC	
12	PX RECEIVE		4	120	3 (34)	00:00:01	Q1,02	PCWP	
13	PX SEND HASH	:TQ10000	4	120	3 (34)	00:00:01		S->P	HASH
14	TABLE ACCESS FULL	DEPT	4	120	3 (34)	00:00:01			

Predicate Information (identified by operation id):

```
6 - access("E"."DEPTNO"="D"."DEPTNO")
10 - filter("E"."ENAME"='hermann')
```

When the query is parallel, information related to parallelism is reported: table queue number (TQ column), table queue type (INOUT) and table queue distribution method (PQ Distrib).

By default, if several plans in the plan table match the statement\_id parameter passed to the display table function (default value is NULL), only the plan corresponding to the last EXPLAIN PLAN command is displayed. Hence, there is no need to purge the plan table after each EXPLAIN PLAN. However, you should purge the plan table regularly to ensure good performance in the execution of the DISPLAY table function. If no plan table is created, Oracle uses a global temporary table to store any plan information for individual users and preserves its content throughout the

lifespan of a session. Note that you cannot truncate the content of a global temporary table.

For ease of use, you can define a view on top of the display table function and then use that view to display the output of the EXPLAIN PLAN command:

### **Using a View to Display Last Explain Plan**

```
# define plan view
CREATE VIEW PLAN AS SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

# display the output of the last explain plan command
SELECT * FROM PLAN;
```



---

## Summary of DBMS\_XPLAN Subprograms

**Table 179-1 DBMS\_XPLAN Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DISPLAY Function</a> on page 179-10	Displays the contents of the plan table
<a href="#">DISPLAY_AWR Function</a> on page 179-13	Displays the contents of an execution plan stored in the AWR
<a href="#">DISPLAY_CURSOR Function</a> on page 179-16	Displays the execution plan of any cursor in the cursor cache
<a href="#">DISPLAY_PLAN Function</a> on page 179-19	Displays the contents of the plan table in a variety of formats with CLOB output type
<a href="#">DISPLAY_SQL_PLAN_BASELINE Function</a> on page 179-22	Displays one or more execution plans for the specified SQL handle of a SQL plan baseline
<a href="#">DISPLAY_SQLSET Function</a> on page 179-24	Displays the execution plan of a given statement stored in a SQL tuning set

## DISPLAY Function

This table function displays the contents of the plan table.

In addition, you can use this table function to display any plan (with or without statistics) stored in a table as long as the columns of this table are named the same as columns of the plan table (or `V$SQL_PLAN_STATISTICS_ALL` if statistics are included). You can apply a predicate on the specified table to select rows of the plan to display.

### Syntax

```
DBMS_XPLAN.DISPLAY(
  table_name      IN VARCHAR2 DEFAULT 'PLAN_TABLE',
  statement_id    IN VARCHAR2 DEFAULT NULL,
  format          IN VARCHAR2 DEFAULT 'TYPICAL',
  filter_preds    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 179–2** *DISPLAY Function Parameters*

Parameter	Description
table_name	Specifies the table name where the plan is stored. This parameter defaults to <code>PLAN_TABLE</code> , which is the default plan table for the <code>EXPLAIN PLAN</code> command. If <code>NULL</code> is specified it also defaults to <code>PLAN_TABLE</code> .
statement_id	Specifies the <code>statement_id</code> of the plan to be displayed. This parameter defaults to <code>NULL</code> , which is the default when the <code>EXPLAIN PLAN</code> command is executed without a <code>set statement_id</code> clause. If no <code>statement_id</code> is specified, the function shows you the plan of the most recent explained statement.

**Table 179–2 (Cont.) DISPLAY Function Parameters**

Parameter	Description
format	<p>Controls the level of details for the plan. It accepts four values:</p> <ul style="list-style-type: none"> <li>■ <b>BASIC</b>: Displays the minimum information in the plan—the operation ID, the operation name and its option.</li> <li>■ <b>TYPICAL</b>: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only <b>PROJECTION</b>, <b>ALIAS</b> and <b>REMOTE SQL</b> information (see below).</li> <li>■ <b>SERIAL</b>: Like <b>TYPICAL</b> except that the parallel information is not displayed, even if the plan executes in parallel.</li> <li>■ <b>ALL</b>: Maximum user level. Includes information displayed with the <b>TYPICAL</b> level with additional information (<b>PROJECTION</b>, <b>ALIAS</b> and information about <b>REMOTE SQL</b> if the operation is distributed).</li> </ul> <p>For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as <b>PARTITION</b>) or logical additions to the base plan table output (such as <b>PREDICATE</b>). Format keywords must be separated by either a comma or a space:</p> <ul style="list-style-type: none"> <li>■ <b>ROWS</b> - if relevant, shows the number of rows estimated by the optimizer</li> <li>■ <b>BYTES</b> - if relevant, shows the number of bytes estimated by the optimizer</li> <li>■ <b>COST</b> - if relevant, shows optimizer cost information</li> <li>■ <b>PARTITION</b> - if relevant, shows partition pruning information</li> <li>■ <b>PARALLEL</b> - if relevant, shows PX information (distribution method and table queue information)</li> <li>■ <b>PREDICATE</b> - if relevant, shows the predicate section</li> <li>■ <b>PROJECTION</b> -if relevant, shows the projection section</li> <li>■ <b>ALIAS</b> - if relevant, shows the "Query Block Name / Object Alias" section</li> <li>■ <b>REMOTE</b> - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</li> <li>■ <b>NOTE</b> - if relevant, shows the note section of the explain plan</li> </ul> <p>Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.</p> <p>If the target plan table (see <code>table_name</code> parameter) also stores plan statistics columns (for example, it is a table used to capture the content of the fixed view <code>V\$SQL_PLAN_STATISTICS_ALL</code>), additional format keywords can be used to specify which class of statistics to display when using the <b>DISPLAY Function</b>. These additional format keywords are <b>IOSTATS</b>, <b>MEMSTATS</b>, <b>ALLSTATS</b> and <b>LAST</b> (see the <b>DISPLAY_CURSOR Function</b> or the <b>DISPLAY_SQLSET Function</b> for a full description of these four keywords).</p>

**Table 179–2 (Cont.) DISPLAY Function Parameters**

Parameter	Description
<code>filter_preds</code>	<p>SQL filter predicate(s) to restrict the set of rows selected from the table where the plan is stored. When value is <code>NULL</code> (the default), the plan displayed corresponds to the last executed explain plan. For example: <code>filter_preds=&gt;'plan_id = 10'</code></p> <p>Can reference any column of the table where the plan is stored and can contain any SQL construct (for example, sub-query, function calls (see <b>WARNING</b> under Usage Notes)</p>

## Usage Notes

Here are some ways you might use variations on the `format` parameter:

- Use `'ALL -PROJECTION -NOTE'` to display everything except the projection and note sections.
- Use `'TYPICAL PROJECTION'` to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply `'PROJECTION'` is equivalent.
- Use `'-BYTES -COST -PREDICATE'` to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use `'BASIC ROWS'` to display basic information with the additional number of rows estimated by the optimizer.

---

**WARNING:** Application developers should expose the `filter_preds` parameter to end-users only after careful consideration because this could expose the application to SQL injection. Indeed, `filter_preds` can potentially reference any table or execute any server function for which the database user invoking the table function has privileges.

---

## Examples

To display the result of the last `EXPLAIN PLAN` command stored in the plan table:

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY);
```

To display from other than the default plan table, "my\_plan\_table":

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY('my_plan_table'));
```

To display the minimum plan information:

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY('plan_table', null, 'basic'));
```

To display the plan for a statement identified by 'foo', such as `statement_id='foo'`:

```
SELECT * FROM table (DBMS_XPLAN.DISPLAY('plan_table', 'foo'));
```

## DISPLAY\_AWR Function

This table function displays the contents of an execution plan stored in the AWR.

### Syntax

```
DBMS_XPLAN.DISPLAY_AWR (
  sql_id          IN      VARCHAR2,
  plan_hash_value IN      NUMBER DEFAULT NULL,
  db_id           IN      NUMBER DEFAULT NULL,
  format          IN      VARCHAR2 DEFAULT TYPICAL);
```

### Parameters

**Table 179–3** *DISPLAY\_AWR Table Function Parameters*

Parameter	Description
sql_id	Specifies the SQL_ID of the SQL statement. You can retrieve the appropriate value for the SQL statement of interest by querying the column SQL_ID in DBA_HIST_SQLTEXT.
plan_hash_value	Specifies the PLAN_HASH_VALUE of a SQL statement. This parameter is optional. If omitted, the table function returns all stored execution plans for a given SQL_ID.
db_id	Specifies the database_id for which the plan of the SQL statement, identified by SQL_ID should be displayed. If not supplied, the database_id of the local database is used, as shown in V\$DATABASE.
format	Controls the level of details for the plan. It accepts four values: <ul style="list-style-type: none"> <li>■ <b>BASIC:</b> Displays the minimum information in the plan—the operation ID, the operation name and its option.</li> <li>■ <b>TYPICAL:</b> This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</li> <li>■ <b>SERIAL:</b> Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</li> <li>■ <b>ALL:</b> Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</li> </ul>

**Table 179–3 (Cont.) DISPLAY\_AWR Table Function Parameters**

Parameter	Description
	<p>For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as <code>PARTITION</code>) or logical additions to the base plan table output (such as <code>PREDICATE</code>). Format keywords must be separated by either a comma or a space:</p> <ul style="list-style-type: none"> <li>■ <code>ROWS</code> - if relevant, shows the number of rows estimated by the optimizer</li> <li>■ <code>BYTES</code> - if relevant, shows the number of bytes estimated by the optimizer</li> <li>■ <code>COST</code> - if relevant, shows optimizer cost information</li> <li>■ <code>PARTITION</code> - if relevant, shows partition pruning information</li> <li>■ <code>PARALLEL</code> - if relevant, shows PX information (distribution method and table queue information)</li> <li>■ <code>PREDICATE</code> - if relevant, shows the predicate section</li> <li>■ <code>PROJECTION</code> -if relevant, shows the projection section</li> <li>■ <code>ALIAS</code> - if relevant, shows the "Query Block Name / Object Alias" section</li> <li>■ <code>REMOTE</code> - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</li> <li>■ <code>NOTE</code> - if relevant, shows the note section of the explain plan</li> </ul> <p>Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.</p>

## Usage Notes

- To use the `DISPLAY_AWR` functionality, the calling user must have `SELECT` privilege on `DBA_HIST_SQL_PLAN`, `DBA_HIST_SQLTEXT`, and `V$DATABASE`, otherwise it shows an appropriate error message.
- Here are some ways you might use variations on the `format` parameter:
  - Use `'ALL -PROJECTION -NOTE'` to display everything except the projection and note sections.
  - Use `'TYPICAL PROJECTION'` to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply `'PROJECTION'` is equivalent.
  - Use `'-BYTES -COST -PREDICATE'` to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
  - Use `'BASIC ROWS'` to display basic information with the additional number of rows estimated by the optimizer.

## Examples

To display the different execution plans associated with the SQL ID 'atfwcg8anrykp':

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_AWR('atfwcg8anrykp'));
```

To display all execution plans of all stored SQL statements containing the string 'TOTO':

```
SELECT tf.* FROM DBA_HIST_SQLTEXT ht, table
      (DBMS_XPLAN.DISPLAY_AWR(ht.sql_id,null, null, 'ALL' )) tf
WHERE ht.sql_text like '%TOTO%';
```

## DISPLAY\_CURSOR Function

This table function displays the explain plan of any cursor loaded in the cursor cache. In addition to the explain plan, various plan statistics (such as I/O, memory and timing) can be reported (based on the V\$SQL\_PLAN\_STATISTICS\_ALL VIEWS).

### Syntax

```
DBMS_XPLAN.DISPLAY_CURSOR(
  sql_id          IN VARCHAR2 DEFAULT NULL,
  cursor_child_no IN NUMBER   DEFAULT 0,
  format         IN VARCHAR2  DEFAULT 'TYPICAL');
```

### Parameters

**Table 179–4 DISPLAY\_CURSOR Function Parameters**

Parameter	Description
sql_id	Specifies the SQL_ID of the SQL statement in the cursor cache. You can retrieve the appropriate value by querying the column SQL_ID in V\$SQL or V\$SQLAREA. Alternatively, you could choose the column PREV_SQL_ID for a specific session out of V\$SESSION. This parameter defaults to NULL in which case the plan of the last cursor executed by the session is displayed.
cursor_child_no	Child number of the cursor to display. If not supplied, the execution plan of all cursors matching the supplied sql_id parameter are displayed. The child_number can be specified only if sql_id is specified.
format	Controls the level of details for the plan. It accepts four values: <ul style="list-style-type: none"> <li>■ BASIC: Displays the minimum information in the plan—the operation ID, the operation name and its option.</li> <li>■ TYPICAL: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</li> <li>■ SERIAL: Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</li> <li>■ ALL: Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</li> </ul> <p>For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE).</p>



**Table 179-4 (Cont.) DISPLAY\_CURSOR Function Parameters**

Parameter	Description
	Format keywords must be separated by either a comma or a space:
	<ul style="list-style-type: none"> <li>■ ROWS - if relevant, shows the number of rows estimated by the optimizer</li> <li>■ BYTES - if relevant, shows the number of bytes estimated by the optimizer</li> <li>■ COST - if relevant, shows optimizer cost information</li> <li>■ PARTITION - if relevant, shows partition pruning information</li> <li>■ PARALLEL - if relevant, shows PX information (distribution method and table queue information)</li> <li>■ PREDICATE - if relevant, shows the predicate section</li> <li>■ PROJECTION - if relevant, shows the projection section</li> <li>■ ALIAS - if relevant, shows the "Query Block Name / Object Alias" section</li> <li>■ REMOTE - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</li> <li>■ NOTE - if relevant, shows the note section of the explain plan</li> <li>■ IOSTATS - assuming that basic plan statistics are collected when SQL statements are executed (either by using the <code>gather_plan_statistics</code> hint or by setting the parameter <code>statistics_level</code> to ALL), this format shows IO statistics for ALL (or only for the LAST as shown below) executions of the cursor.</li> <li>■ MEMSTATS - Assuming that PGA memory management is enabled (that is, <code>pga_aggregate_target</code> parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators.</li> <li>■ ALLSTATS - A shortcut for 'IOSTATS MEMSTATS'</li> <li>■ LAST - By default, plan statistics are shown for all executions of the cursor. The keyword LAST can be specified to see only the statistics for the last execution.</li> </ul>
	The following two formats are deprecated but supported for backward compatibility:
	<ul style="list-style-type: none"> <li>■ RUNSTATS_TOT - Same as IOSTATS, that is, displays IO statistics for all executions of the specified cursor.</li> <li>■ RUNSTATS_LAST - Same as IOSTATS LAST, that is, displays the runtime statistics for the last execution of the cursor</li> </ul>
	Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.

## Usage Notes

- To use the DISPLAY\_CURSOR functionality, the calling user must have SELECT privilege on the fixed views V\$SQL\_PLAN\_STATISTICS\_ALL, V\$SQL and V\$SQL\_PLAN, otherwise it shows an appropriate error message.
- Here are some ways you might use variations on the format parameter:

- Use 'ALL -PROJECTION -NOTE' to display everything except the projection and note sections.
- Use 'TYPICAL PROJECTION' to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply 'PROJECTION' is equivalent.
- Use '-BYTES -COST -PREDICATE' to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use 'BASIC ROWS' to display basic information with the additional number of rows estimated by the optimizer.

## Examples

To display the execution plan of the last SQL statement executed by the current session:

```
SELECT * FROM table (  
    DBMS_XPLAN.DISPLAY_CURSOR);
```

To display the execution plan of all children associated with the SQL ID 'atfwcg8anrykp':

```
SELECT * FROM table (  
    DBMS_XPLAN.DISPLAY_CURSOR('atfwcg8anrykp'));
```

To display runtime statistics for the cursor included in the preceding statement:

```
SELECT * FROM table (  
    DBMS_XPLAN.DISPLAY_CURSOR('atfwcg8anrykp', NULL, 'ALLSTATS LAST');
```

## DISPLAY\_PLAN Function

This table function displays the contents of the plan table in a variety of formats with CLOB output type.

### Syntax

```
DBMS_XPLAN.DISPLAY_PLAN (
  table_name      IN   VARCHAR2  DEFAULT 'PLAN_TABLE',
  statement_id    IN   VARCHAR2  DEFAULT NULL,
  format          IN   VARCHAR2  DEFAULT 'TYPICAL',
  filter_preds    IN   VARCHAR2  DEFAULT NULL,
  type            IN   VARCHAR2  DEFAULT 'TEXT')
RETURN CLOB;
```

### Parameters

**Table 179-5** *DISPLAY\_PLAN Function Parameters*

Parameter	Description
table_name	Specifies the table name where the plan is stored. This parameter defaults to <code>PLAN_TABLE</code> , which is the default plan table for the <code>EXPLAIN PLAN</code> command. If <code>NULL</code> is specified it also defaults to <code>PLAN_TABLE</code> .
statement_id	Specifies the <code>statement_id</code> of the plan to be displayed. This parameter defaults to <code>NULL</code> , which is the default when the <code>EXPLAIN PLAN</code> command is executed without a <code>set statement_id</code> clause. If no <code>statement_id</code> is specified, the function shows you the plan of the most recent explained statement.
filter_preds	SQL filter predicate(s) to restrict the set of rows selected from the table where the plan is stored. When value is <code>NULL</code> (the default), the plan displayed corresponds to the last executed explain plan. For example: <code>filter_preds=&gt;'plan_id = 10'</code>  Can reference any column of the table where the plan is stored and can contain any SQL construct (for example, sub-query, function calls (see <b>WARNING</b> under Usage Notes)

**Table 179–5 (Cont.) DISPLAY\_PLAN Function Parameters**

Parameter	Description
format	<p>Controls the level of details for the plan. It accepts four values:</p> <ul style="list-style-type: none"> <li>■ <b>BASIC:</b> Displays the minimum information in the plan—the operation ID, the operation name and its option.</li> <li>■ <b>TYPICAL:</b> This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only PROJECTION, ALIAS and REMOTE SQL information (see below).</li> <li>■ <b>SERIAL:</b> Like TYPICAL except that the parallel information is not displayed, even if the plan executes in parallel.</li> <li>■ <b>ALL:</b> Maximum user level. Includes information displayed with the TYPICAL level with additional information (PROJECTION, ALIAS and information about REMOTE SQL if the operation is distributed).</li> </ul> <p>For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as PARTITION) or logical additions to the base plan table output (such as PREDICATE). Format keywords must be separated by either a comma or a space:</p> <ul style="list-style-type: none"> <li>■ <b>ROWS</b> - if relevant, shows the number of rows estimated by the optimizer</li> <li>■ <b>BYTES</b> - if relevant, shows the number of bytes estimated by the optimizer</li> <li>■ <b>COST</b> - if relevant, shows optimizer cost information</li> <li>■ <b>PARTITION</b> - if relevant, shows partition pruning information</li> <li>■ <b>PARALLEL</b> - if relevant, shows PX information (distribution method and table queue information)</li> <li>■ <b>PREDICATE</b> - if relevant, shows the predicate section</li> <li>■ <b>PROJECTION</b> -if relevant, shows the projection section</li> <li>■ <b>ALIAS</b> - if relevant, shows the "Query Block Name / Object Alias" section</li> <li>■ <b>REMOTE</b> - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</li> <li>■ <b>NOTE</b> - if relevant, shows the note section of the explain plan</li> </ul> <p>Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-PROJECTION' excludes projection information.</p> <p>If the target plan table (see table_name parameter) also stores plan statistics columns (for example, it is a table used to capture the content of the fixed view V\$SQL_PLAN_STATISTICS_ALL), additional format keywords can be used to specify which class of statistics to display when using the <a href="#">DISPLAY Function</a>. These additional format keywords are IOSTATS, MEMSTATS, ALLSTATS and LAST (see the <a href="#">DISPLAY_CURSOR Function</a> or the <a href="#">DISPLAY_SQLSET Function</a> for a full description of these four keywords).</p>
type	<p>Output type, one of: 'TEXT', 'ACTIVE', 'HTML', or 'XML' (see Usage Notes regarding type ACTIVE).</p>

## Return Values

Returns the requested report as CLOB

## Usage Notes

Active reports have a rich, interactive user interface akin to that found in Enterprise Manager while not requiring any EM installation. The report file built is in HTML format, so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.

---

---

**WARNING:** Application developers should expose the `filter_preds` parameter to end-users only after careful consideration because this could expose the application to SQL injection. Indeed, `filter_preds` can potentially reference any table or execute any server function for which the database user invoking the table function has privileges.

---

---

## DISPLAY\_SQL\_PLAN\_BASELINE Function

This table function displays one or more execution plans for the specified SQL handle of a SQL plan baseline.

### Syntax

```
DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE (
  sql_handle      IN VARCHAR2 := NULL,
  plan_name       IN VARCHAR2 := NULL,
  format          IN VARCHAR2 := 'TYPICAL')
RETURN dbms_xplan_type_table;
```

### Parameters

**Table 179–6** DISPLAY\_SQL\_PLAN\_BASELINE Function Parameters

Parameter	Description
sql_handle	SQL statement handle. It identifies a SQL statement whose plan(s) are to be displayed.
plan_name	Plan name. It identifies a specific plan. Default NULL means all plans associated with identified SQL statement are explained and displayed.
format	Format string determines what information stored in the plan displayed. One of three format values ('BASIC', 'TYPICAL', 'ALL') can be used, each representing a common use case.

### Return Values

A PL/SQL type table

### Usage Notes

This procedure uses plan information stored in the plan baseline to explain and display the plans. It is possible that the `plan_id` stored in the SQL management base may not match with the `plan_id` of the generated plan. A mismatch between stored `plan_id` and generated `plan_id` means that it is a non-reproducible plan. Such a plan is deemed invalid and is bypassed by the optimizer during SQL compilation.

### Examples

#### Display all plans of a SQL statement identified by the SQL handle 'SYS\_SQL\_b1d49f6074ab95af' using TYPICAL format

```
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.*
  FROM TABLE(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(
              'SYS_SQL_b1d49f6074ab95af')) t;
```

#### Display all plans of one or more SQL statements containing the string 'HR2' using BASIC format

```
SET LINESIZE 150
SET PAGESIZE 2000
SELECT t.*
  FROM (SELECT DISTINCT sql_handle FROM dba_sql_plan_baselines
```

```
WHERE sql_text like '%HR2%') pb,  
TABLE(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(pb.sql_handle, NULL,  
      'BASIC')) t;
```

## DISPLAY\_SQLSET Function

This table function displays the execution plan of a given statement stored in a SQL tuning set.

### Syntax

```
DBMS_XPLAN.DISPLAY_SQLSET(
  sqlset_name      IN VARCHAR2,
  sql_id           IN VARCHAR2,
  plan_hash_value  IN NUMBER := NULL,
  format          IN VARCHAR2 := 'TYPICAL',
  sqlset_owner     IN VARCHAR2 := NULL)
RETURN DBMS_XPLAN_TYPE_TABLE PIPELINED;
```

### Parameters

**Table 179–7 DISPLAY\_SQLSET Function Parameters**

Parameter	Description
sqlset_name	Name of the SQL Tuning Set
sql_id	Specifies the sql_id value for a SQL statement having its plan stored in the SQL tuning set. You can find all stored SQL statements by querying table function <code>DBMS_SQLTUNE.SELECT_SQLSET</code>
plan_hash_value	Optional parameter. Identifies a specific stored execution plan for a SQL statement. If suppressed, all stored execution plans are shown.
format	Controls the level of details for the plan. It accepts four values: <ul style="list-style-type: none"> <li>▪ <b>BASIC</b>: Displays the minimum information in the plan—the operation ID, the operation name and its option.</li> <li>▪ <b>TYPICAL</b>: This is the default. Displays the most relevant information in the plan (operation id, name and option, #rows, #bytes and optimizer cost). Pruning, parallel and predicate information are only displayed when applicable. Excludes only <code>PROJECTION</code>, <code>ALIAS</code> and <code>REMOTE SQL</code> information (see below).</li> <li>▪ <b>SERIAL</b>: Like <b>TYPICAL</b> except that the parallel information is not displayed, even if the plan executes in parallel.</li> <li>▪ <b>ALL</b>: Maximum user level. Includes information displayed with the <b>TYPICAL</b> level with additional information (<code>PROJECTION</code>, <code>ALIAS</code> and information about <code>REMOTE SQL</code> if the operation is distributed).</li> </ul>



**Table 179-7 (Cont.) DISPLAY\_SQLSET Function Parameters**

Parameter	Description
	<p>For finer control on the display output, the following keywords can be added to the above three standard format options to customize their default behavior. Each keyword either represents a logical group of plan table columns (such as <code>PARTITION</code>) or logical additions to the base plan table output (such as <code>PREDICATE</code>). Format keywords must be separated by either a comma or a space:</p> <ul style="list-style-type: none"> <li>■ <code>ROWS</code> - if relevant, shows the number of rows estimated by the optimizer</li> <li>■ <code>BYTES</code> - if relevant, shows the number of bytes estimated by the optimizer</li> <li>■ <code>COST</code> - if relevant, shows optimizer cost information</li> <li>■ <code>PARTITION</code> - if relevant, shows partition pruning information</li> <li>■ <code>PARALLEL</code> - if relevant, shows PX information (distribution method and table queue information)</li> <li>■ <code>PREDICATE</code> - if relevant, shows the predicate section</li> <li>■ <code>PROJECTION</code> - if relevant, shows the projection section</li> <li>■ <code>ALIAS</code> - if relevant, shows the "Query Block Name / Object Alias" section</li> <li>■ <code>REMOTE</code> - if relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)</li> <li>■ <code>NOTE</code> - if relevant, shows the note section of the explain plan</li> <li>■ <code>IOSTATS</code> - assuming that basic plan statistics are collected when SQL statements are executed (either by using the <code>gather_plan_statistics</code> hint or by setting the parameter <code>statistics_level</code> to <code>ALL</code>), this format shows IO statistics for <code>ALL</code> (or only for the <code>LAST</code> as shown below) executions of the cursor.</li> <li>■ <code>MEMSTATS</code> - Assuming that PGA memory management is enabled (that is, <code>pga_aggregate_target</code> parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators.</li> <li>■ <code>ALLSTATS</code> - A shortcut for ' <code>IOSTATS MEMSTATS</code> '</li> <li>■ <code>LAST</code> - By default, plan statistics are shown for all executions of the cursor. The keyword <code>LAST</code> can be specified to see only the statistics for the last execution.</li> </ul> <p>The following two formats are deprecated but supported for backward compatibility:</p> <ul style="list-style-type: none"> <li>■ <code>RUNSTATS_TOT</code> - Same as <code>IOSTATS</code> , that is, displays IO statistics for all executions of the specified cursor.</li> <li>■ <code>RUNSTATS_LAST</code> - Same as <code>IOSTATS LAST</code>, that is, displays the runtime statistics for the last execution of the cursor</li> </ul> <p>Format keywords can be prefixed by the sign '-' to exclude the specified information. For example, '-<code>PROJECTION</code>' excludes projection information.</p>
<code>sqlset_owner</code>	The owner of the SQL tuning set. The default is the current user.

## Usage Notes

Here are some ways you might use variations on the `format` parameter:

- Use `'ALL -PROJECTION -NOTE'` to display everything except the projection and note sections.
- Use `'TYPICAL PROJECTION'` to display using the typical format with the additional projection section (which is normally excluded under the typical format). Since typical is default, using simply `'PROJECTION'` is equivalent.
- Use `'-BYTES -COST -PREDICATE'` to display using the typical format but excluding optimizer cost and byte estimates as well as the predicate section.
- Use `'BASIC ROWS'` to display basic information with the additional number of rows estimated by the optimizer.

## Examples

To display the execution plan for the SQL statement associated with SQL ID `'gwp663cqh5qbf'` and PLAN HASH 3693697075 in the SQL Tuning Set called `'OLTP_optimization_0405'`:

```
SELECT * FROM table (  
  DBMS_XPLAN.DISPLAY_SQLSET(  
    'OLTP_optimization_0405', 'gwp663cqh5qbf', 3693697075));
```

To display all execution plans of the SQL ID `'atfwcg8anrykp'` stored in the SQL tuning set:

```
SELECT * FROM table (  
  DBMS_XPLAN.DISPLAY_SQLSET(  
    'OLTP_optimization_0405', 'gwp663cqh5qbf'));
```

To display runtime statistics for the SQL statement included in the preceding statement:

```
SELECT * FROM table (  
  DBMS_XPLAN.DISPLAY_SQLSET(  
    'OLTP_optimization_0405', 'gwp663cqh5qbf', NULL, 'ALLSTATS LAST');
```

---

---

## DBMS\_XSLPROCESSOR

The DBMS\_XSLPROCESSOR package provides an interface to manage the contents and structure of XML documents.

This chapter contains the following topics:

- [Using DBMS\\_XSLPROCESSOR](#)
  - Overview
  - Security Model
- [Summary of DBMS\\_XSLPROCESSOR Subprograms](#)

**See Also:**

- *Oracle XML DB Developer's Guide*

---

## Using DBMS\_XSLPROCESSOR

This section contains topics which relate to using the DBMS\_XSLPROCESSOR package.

- [Overview](#)
- [Security Model](#)

## Overview

The DBMS\_XSLPROCESSOR package provides an interface to manage the contents and structure of XML documents.

### Standards

This PL/SQL implementation of the XSL processor follows the W3C XSLT working draft rev WD-xslt-19990813 and includes the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformation it must effect.

### Concepts

The Extensible Stylesheet Language Transformation (XSLT) describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet. The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree.

### Implementation

The following is the default behavior for this PL/SQL XSL Processor:

- A result tree which can be accessed by DOM programmatic interface
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

## Security Model

Owned by XDB, the DBMS\_XSLPROCESSOR package must be created by SYS or XDB. The EXECUTE privilege is granted to PUBLIC. Subprograms in this package are executed using the privileges of the current user.

---

## Summary of DBMS\_XSLPROCESSOR Subprograms

**Table 180–1 DBMS\_XSLPROCESSOR Package Subprograms**

Method	Description
<a href="#">CLOB2FILE Procedure</a> on page 180-6	Writes content of a CLOB into a file
<a href="#">FREEPROCESSOR Procedure</a> on page 180-7	Frees a processor object
<a href="#">FREESTYLESHEET Procedure</a> on page 180-8	Frees a stylesheet object
<a href="#">NEWPROCESSOR Function</a> on page 180-8	Returns a new processor instance
<a href="#">NEWSTYLESHEET Functions</a> on page 180-10	Creates a new stylesheet from input and reference URLs
<a href="#">PROCESSXSL Functions and Procedures</a> on page 180-11	Transforms an input XML document
<a href="#">READ2CLOB Function</a> on page 180-13	Reads content of the file into a CLOB
<a href="#">REMOVEPARAM Procedure</a> on page 180-14	Removes a top-level stylesheet parameter
<a href="#">RESETPARAMS Procedure</a> on page 180-15	Resets the top-level stylesheet parameters
<a href="#">SELECTNODES Function</a> on page 180-16	Selects nodes from a DOM tree that match a pattern
<a href="#">SELECTSINGLENODE Function</a> on page 180-17	Selects the first node from the tree that matches a pattern
<a href="#">SETERRORLOG Procedure</a> on page 180-18	Sets errors to be sent to the specified file
<a href="#">SETPARAM Procedure</a> on page 180-19	Sets a top-level parameter in the stylesheet
<a href="#">SHOWWARNINGS Procedure</a> on page 180-20	Turns warnings on or off
<a href="#">TRANSFORMNODE Function</a> on page 180-21	Transforms a node in a DOM tree using a stylesheet
<a href="#">VALUEOF Function and Procedure</a> on page 180-22	Gets the value of the first node that matches a pattern

---

## CLOB2FILE Procedure

This procedure writes content of a CLOB into a file.

### Syntax

```
DBMS_XSLPROCESSOR.CLOB2FILE(  
  cl          IN  CLOB;  
  flocation  IN  VARCHAR2,  
  fname      IN  VARCHAR2,  
  csid       IN  NUMBER:=0);
```

### Parameters

**Table 180–2 CLOB2FILE Procedure Parameters**

Parameter	Description
CLOB	File directory
flocation	File directory
fname	File name
csid	Character set id of the file <ul style="list-style-type: none"><li>■ Must be a valid Oracle id; otherwise returns an error</li><li>■ If 0, content of the output file will be in the database character set</li></ul>



## **FREEPROCESSOR Procedure**

This procedure Frees a Processor object.

### **Syntax**

```
DBMS_XSLPROCESSOR.FREEPROCESSOR(  
  p IN Processor);
```

### **Parameters**

**Table 180–3** *FREEPROCESSOR Procedure Parameters*

<b>Parameter</b>	<b>Description</b>
p	Processor

## FREESTYLESHEET Procedure

This procedure frees a `Stylesheet` object.

### Syntax

```
DBMS_XSLPROCESSOR.FREESTYLESHEET(  
    ss IN Stylesheet);
```

### Parameters

**Table 180–4** *FREESTYLESHEET Procedure Parameters*

Parameter	Description
ss	Stylesheet

## NEWPROCESSOR Function

This function returns a new `Processor` instance. The function must be called before the default behavior of `Processor` can be changed and if other processor methods need to be used.

### Syntax

```
DBMS_XSLPROCESSOR.NEWPROCESSOR  
RETURN Processor;
```

## NEWSTYLESHEET Functions

This function creates and returns a new `Stylesheet` instance. The options are described in the following table.

### Syntax

Creates and returns a new `stylesheet` instance using the given `DOMDOCUMENT` and reference URLs:

```
DBMS_XSLPROCESSOR.NEWSTYLESHEET(  
    xmlDoc IN DOMDOCUMENT,  
    ref    IN VARCHAR2)  
RETURN Stylesheet;
```

Creates and returns a new `Stylesheet` instance using the given input and reference URLs:

```
DBMS_XSLPROCESSOR.NEWSTYLESHEET(  
    inp    IN VARCHAR2,  
    ref    IN VARCHAR2)  
RETURN Stylesheet;
```

### Parameters

**Table 180–5** *NEWSTYLESHEET Function Parameters*

Parameter	Description
<code>xmlDoc</code>	DOMDocument to use for construction
<code>inp</code>	Input URL to use for construction
<code>ref</code>	Reference URL

## PROCESSXSL Functions and Procedures

This function transforms input `XMLDocument`. Any changes to the default processor behavior should be effected before calling this procedure. An application error is raised if processing fails.

### Syntax

Transforms input `XMLDocument` using given `DOMDocument` and `stylesheet`, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL (
  p      IN  Processor,
  ss     IN  Stylesheet,
  xmldoc IN  DOMDOCUMENT),
RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input `XMLDocument` using given document as URL and the `Stylesheet`, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL (
  p      IN  Processor,
  ss     IN  Stylesheet,
  url    IN  VARCHAR2,
RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input `XMLDocument` using given document as CLOB and the `Stylesheet`, and returns the resultant document fragment:

```
DBMS_XSLPROCESSOR.PROCESSXSL (
  p      IN  Processor,
  ss     IN  Stylesheet,
  clb    IN  CLOB)
RETURN DOMDOCUMENTFRAGMENT;
```

Transforms input `XMLDocument` using given `DOMDOCUMENT` and the `stylesheet`, and writes the output to the specified file:

```
DBMS_XSLPROCESSOR.DBMS_XSLPROCESSOR. (
  p          IN  Processor,
  ss         IN  Stylesheet,
  xmldoc     IN  DOMDOCUMENT,
  dir        IN  VARCHAR2,
  fileName   IN  VARCHAR2);
```

Transforms input `XMLDocument` using given URL and the `stylesheet`, and writes the output to the specified file in a specified directory:

```
DBMS_XSLPROCESSOR.PROCESSXSL (
  p          IN  Processor,
  ss         IN  Stylesheet,
  url        IN  VARCHAR2,
  dir        IN  VARCHAR2,
  fileName   IN  VARCHAR2);
```

Transforms input `XMLDocument` using given `DOMDOCUMENT` and the `stylesheet`, and writes the output to a CLOB:

```
DBMS_XSLPROCESSOR.PROCESSXSL (
  p          IN  Processor,
```

```

ss          IN      Stylesheet,
xmldoc     IN      DOMDOCUMENT,
cl         IN OUT  CLOB);

```

Transforms input `XMLDocument` using given `DOMDOCUMENTFRAGMENT` and the stylesheet, and returns the resultant document fragment:

```

DBMS_XSLPROCESSOR.PROCESSXSL(
  p          IN      Processor,
  ss         IN      Stylesheet,
  xmldf     IN      DOMDOCUMENTFRAGMENT)
RETURN DOMDOCUMENTFRAGMENT;

```

Transforms input `XMLDocumentFragment` using given `DOMDocumentFragment` and the stylesheet, and writes the output to the specified file in a specified directory:

```

DBMS_XSLPROCESSOR.PROCESSXSL(
  p          IN      Processor,
  ss         IN      Stylesheet,
  xmldf     IN      DOMDOCUMENTFRAGMENT,
  dir       IN      VARCHAR2,
  filename  IN      VARCHAR2);

```

Transforms input `XMLDocumentFragment` using given `DOMDOCUMENTFRAGMENT` and the stylesheet, and writes the output to a buffer:

```

DBMS_XSLPROCESSOR.PROCESSXSL(
  p          IN      Processor,
  ss         IN      Stylesheet,
  xmldf     IN      DOMDOCUMENTFRAGMENT,
  buf       IN OUT  VARCHAR2);

```

Transforms input `XMLDocumentFragment` using given `DOMDOCUMENTFRAGMENT` and the stylesheet, and writes the output to a CLOB:

```

DBMS_XSLPROCESSOR.PROCESSXSL(
  p          IN      Processor,
  ss         IN      Stylesheet,
  xmldf     IN      DOMDOCUMENTFRAGMENT,
  cl        IN OUT  CLOB);

```

## Parameters

**Table 180–6** *PROCESSXSL Function and Procedure Parameters*

Parameter	Description
p	Processor instance
ss	Stylesheet instance
xmldoc	XML document being transformed
url	URL for the information being transformed
clb	CLOB containing information to be transformed
dir	Directory where processing output file is saved
filename	Processing output file
cl	CLOB to which the processing output is saved
xmldf	<code>XMLDocumentFragment</code> being transformed

## READ2CLOB Function

This function reads content of a file into a CLOB.

### Syntax

```
DBMS_XSLPROCESSOR.READ2CLOB(  
    flocation    IN    VARCHAR2,  
    fname        IN    VARCHAR2,  
    csid         IN    NUMBER:=0)  
RETURN CLOB;
```

### Parameters

**Table 180–7** *READ2CLOB Function Parameters*

Parameter	Description
flocation	File directory
fname	File name
csid	Character set id of the file <ul style="list-style-type: none"><li>■ Must be a valid Oracle id; otherwise returns an error</li><li>■ If 0, input file is assumed to be in the database character set</li></ul>

## REMOVEPARAM Procedure

This procedure removes a top level stylesheet parameter.

### Syntax

```
DBMS_XSLPROCESSOR.REMOVEPARAM(  
    ss      IN  Stylesheet,  
    name    IN  VARCHAR2);
```

### Parameters

**Table 180–8 REMOVEPARAM Procedure Parameters**

Parameter	Description
ss	Stylesheet instance
name	Name of the parameter



## RESETPARAMS Procedure

This procedure resets the top-level stylesheet parameters.

### Syntax

```
DBMS_XSLPROCESSOR.RESETPARAMS(  
    ss IN    Stylesheet);
```

### Parameters

**Table 180–9** *RESETPARAMS Procedure Parameters*

Parameter	Description
ss	Stylesheet instance

## SELECTNODES Function

This function selects nodes which match the supplied path expression from a DOM tree, and returns the result of the selection.

### Syntax

```
DBMS_XSLPROCESSOR.SELECTNODES(  
  n          IN  DBMS_XMLDOM.DOMNODE,  
  pattern    IN  VARCHAR2,  
  namespace  IN  VARCHAR2 := NULL)  
RETURN DBMS_XMLDOM.DOMNODELIST;
```

### Parameters

**Table 180–10** *SELECTNODES Function Parameters*

Parameter	Description
n	Root DOMNode of the tree
pattern	Pattern to use
namespace	Namespace declared

## SELECTSINGLENODE Function

This function selects the first node from the tree that match the supplied path expression, and returns that node.

### Syntax

```
DBMS_XSLPROCESSOR.SELECTSINGLENODE(  
  n          IN  DBMS_XMLDOM.DOMNODE,  
  pattern    IN  VARCHAR2,  
  namespace  IN  VARCHAR2 := NULL)  
RETURN DBMS_XMLDOM.DOMNODE;
```

### Parameters

**Table 180–11** *SELECTSINGLENODE Function Parameters*

Parameter	Description
n	Root DOMNode of the tree
pattern	Pattern to use
namespace	Namespace declared

## SETERRORLOG Procedure

This procedure sets errors to be sent to the specified file.

---

---

**Note:** This subprogram has been deprecated, and is included only for reasons of backward compatibility.

---

---

### Syntax

```
DBMS_XSLPROCESSOR.SETERRORLOG(  
  p          IN  Processor,  
  fileName  IN  VARCHAR2);
```

### Parameters

**Table 180–12** *SETERRORLOG Procedure Parameters*

Parameter	Description
p	Processor instance
fileName	Complete path of the file to use as the error log

## SETPARAM Procedure

This procedure sets a top level parameter in the stylesheet. The parameter value must be a valid XPath expression. Literal string values must be quoted.

### Syntax

```
DBMS_XSLPROCESSOR.SETPARAM(  
  ss      IN  Stylesheet,  
  name    IN  VARCHAR2,  
  value   IN  VARCHAR2);
```

### Parameters

**Table 180–13** *SETPARAM Procedure Parameters*

Parameter	Description
ss	Stylesheet instance
name	Name of the parameter
value	Value of the parameter

## SHOWWARNINGS Procedure

This procedure turns warnings on (TRUE) or off (FALSE).

### Syntax

```
DBMS_XSLPROCESSOR.SHOWWARNINGS (  
  p      IN  Processor,  
  yes   IN  BOOLEAN);
```

### Parameters

**Table 180–14** *SHOWWARNINGS Procedure Parameters*

Parameter	Description
p	Processor instance
yes	Mode to set: TRUE to show warnings, FALSE otherwise

## TRANSFORMNODE Function

This function transforms a node in a DOM tree using the given stylesheet, and returns the result of the transformation as a `DOMDocumentFragment`.

### Syntax

```
DBMS_XSLPROCESSOR.TRANSFORMNODE(  
  n    IN DOMNODE,  
  ss   IN Stylesheet)  
RETURN DOMDocumentFragment;
```

### Parameters

**Table 180–15** TRANSFORMNODE Function Parameters

Parameter	Description
n	DOMNode to transform
ss	Stylesheet to use

## VALUEOF Function and Procedure

This subprogram retrieves the value of the first node from the tree that matches the given pattern. You can use either a function or a procedure.

### Syntax

```
DBMS_XSLPROCESSOR.VALUEOF (  
  n          IN      DBMS_XMLDOM.DOMNODE,  
  pattern    IN      VARCHAR2,  
  namespace  IN      VARCHAR2 := NULL)  
RETURN VARCHAR2;
```

```
DBMS_XSLPROCESSOR.VALUEOF (  
  n          IN      DBMS_XMLDOM.DOMNODE,  
  pattern    IN      VARCHAR2,  
  val        OUT     VARCHAR2,  
  namespace  IN      VARCHAR2 := NULL);
```

### Parameters

**Table 180–16** VALUEOF Function and Procedure Parameters

Parameter	Description
n	Node whose value is being retrieved
pattern	Pattern to use
val	Retrieved value
namespace	Namespace to use



The `DEBUG_EXTPROC` package enables you to start up the `extproc` agent within a session. This utility package can help you debug external procedures.

This chapter contains the following topics:

- [Using `DEBUG\_EXTPROC`](#)
  - Security Model
  - Operational Notes
  - Rules and Limits
- [Summary of `DEBUG\_EXTPROC` Subprograms](#)

## Using DEBUG\_EXTPROC

- [Security Model](#)
- [Operational Notes](#)
- [Rules and Limits](#)

## Security Model

Your Oracle account must have `EXECUTE` privileges on the package and `CREATE LIBRARY` privileges.

## Operational Notes

To install the package, run the script `DBGEXTP.SQL`.

- Install/load this package in the Oracle USER where you want to debug the 'extproc' process.

- Ensure that you have execute privileges on package `DEBUG_EXTPROC`

```
SELECT SUBSTR(OBJECT_NAME, 1, 20)
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'DEBUG_EXTPROC';
```

- You can install this package as any other user, as long as you have EXECUTE privileges on the package.

---

---

**Note:** These notes assumes that you built your shared library with debug symbols to aid in the debugging process. Please check the C compiler manual pages for the appropriate C compiler switches to build the shared library with debug symbols.

---

---

Having installed the package, proceed accordingly:

- Start a new Oracle session through SQL\*Plus or OCI program by connecting to ORACLE.
- Execute procedure `DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT` to startup the extproc agent in this session; for example, execute `DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT`; Do not exit this session, because that terminates the extproc agent.
- Determine the PID of the extproc agent that was started up for this session.
- Using a debugger (for example, gdb, dbx, or the native system debugger), load the extproc executable and attach to the running process.
- Set a breakpoint on function 'pextproc' and let the debugger continue with its execution.
- Now execute your external procedure in the same session where you first executed `DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT`
- Your debugger should now break in function 'pextproc'. At this point in time, the shared library referenced by your PL/SQL external function would have been loaded and the function resolved. Now set a breakpoint in your C function and let the debugger continue its execution.

Because PL/SQL loads the shared library at runtime, the debugger you use may or may not automatically be able to track the new symbols from the shared library. You may have to issue some debugger command to load the symbols (for example, 'share' in gdb)

- The debugger should now break in your C function. Its assumed that you had built the shared library with debugging symbols.
- Now proceed with your debugging.

## Rules and Limits

---

---

**Note:** DEBUG\_EXTPROC works only on platforms with debuggers that can attach to a running process.

---

---

## Summary of DEBUG\_EXTPROC Subprograms

**Table 181-1** *DEBUG\_EXTPROC Package Subprograms*

Subprogram	Description
<a href="#">STARTUP_EXTPROC_AGENT Procedure</a> on page 181-7	Starts up the extproc agent process in the session

## STARTUP\_EXTPROC\_AGENT Procedure

This procedure starts up the extproc agent process in the session. This enables you to get the PID of the executing process. This PID is needed to be able to attach to the running process using a debugger.

### Syntax

```
DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT;
```





The HTF (hypertext functions) and HTP (hypertext procedures) packages generate HTML tags. For example, the HTF.ANCHOR function generates the HTML anchor tag, <A>.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

This chapter contains the following topics:

- [Using HTF](#)
  - Deprecated Subprograms
  - Operational Notes
  - Rules and Limits
  - Examples
- [Summary of Tags](#)
- [Summary of HTF Subprograms](#)

## Using HTF

- [Deprecated Subprograms](#)
- [Operational Notes](#)
- [Rules and Limits](#)
- [Examples](#)

## Deprecated Subprograms

---

---

**Note:** Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

---

---

The following subprogram is deprecated with Oracle Database 10g:

- [ESCAPE\\_URL Function](#)

## Operational Notes

For every HTF function that generates one or more HTML tags, there is a corresponding HTP procedure with identical parameters with the following exception:

- The [PRINTS Procedure](#) and the [PS Procedure](#) do not have HTF function equivalents. Use the [ESCAPE\\_SC Function](#) or the [ESCAPE\\_URL Function](#) if you need a string conversion function. Note that while there is a [ESCAPE\\_SC Procedure](#) that performs the same operation as the [PRINTS Procedure](#) and the [PS Procedure](#), there is no procedural equivalent for the [ESCAPE\\_URL Function](#).
- The [FORMAT\\_CELL Function](#) does not have an HTP equivalent. The function formats column values inside an HTML table using [TABLEDATA Function](#) which does have an HTP equivalent in the [TABLEDATA Procedure](#). The advantage of this using the [FORMAT\\_CELL Function](#) is that it allows for better control over the HTML tables.

The function versions do not directly generate output in your Web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls. To print the output of HTF functions, call the functions from within the `HTF.PRINT` function. It then prints its parameters to the generated Web page.

## Rules and Limits

If you use values of the LONG data type in functions such as HTF . PRINT, HTF . PRN, HTF . PA or OWA\_UTIL.CELLSPRINT, only the first 32 K of the LONG data is used. The LONG data is bound to a VARCHAR2 data type in the function.

## Examples

The following commands generate a simple HTML document:

```
CREATE OR REPLACE PROCEDURE hello AS
BEGIN
    HTP.P (HTF.HTMLOPEN); -- generates <HTML>
    HTP.P (HTF.HEADOPEN); -- generates <HEAD>
    HTP.P (HTF.TITLE('Hello')); -- generates <TITLE>Hello</TITLE>
    HTP.P (HTF.HEADCLOSE); -- generates </HEAD>
    HTP.P (HTF.BODYOPEN); -- generates <BODY>
    HTP.P (HTF.HEADER(1, 'Hello')); -- generates <H1>Hello</H1>
    HTP.P (HTF.BODYCLOSE); -- generates </BODY>
    HTP.P (HTF.HTMLCLOSE); -- generates </HTML>
END;
```

---

## Summary of Tags

### HTML, HEAD, and BODY Tags

[HTMLOPEN Function](#), [HTMLCLOSE Function](#) - generate <HTML> and </HTML>

[HEADOPEN Function](#), [HEADCLOSE Function](#) - generate <HEAD> and </HEAD>

[BODYOPEN Function](#), [BODYCLOSE Function](#) - generate <BODY> and </BODY>

### Comment Tag

[COMMENT Function](#) - generates <!-- and -->

[http://www.w3.org.BASE Function](#) - generates <BASE>

[LINKREL Function](#) - generates <LINK> with the REL attribute

[LINKREV Function](#) - generates <LINK> with the REV attribute

[TITLE Function](#) - generates <TITLE>

[META Function](#) - generates <META>

[SCRIPT Function](#) - generates <SCRIPT>

[STYLE Function](#) - generates <STYLE>

[ISINDEX Function](#) - generates <ISINDEX>

### Applet Tags

[APPLETOPEN Function](#), [APPLETCLOSE Function](#) - generate <APPLET> and </APPLET>

[PARAM Function](#) - generates <PARAM>

### List Tags

[OLISTOPEN Function](#), [OLISTCLOSE Function](#) - generate <OL> and </OL>

[ULISTOPEN Function](#), [ULISTCLOSE Function](#) - generate <UL> and </UL>

[DLISTOPEN Function](#), [DLISTCLOSE Function](#) - generate <DL> and </DL>

[DLISTTERM Function](#) - generates <DT>

[DLISTDEF Function](#) - generates <DD>

[DIRLISTOPEN Function](#), [DIRLISTCLOSE Function](#) - generate <DIR> and </DIR>

[LISTHEADER Function](#) - generates <LH>

[LISTINGOPEN Function](#), [LISTINGCLOSE Function](#) - generate <LISTING> and </LISTING>

[MENULISTOPEN Function](#) - generate <MENU> and </MENU>

[LISTITEM Function](#) - generates <LI>

### Form Tags

[FORMOPEN Function](#), [FORMCLOSE Function](#) - generate <FORM> and </FORM>

[FORMCHECKBOX Function](#) - generates <INPUT TYPE="CHECKBOX">

[FORMHIDDEN Function](#) - generates <INPUT TYPE="HIDDEN">

[FORMIMAGE Function](#) - generates `<INPUT TYPE="IMAGE">`  
[FORMPASSWORD Function](#) - generates `<INPUT TYPE="PASSWORD">`  
[FORMRADIO Function](#) - generates `<INPUT TYPE="RADIO">`  
[FORMSELECTOPEN Function](#), [FORMSELECTCLOSE Function](#) - generate `<SELECT>`  
and `</SELECT>`  
[FORMSELETOPTION Function](#) - generates `<OPTION>`  
[FORMTEXT Function](#) - generates `<INPUT TYPE="TEXT">`  
[FORMTEXTAREA Function](#) - generate `<TEXTAREA>`  
[FORMTEXTAREAOPEN Function](#), [FORMTEXTAREACLOSE Function](#) - generate  
`<TEXTAREA>` and `</TEXTAREA>`  
[FORMRESET Function](#) - generates `<INPUT TYPE="RESET">`  
[FORMSUBMIT Function](#) - generates `<INPUT TYPE="SUBMIT">`

## Table Tags

[TABLEOPEN Function](#), [TABLECLOSE Function](#) - generate `<TABLE>` and `</TABLE>`  
[TABLECAPTION Function](#) - generates `<CAPTION>`  
[TABLEROWOPEN Function](#), [TABLEROWCLOSE Function](#) - generate `<TR>` and  
`</TR>`  
[TABLEHEADER Function](#) - generates `<TH>`  
[TABLEDATA Function](#) - generates `<TD>`

## IMG, HR, and A Tags

[HR Function](#), [LINE Function](#) - generate `<HR>`  
[IMG Function](#), [IMG2 Function](#) - generate `<IMG>`  
[ANCHOR Function](#), [ANCHOR2 Function](#) - generate `<A>`  
[MAPOPEN Function](#), [MAPCLOSE Function](#) - generate `<MAP>` and `</MAP>`

## Paragraph Formatting Tags

[HEADER Function](#) - generates heading tags (`<H1>` to `<H6>`)  
[PARA Function](#), [PARAGRAPH Function](#) - generate `<P>`  
[PRN Functions](#), [PRINT Functions](#) - generate any text that is passed in  
[PRN Functions](#), [S Function](#) - generate any text that is passed in; special characters in  
HTML are escaped  
[PREOPEN Function](#), [PRECLOSE Function](#) - generate `<PRE>` and `</PRE>`  
[BLOCKQUOTEOPEN Function](#), [BLOCKQUOTECLOSE Function](#) - generate  
`<BLOCKQUOTE>` and `</BLOCKQUOTE>`  
[DIV Function](#) - generates `<DIV>`  
[NL Function](#), [BR Function](#) - generate `<BR>`  
[NOBR Function](#) - generates `<NOBR>`  
[WBR Function](#) - generates `<WBR>`



[PLAINTEXT Function](#) - generates <PLAINTEXT>

[ADDRESS Function](#) - generates <ADDRESS>

[MAILTO Function](#) - generates <A> with the MAILTO attribute

[AREA Function](#) - generates <AREA>

[BGSOUND Function](#) - generates <BGSOUND>

## Character Formatting Tags

[BASEFONT Function](#) - generates <BASEFONT>

[BIG Function](#) - generates <BIG>

[BOLD Function](#) - generates <B>

[CENTER Function](#) - generates <CENTER> and </CENTER>

[CENTEROPEN Function](#), [CENTERCLOSE Function](#) - generate <CENTER> and </CENTER>

[CITE Function](#) - generates <CITE>

[CODE Function](#) - generates <CODE>

[DFN Function](#) - generates <DFN>

[EM Function](#), [EMPHASIS Function](#) - generate <EM>

[FONTOPEN Function](#), [FONTCLOSE Function](#) - generate <FONT> and </FONT>

[ITALIC Function](#) - generates <I>

[KBD Function](#), [KEYBOARD Function](#) - generate <KBD> and </KBD>

[S Function](#) - generates <S>

[SAMPLE Function](#) - generates <SAMP>

[SMALL Function](#) - generates <SMALL>

[STRIKE Function](#) - generates <STRIKE>

[STRONG Function](#) - generates <STRONG>

[SUB Function](#) - generates <SUB>

[SUP Function](#) - generates <SUP>

[TELETYPE Function](#) - generates <TT>

[UNDERLINE Function](#) - generates <U>

[VARIABLE Function](#) - generates <VAR>

## Frame Tags

[FRAME Function](#) - generates <FRAME>

[FRAMESETOPEN Function](#), [FRAMESETCLOSE Function](#) - generate <FRAMESET> and </FRAMESET>

[NOFRAMESOPEN Function](#), [NOFRAMESCLOSE Function](#) - generate <NOFRAMES> and </NOFRAMES>

---

## Summary of HTF Subprograms

**Table 182–1 HTF Package Subprograms**

Subprogram	Description
<a href="#">ADDRESS Function</a> on page 182-16	Generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document
<a href="#">ANCHOR Function</a> on page 182-17	Generates the <A> and </A> tags which specify the source or destination of a hypertext link
<a href="#">ANCHOR2 Function</a> on page 182-18	Generates the <A> and </A> tags which specify the source or destination of a hypertext link
<a href="#">APPLETCLOSE Function</a> on page 182-19	Closes the applet invocation with the </APPLET> tag
<a href="#">APPLETOPEN Function</a> on page 182-20	Generates the <APPLET> tag which begins the invocation of a Java applet
<a href="#">AREA Function</a> on page 182-21	Generates the <AREA> tag, which defines a client-side image map
<a href="#">BASE Function</a> on page 182-22	Generates the <BASE> tag which records the URL of the document
<a href="#">BASEFONT Function</a> on page 182-23	Generates the <BASEFONT> tag which specifies the base font size for a Web page
<a href="#">BGSOUND Function</a> on page 182-24	Generates the <BGSOUND> tag which includes audio for a Web page
<a href="#">BIG Function</a> on page 182-25	Generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font
<a href="#">BLOCKQUOTECLOSE Function</a> on page 182-26	Generates the </BLOCKQUOTE> tag which mark the end of a section of quoted text
<a href="#">BLOCKQUOTEOPEN Function</a> on page 182-27	Generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text
<a href="#">BODYCLOSE Function</a> on page 182-28	Generates the </BODY> tag which marks the end of a body section of an HTML document
<a href="#">BODYOPEN Function</a> on page 182-29	Generates the <BODY> tag which marks the beginning of the body section of an HTML document
<a href="#">BOLD Function</a> on page 182-30	Generates the <B> and </B> tags which direct the browser to display the text in boldface
<a href="#">BR Function</a> on page 182-31	Generates the   tag which begins a new line of text
<a href="#">CENTER Function</a> on page 182-32	Generates the <CENTER> and </CENTER> tags which center a section of text within a Web page
<a href="#">CENTERCLOSE Function</a> on page 182-33	Generates the </CENTER> tag which marks the end of a section of text to center
<a href="#">CENTEROPEN Function</a> on page 182-34	Generates the <CENTER> tag which mark the beginning of a section of text to center
<a href="#">CITE Function</a> on page 182-35	Generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation
<a href="#">CODE Function</a> on page 182-36	Generates the <CODE> and </CODE> tags which direct the browser to render the text in monospace font or however "code" is defined stylistically

**Table 182-1 (Cont.) HTF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">COMMENT Function</a> on page 182-37	Generates the comment tags <code>&lt;!-- ctext --&gt;</code>
<a href="#">DFN Function</a> on page 182-38	Generates the <code>&lt;DFN&gt;</code> and <code>&lt;/DFN&gt;</code> tags which direct the browser to mark the text as italics or however "definition" is defined stylistically
<a href="#">DIRLISTCLOSE Function</a> on page 182-39	Generates the <code>&lt;/DIR&gt;</code> tag which ends a directory list section
<a href="#">DIRLISTOPEN Function</a> on page 182-40	Generates the <code>&lt;DIR&gt;</code> which starts a directory list section
<a href="#">DIV Function</a> on page 182-41	Generates the <code>&lt;DIV&gt;</code> tag which creates document divisions
<a href="#">DLISTCLOSE Function</a> on page 182-42	Generates the <code>&lt;/DL&gt;</code> tag which ends a definition list
<a href="#">DLISTDEF Function</a> on page 182-43	Generates the <code>&lt;DD&gt;</code> tag, which inserts definitions of terms
<a href="#">DLISTOPEN Function</a> on page 182-44	Generates the <code>&lt;DL&gt;</code> tag which starts a definition list
<a href="#">DLISTTERM Function</a> on page 182-45	Generates the <code>&lt;DT&gt;</code> tag which defines a term in a definition list <code>&lt;DL&gt;</code>
<a href="#">EM Function</a> on page 182-46	Generates the <code>&lt;EM&gt;</code> and <code>&lt;/EM&gt;</code> tags, which define text to be emphasized
<a href="#">EMPHASIS Function</a> on page 182-47	Generates the <code>&lt;EM&gt;</code> and <code>&lt;/EM&gt;</code> tags, which define text to be emphasized
<a href="#">ESCAPE_SC Function</a> on page 182-48	Replaces characters that have special meaning in HTML with their escape sequences
<a href="#">ESCAPE_URL Function</a> on page 182-49	Replaces characters that have special meaning in HTML and HTTP with their escape sequences
<a href="#">FONTCLOSE Function</a> on page 182-51	Generates the <code>&lt;/FONT&gt;</code> tag which marks the end of a section of text with the specified font characteristics
<a href="#">FONTOPEN Function</a> on page 182-51	Generates the <code>&lt;FONT&gt;</code> which marks the beginning of section of text with the specified font characteristics
<a href="#">FORMAT_CELL Function</a> on page 182-52	formats column values inside an HTML table using the <a href="#">TABLEDATA Function</a>
<a href="#">FORMCHECKBOX Function</a> on page 182-53	Generates the <code>&lt;INPUT&gt;</code> tag with <code>TYPE="checkbox"</code> which inserts a checkbox element in a form
<a href="#">FORMCLOSE Function</a> on page 182-54	Generates the <code>&lt;/FORM&gt;</code> tag which marks the end of a form section in an HTML document
<a href="#">FORMFILE Function</a> on page 182-55	Generates the <code>&lt;INPUT&gt;</code> tag with <code>TYPE="file"</code> which inserts a file form element, and is used for file uploading for a given page
<a href="#">FORMHIDDEN Function</a> on page 182-56	Generates the <code>&lt;INPUT&gt;</code> tag with <code>TYPE="hidden"</code> which inserts a hidden form element
<a href="#">FORMIMAGE Function</a> on page 182-57	Generates the <code>&lt;INPUT&gt;</code> tag with <code>TYPE="image"</code> which creates an image field that the user clicks to submit the form immediately
<a href="#">FORMOPEN Function</a> on page 182-58	Generates the <code>&lt;FORM&gt;</code> tag which marks the beginning of a form section in an HTML document

**Table 182-1 (Cont.) HTF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">FORMPASSWORD Function</a> on page 182-59	Generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field
<a href="#">FORMRADIO Function</a> on page 182-60	Generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form
<a href="#">FORMRESET Function</a> on page 182-61	Generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values
<a href="#">FORMSELECTCLOSE Function</a> on page 182-62	Generates the </SELECT> tag which marks the end of a Select form element
<a href="#">FORMSELECTOPEN Function</a> on page 182-63	Generates the </SELECT> tag which marks the beginning of a Select form element
<a href="#">FORMSELECTOPTION Function</a> on page 182-64	Generates the <OPTION> tag which represents one choice in a Select element
<a href="#">FORMSUBMIT Function</a> on page 182-65	Generates the <INPUT> tag with TYPE="submit" which creates a button that, when clicked, submits the form
<a href="#">FORMTEXT Function</a> on page 182-66	Generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text
<a href="#">FORMTEXTAREA Function</a> on page 182-67	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area
<a href="#">FORMTEXTAREA2 Function</a> on page 182-68	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area with the ability to specify a wrap style
<a href="#">FORMTEXTAREACLOSE Function</a> on page 182-69	Generates the </TEXTAREA> tag which ends a text area form element
<a href="#">FORMTEXTAREAOPEN Function</a> on page 182-70	Generates the <TEXTAREA> which marks the beginning of a text area form element
<a href="#">FORMTEXTAREAOPEN2 Function</a> on page 182-71	Generates the <TEXTAREA> which marks the beginning of a text area form element with the ability to specify a wrap style
<a href="#">FRAME Function</a> on page 182-72	Generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag
<a href="#">FRAMESETCLOSE Function</a> on page 182-73	Generates the </FRAMESET> tag which ends a frameset section
<a href="#">FRAMESETOPEN Function</a> on page 182-74	Generates the </FRAMESET> tag which begins a frameset section
<a href="#">HEADCLOSE Function</a> on page 182-75	Generates the </HEAD> tag which marks the end of an HTML document head section
<a href="#">HEADER Function</a> on page 182-76	Generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>)
<a href="#">HEADOPEN Function</a> on page 182-77	Generates the <HEAD> tag which marks the beginning of the HTML document head section
<a href="#">HR Function</a> on page 182-78	Generates the <HR> tag, which generates a line in the HTML document
<a href="#">HTMLCLOSE Function</a> on page 182-80	Generates the </HTML> tag which marks the end of an HTML document

**Table 182-1 (Cont.) HTF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">HTMLOPEN Function</a> on page 182-80	Generates the <HTML> tag which marks the beginning of an HTML document
<a href="#">IMG Function</a> on page 182-81	Generates the <IMG> tag which directs the browser to load an image onto the HTML page
<a href="#">IMG2 Function</a> on page 182-82	Generates the <IMG> tag which directs the browser to load an image onto the HTML page with the option of specifying values for the USEMAP attribute
<a href="#">ISINDEX Function</a> on page 182-83	Creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program
<a href="#">ITALIC Function</a> on page 182-84	Generates the <I> and </I> tags which direct the browser to render the text in italics
<a href="#">KBD Function</a> on page 182-85	Generates the <KBD> and </KBD> tags which direct the browser to render the text in monospace font
<a href="#">KEYBOARD Function</a> on page 182-86	Generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font
<a href="#">LINE Function</a> on page 182-87	Generates the <HR> tag, which generates a line in the HTML document
<a href="#">LINKREL Function</a> on page 182-88	Generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target
<a href="#">LINKREV Function</a> on page 182-89	Generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor
<a href="#">LISTHEADER Function</a> on page 182-90	Generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list
<a href="#">LISTINGCLOSE Function</a> on page 182-91	Generates the </LISTING> tags which marks the end of a section of fixed-width text in the body of an HTML page
<a href="#">LISTINGOPEN Function</a> on page 182-92	Generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page
<a href="#">LISTITEM Function</a> on page 182-93	Generates the <LI> tag, which indicates a list item
<a href="#">MAILTO Function</a> on page 182-94	Generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument
<a href="#">MAPCLOSE Function</a> on page 182-95	Generates the </MAP> tag which marks the end of a set of regions in a client-side image map
<a href="#">MAPOPEN Function</a> on page 182-96	Generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map
<a href="#">MENULISTCLOSE Function</a> on page 182-97	Generates the </MENU> tag which ends a list that presents one line for each item
<a href="#">MENULISTOPEN Function</a> on page 182-98	Generates the <MENU> tag which create a list that presents one line for each item
<a href="#">META Function</a> on page 182-99	Generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers

**Table 182–1 (Cont.) HTF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">NL Function</a> on page 182-87	Generates the   tag which begins a new line of text
<a href="#">NOBR Function</a> on page 182-101	Generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text
<a href="#">NOFRAMESCLOSE Function</a> on page 182-102	Generates the </NOFRAMES> tag which marks the end of a no-frames section
<a href="#">NOFRAMESOPEN Function</a> on page 182-103	Generates the <NOFRAMES> tag which mark the beginning of a no-frames section
<a href="#">OLISTCLOSE Function</a> on page 182-104	Generates the </OL> tag which defines the end of an ordered list
<a href="#">OLISTOPEN Function</a> on page 182-105	Generates the <OL> tag which marks the beginning of an ordered list
<a href="#">PARA Function</a> on page 182-106	Generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph
<a href="#">PARAGRAPH Function</a> on page 182-107	Adds attributes to the <P> tag
<a href="#">PARAM Function</a> on page 182-108	Generates the <PARAM> tag which specifies parameter values for Java applets
<a href="#">PLAINTEXT Function</a> on page 182-109	Generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type
<a href="#">PRECLOSE Function</a> on page 182-110	Generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page
<a href="#">PREOPEN Function</a> on page 182-111	Generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page
<a href="#">PRINT Functions</a> on page 182-112	Generates the specified parameter as a string terminated with the \n newline character
<a href="#">PRN Functions</a> on page 182-113	Generates the specified parameter as a string
<a href="#">S Function</a> on page 182-114	Generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type
<a href="#">SAMPLE Function</a> on page 182-115	Generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically
<a href="#">SCRIPT Function</a> on page 182-116	Generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript
<a href="#">SMALL Function</a> on page 182-117	Generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font
<a href="#">STRIKE Function</a> on page 182-118	Generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type
<a href="#">STRONG Function</a> on page 182-119	Generates the <STRONG> and </STRONG> tags which direct the browser to render the text they surround in bold or however "strong" is defined stylistically
<a href="#">STYLE Function</a> on page 182-120	Generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page

**Table 182-1 (Cont.) HTF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SUB Function</a> on page 182-121	Generates the <SUB> and </SUB> tags which direct the browser to render the text they surround as subscript
<a href="#">SUP Function</a> on page 182-122	Generates the <SUP> and </SUP> tags which direct the browser to render the text they surround as superscript
<a href="#">TABLECAPTION Function</a> on page 182-123	Generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table
<a href="#">TABLECLOSE Function</a> on page 182-124	Generates the </TABLE> tag which marks the end of an HTML table
<a href="#">TABLEDATA Function</a> on page 182-125	Generates the <TD> and </TD> tags which insert data into a cell of an HTML table
<a href="#">TABLEHEADER Function</a> on page 182-126	Generates the <TH> and </TH> tags which insert a header cell in an HTML table.
<a href="#">TABLEOPEN Function</a> on page 182-127	Generates the <TABLE> tag which marks the beginning of an HTML table
<a href="#">TABLEROWCLOSE Function</a> on page 182-128	Generates the </TR> tag which marks the end of a new row in an HTML table
<a href="#">TABLEROWOPEN Function</a> on page 182-129	Generates the <TR> tag which marks the beginning of a new row in an HTML table
<a href="#">TELETYPE Function</a> on page 182-130	Generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font
<a href="#">TITLE Function</a> on page 182-131	Generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window
<a href="#">ULISTCLOSE Function</a> on page 182-132	Generates the </UL> tag which marks the end of an unordered list
<a href="#">ULISTOPEN Function</a> on page 182-133	Generates the <UL> tag which marks the beginning of an unordered list
<a href="#">UNDERLINE Function</a> on page 182-134	Generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline
<a href="#">VARIABLE Function</a> on page 182-135	Generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.
<a href="#">WBR Function</a> on page 182-136	Generates the <WBR> tag, which inserts a soft line break within a section of NOBR text

## ADDRESS Function

This function generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document.

### Syntax

```
HTF.ADDRESS (  
    cvalue          IN          VARCHAR2  
    cnowrap         IN          VARCHAR2  DEFAULT NULL  
    cclear          IN          VARCHAR2  DEFAULT NULL  
    cattributes     IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–2 ADDRESS Function Parameters**

Parameter	Description
cvalue	The string that goes between the <ADDRESS> and </ADDRESS> tags.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is included in the tag
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag

### Examples

This function generates

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```



## ANCHOR Function

This function and the [ANCHOR2 Function](#) functions generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that the [ANCHOR2 Function](#) provides a target and therefore can be used for a frame.

### Syntax

```
HTF.ANCHOR (
  curl          IN          VARCHAR2,
  ctext         IN          VARCHAR2,
  cname         IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–3 ANCHOR Function Parameters**

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

### Usage Notes

This tag accepts several attributes, but either HREF or NAME is required. HREF specifies to where to link. NAME allows this tag to be a target of a hypertext link.

## ANCHOR2 Function

This function and the [ANCHOR Function](#) generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that this functions provides a target and therefore can be used for a frame.

### Syntax

```
HTF.ANCHOR2 (  
    curl          IN          VARCHAR2,  
    ctext         IN          VARCHAR2,  
    cname         IN          VARCHAR2  DEFAULT NULL,  
    ctarget       in          varchar2  DEFAULT NULL,  
    cattributes   IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–4 ANCHOR2 Function Parameters**

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag

### Examples

This function generates

```
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

## APPLETCLOSE Function

This function closes the applet invocation with the `</APPLET>` tag. You must first invoke the a Java applet using [APPLETOPEN Function](#) on page 182-20

### Syntax

```
HTF.APPLETCLOSE  
RETURN VARCHAR2;
```

## APPLETOPEN Function

This function generates the <APPLET> tag which begins the invocation of a Java applet. You close the applet invocation with [APPLETCLOSE Function](#) on page 182-19 which generates the </APPLET> tag.

### Syntax

```
HTF.APPLETOPEN (
    ccode          IN          VARCHAR2,
    cheight        IN          NUMBER,
    cwidth         IN          NUMBER,
    cattributes    IN          VARCHAR2    DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–5** *APPLETOPEN Function Parameters*

Parameter	Description
ccode	The the value for the CODE attribute which specifies the name of the applet class.
cheight	The value for the HEIGHT attribute.
cwidth	The value for the WIDTH attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<APPLET CODE=ccode HEIGHT=cheight WIDTH=cwidth cattributes>
```

so that, for example,

```
HTF.appletopen('testclass.class', 100, 200, 'CODEBASE="/ows-applets"')
```

generates

```
<APPLET CODE="testclass.class" height=100 width=200 CODEBASE="/ows-applets">
```

### Usage Notes

- Specify parameters to the Java applet using the [PARAM Function](#) function on page 182-108.
- Use the `cattributes` parameter to specify the `CODEBASE` attribute since the PL/SQL cartridge does not know where to find the class files. The `CODEBASE` attribute specifies the virtual path containing the class files.

## AREA Function

This function generates the <AREA> tag, which defines a client-side image map. The <AREA> tag defines areas within the image and destinations for the areas.

### Syntax

```
HTF.AREA (
  ccoords      IN      VARCHAR2
  cshape       IN      VARCHAR2  DEFAULT NULL,
  chref        IN      VARCHAR2  DEFAULT NULL,
  cnohref      IN      VARCHAR2  DEFAULT NULL,
  ctargt       IN      VARCHAR2  DEFAULT NULL,
  cattributes  IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–6 AREA Function Parameters**

Parameter	Description
ccords	The the value for the COORDS attribute.
cshape	The value for the SHAPE attribute.
chref	The value for the HREF attribute.
cnohref	If the value for this parameter is not NULL, the NOHREF attribute is added to the tag.
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<AREA COORDS="ccords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget"
cattributes>
```

## BASE Function

This function generates the <BASE> tag which records the URL of the document.

### Syntax

```
HTF.BASE (  
    ctarget          IN          VARCHAR2    DEFAULT NULL,  
    cattributes      IN          VARCHAR2    DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–7** *BASE Function Parameters*

Parameter	Description
ctarget	The value for the TARGET attribute which establishes a window name to which all links in this document are targeted.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BASE HREF="<current URL>" TARGET="ctarget" cattributes>
```

## BASEFONT Function

This function generates the <BASEFONT> tag which specifies the base font size for a Web page.

### Syntax

```
HTF.BASEFONT (  
    nsize    IN    INTEGER)  
    RETURN VARCHAR2;
```

### Parameters

**Table 182–8** *BASEFONT Function Parameters*

Parameter	Description
nsize	The value for the SIZE attribute.

### Examples

This function generates

```
<BASEFONT SIZE="nsize">
```

## BGSOUND Function

This function generates the <BGSOUND> tag which includes audio for a Web page.

### Syntax

```
HTF.BGSOUND (  
    csrc          IN          VARCHAR2,  
    cloop        IN          VARCHAR2  DEFAULT NULL,  
    cattributes  IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–9 BGSOUND Function Parameters**

Parameter	Description
csrc	The value for the SRC attribute.
cloop	The value for the LOOP attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BGSOUND SRC="csrc" LOOP="cloop" cattributes>
```



## BIG Function

This function generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font.

### Syntax

```
HTF.BIG (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–10** *BIG Function Parameters*

Parameter	Description
ctext	The the text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BIG cattributes>ctext</BIG>
```

## BLOCKQUOTECLOSE Function

This function generates the `</BLOCKQUOTE>` tag which mark the end of a section of quoted text. You mark the beginning of a section of text by means of the [BLOCKQUOTEOPEN Function](#).

### Syntax

```
HTF.BLOCKQUOTECLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</BLOCKQUOTE>
```

## BLOCKQUOTEOPEN Function

This function generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text. You mark the end of a section of text by means of the [BLOCKQUOTECLOSE Function](#).

### Syntax

```
HTF.BLOCKQUOTEOPEN (
  cnowrap      IN      VARCHAR2  DEFAULT NULL,
  cclear       IN      VARCHAR2  DEFAULT NULL,
  cattributes  IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–11** *BLOCKQUOTEOPEN Function Parameters*

Parameter	Description
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

## BODYCLOSE Function

This function generates the `</BODY>` tag which marks the end of a body section of an HTML document. You mark the beginning of a body section by means of the [BODYOPEN Function](#).

### Syntax

```
HTF.BODYCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</BODY>
```

## BODYOPEN Function

This function generates the <BODY> tag which marks the beginning of the body section of an HTML document. You mark the end of a body section by means of the [BODYCLOSE Function](#).

### Syntax

```
HTF.BODYOPEN (
  cbackground   IN      VARCHAR2   DEFAULT NULL,
  cattributes   IN      VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–12 BODYOPEN Function Parameters**

Parameter	Description
cbackground	The value for the BACKGROUND attribute which specifies a graphic file to use for the background of the document.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BODY background="cbackground" cattributes>
```

so that

```
HTF.BODYOPEN('/img/background.gif')
RETURN VARCHAR2;
```

generates:

```
<BODY background="/img/background.gif">
```

## BOLD Function

This function generates the <B> and </B> tags which direct the browser to display the text in boldface.

### Syntax

```
HTF.BOLD (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–13 BOLD Function Parameters**

Parameter	Description
ctext	The text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<B cattributes>ctext</B>
```

## BR Function

This function generates the <BR> tag which begins a new line of text. It performs the same operation as the [NL Function](#).

### Syntax

```
HTF.BR(  
  cclear          IN          VARCHAR2  DEFAULT NULL,  
  cattributes     IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–14 BR Function Parameters**

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BR CLEAR="cclear" cattributes>
```

## CENTER Function

This function generates the <CENTER> and </CENTER> tags which center a section of text within a Web page.

### Syntax

```
HTF.CENTER (  
    ctext          IN          VARCHAR2)  
    RETURN VARCHAR2;
```

### Parameters

**Table 182–15** *CENTER Parameters*

Parameter	Description
ctext	The text that goes between the tags.

### Examples

This function generates

```
<CENTER>ctext</CENTER>
```



## CENTERCLOSE Function

This function generates the `</CENTER>` tag which marks the end of a section of text to center. You mark the beginning of a section of text to center by means of the [CENTEROPEN Function](#).

### Syntax

```
HTF.CENTERCLOSE  
    RETURN VARCHAR2;
```

### Examples

This function generates

```
</CENTER>
```

## CENTEROPEN Function

This function generates the <CENTER> tag which mark the beginning of a section of text to center. You mark the beginning of a of a section of text to center by means of the [CENTERCLOSE Function](#).

### Syntax

```
HTF.CENTEROPEN  
    RETURN VARCHAR2;
```

### Examples

This function generates

```
<CENTER>
```

## CITE Function

This function generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation.

### Syntax

```
HTF.CITE (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–16** CITE Function Parameters

Parameter	Description
ctext	The text to render as citation.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<CITE cattributes>ctext</CITE>
```

## CODE Function

This function generates the `<CODE>` and `</CODE>` tags which direct the browser to render the text in monospace font or however "code" is defined stylistically.

### Syntax

```
HTF.CODE (  
  ctext          IN          VARCHAR2,  
  cattributes   IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–17** *CODE Function Parameters*

Parameter	Description
<code>ctext</code>	The text to render as code.
<code>cattributes</code>	The other attributes to be included as-is in the tag

### Examples

This function generates

```
<CODE cattributes>ctext</CODE>
```

## COMMENT Function

This function generates the comment tags.

### Syntax

```
HTF.COMMENT (  
    ctext          IN          VARCHAR2)  
    RETURN VARCHAR2;
```

### Parameters

**Table 182–18** *COMMENT Function Parameters*

Parameter	Description
ctext	The comment.

### Examples

This function generates

```
<!-- ctext -->
```

## DFN Function

This function generates the <DFN> and </DFN> tags which direct the browser to mark the text in italics or however "definition" is described stylistically.

### Syntax

```
HTF.DFN (  
    ctext          IN          VARCHAR2)  
    RETURN VARCHAR2;
```

### Parameters

**Table 182–19 DFN Function Parameters**

Parameter	Description
ctext	The text to render in italics.

### Examples

This function generates

```
<DFN>ctext</DFN>
```

## DIRLISTCLOSE Function

This function generates the `</DIR>` tag which ends a directory list section. You start a directory list section with the [DIRLISTOPEN Function](#).

### Syntax

```
HTF.DIRLISTCLOSE  
RETURN VARCHAR2;
```

### Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the `<LI>` tag directly or invoke the [LISTITEM Function](#) so that the `<LI>` tag appears directly after the `</DIR>` tag to define the items as a list.

### Examples

This function generates

```
</DIR>
```

## DIRLISTOPEN Function

This function generates the <DIR> which starts a directory list section. You end a directory list section with the [DIRLISTCLOSE Function](#).

### Syntax

```
HTF.DIRLISTOPEN  
    RETURN VARCHAR2;
```

### Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the [LISTITEM Function](#) so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

### Examples

This function generates

```
<DIR>
```



## DIV Function

This function generates the <DIV> tag which creates document divisions.

### Syntax

```
HTF.DIV (  
    calign          IN          VARCHAR2  DEFAULT NULL,  
    cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–20** *DIV Function Parameters*

Parameter	Description
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<DIV ALIGN="calign" cattributes>
```

## DLISTCLOSE Function

This function generates the `</DL>` tag which ends a definition list. You start a definition list by means of the [DLISTOPEN Function](#).

### Syntax

```
HTF.DLISTCLOSE  
RETURN VARCHAR2;
```

### Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the [DLISTTERM Function](#) and definitions are inserted using the [DLISTDEF Function](#).

### Examples

This function generates

```
</DL>
```

## DLISTDEF Function

This function generates the <DD> tag, which inserts definitions of terms. Use this tag for a definition list <DL>. Terms are tagged <DT> and definitions are tagged <DD>.

### Syntax

```
HTF.DLISTDEF (
  ctext          IN          VARCHAR2  DEFAULT NULL,
  cclear         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–21** *DLISTDEF Function Parameters*

Parameter	Description
ctext	The definition of the term.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<DD CLEAR="cclear" cattributes>ctext
```

## DLISTOPEN Function

This function generates the <DL> tag which starts a definition list. You end a definition list by means of the [DLISTCLOSE Function](#).

### Syntax

```
HTF.DLISTOPEN (  
    cclear          IN          VARCHAR2    DEFAULT NULL,  
    cattributes     IN          VARCHAR2    DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–22** *DLISTOPEN Function Parameters*

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the [DLISTTERM Function](#) and definitions are inserted using the [DLISTDEF Function](#).

### Examples

This function generates

```
<DL CLEAR="cclear" cattributes>
```

## DLISTTERM Function

This function generates the <DT> tag which defines a term in a definition list <DL>.

### Syntax

```
HTF.DLISTTERM (
  ctext          IN          VARCHAR2  DEFAULT NULL,
  cclear        IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–23** *DLISTTERM Function Parameters*

Parameter	Description
ctext	The term.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<DT CLEAR="cclear" cattributes>ctext
```

## EM Function

This function generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the [EMPHASIS Function](#).

### Syntax

```
HTF.EM(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–24** *EM Function Parameters*

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<EM cattributes>ctext</EM>
```

## EMPHASIS Function

This function generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the [EM Function](#).

### Syntax

```
HTF.EMPHASIS(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–25 EMPHASIS Function Parameters**

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<EM cattributes>ctext</EM>
```

## ESCAPE\_SC Function

This function replaces characters that have special meaning in HTML with their escape sequences. The following characters are converted:

- & to &amp;
- " to &quot;
- < to &lt;
- > to &gt;

This function performs the same operation as HTP. [PRINTS Procedure](#) and HTP. [PS Procedure](#).

### Syntax

```
HTF.ESCAPE_SC(  
    ctext          IN          VARCHAR2);
```

### Parameters

**Table 182–26** *ESCAPE\_SC Procedure Parameters*

Parameter	Description
ctext	The text string to convert.



## ESCAPE\_URL Function

---



---

**Note:** This procedure, deprecated in Release 10g, and provided here only for reasons of backward compatibility, does not comply with the Internet Engineering Task Force (IETF) Request for Comments (RFC) standards of URL encoding. If you need to encode URLs, it is recommended you use the [ESCAPE Function](#) in the [UTL\\_URL](#) package.

---



---

This function replaces characters that have special meaning in HTML and HTTP with their escape sequences. The following characters are converted:

- & to &amp;
- " to &quot;
- < to &lt;
- > to &gt;
- % to &25

### Syntax

```
HTF.ESCAPE_URL(
    p_url          IN          VARCHAR2);
```

### Parameters

**Table 182–27** *ESCAPE\_URL Procedure Parameters*

Parameter	Description
p_url	The string to convert.

## FONTCLOSE Function

This function generates the `</FONT>` tag which marks the end of a section of text with the specified font characteristics. You mark the beginning of the section text by means of the [FONTOPEN Function](#).

### Syntax

```
HTF.FONTCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</FONT>
```

## FONTOPEN Function

This function generates the <FONT> which marks the beginning of section of text with the specified font characteristics. You mark the end of the section text by means of the [FONTCLOSE Function](#).

### Syntax

```
HTF.FONTOPEN(
  ccolor      IN      VARCHAR2  DEFAULT NULL,
  cface       IN      VARCHAR2  DEFAULT NULL,
  csize       IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–28** *FONTOPEN Function Parameters*

Parameter	Description
ccolor	The value for the COLOR attribute.
cface	The value for the FACE attribute
csize	The value for the SIZE attribute
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

## FORMAT\_CELL Function

This function formats column values inside an HTML table using the [TABLEDATA Function](#). It allows for better control over the HTML tables.

### Syntax

```
HTF.FORMAT_CELL(  
    columnValue      IN      VARCHAR2  
    format_numbers  IN      VARCHAR2  DEFAULT NULL  
RETURN VARCHAR2;
```

### Parameters

**Table 182–29** *FORMAT\_CELL Function Parameters*

Parameter	Description
columnValue	The value that needs to be formatted in an HTML table.
format_numbers	The format that numeric data is displayed in. If the value of this parameter is not NULL, the number fields are right-justified and rounded to two decimal places.

### Examples

This function generates

```
<TD >columnValue</TD>
```

## FORMCHECKBOX Function

This function generates the <INPUT> tag with `TYPE="checkbox"` which inserts a checkbox element in a form. A checkbox element is a button that the user toggles on or off.

### Syntax

```
HTF.FORMCHECKBOX(
  cname          IN          VARCHAR2,
  cvalue         IN          VARCHAR2  DEFAULT 'ON',
  cchecked       IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–30** *FORMCHECKBOX Function Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cchecked</code>	If the value for this parameter is not <code>NULL</code> , the <code>CHECKED</code> attribute is added to the tag.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

## FORMCLOSE Function

This function generates the `</FORM>` tag which marks the end of a form section in an HTML document. You mark the beginning of the form section by means of the [FORMOPEN Function](#).

### Syntax

```
HTF.FORMCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</FORM>
```

## FORMFILE Function

This function generates the <INPUT> tag with `TYPE="file"` which inserts a file form element. This is used for file uploading for a given page.

### Syntax

```
HTF.FORMFILE(
  cname          IN          VARCHAR2,
  caccept        IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–31** *FORMFILE Function Parameters*

Parameter	Description
cname	The value for the NAME attribute.
caccept	A comma-delimited list of MIME types for upload.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes>
```

## FORMHIDDEN Function

This function generates the <INPUT> tag with TYPE="hidden", which inserts a hidden form element. This element is not seen by the user. It submits additional values to the script.

### Syntax

```
HTF.FORMHIDDEN(  
  cname          IN          VARCHAR2,  
  cvalue         IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–32** *FORMHIDDEN Function Parameters*

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```



## FORMIMAGE Function

This function generates the <INPUT> tag with `TYPE="image"` which creates an image field that the user clicks to submit the form immediately. The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with `.x` appended, and the y coordinate with `.y` appended. Any `VALUE` attribute is ignored.

### Syntax

```
HTF.FORMIMAGE (
  cname          IN          VARCHAR2,
  csrc           IN          VARCHAR2,
  calign        IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–33** *FORMIMAGE Function Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>csrc</code>	The value for the <code>SRC</code> attribute that specifies the image file.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

## FORMOPEN Function

This function generates the <FORM> tag which marks the beginning of a form section in an HTML document. You mark the end of the form section by means of the [FORMCLOSE Function](#).

### Syntax

```
HTF.FORMOPEN (
    curl          IN          VARCHAR2,
    cmethod       IN          VARCHAR2  DEFAULT 'POST',
    ctarger       IN          VARCHAR2  DEFAULT NULL,
    cenctype      IN          VARCHAR2  DEFAULT NULL,
    cattributes   IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–34 FORMOPEN Function Parameters**

Parameter	Description
curl	The URL of the Web Request Broker or CGI script where the contents of the form is sent. This parameter is required.
cmethod	The value for the METHOD attribute. The value can be "GET" or "POST".
ctarger	The value for the TARGET attribute.
cenctype	The value for the ENCTYPE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarger" ENCTYPE="cenctype"
cattributes>
```

## FORMPASSWORD Function

This function generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field. When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

### Syntax

```
HTF.FORMPASSWORD(
  cname          IN          VARCHAR2,
  csize          IN          VARCHAR2,
  cmaxlength     IN          VARCHAR2  DEFAULT NULL,
  cvalue         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–35 FORMPASSWORD Function Parameters**

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmmaxlength"
VALUE="cvalue" cattributes>
```

## FORMRADIO Function

This function generates the `<INPUT>` tag with `TYPE="radio"`, which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

### Syntax

```
HTF.FORMRADIO(  
  cname          IN          VARCHAR2,  
  cvalue         IN          VARCHAR2,  
  cchecked       IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–36 FORMRADIO Function Parameters**

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cchecked</code>	If the value for this parameter is not <code>NULL</code> , the <code>CHECKED</code> attribute is added to the tag.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

## FORMRESET Function

This function generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values.

### Syntax

```
HTF.FORMRESET(  
  cvalue          IN          VARCHAR2  DEFAULT 'Reset',  
  cattributes     IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182-37 FORMRESET Function Parameters**

Parameter	Description
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

## FORMSELECTCLOSE Function

This function generates the `</SELECT>` tag which marks the end of a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the beginning of Select form element by means of the [FORMSELECTOPEN Function](#). The values are inserted using [FORMSELECTOPTION Function](#).

### Syntax

```
HTF.FORMSELECTCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</SELECT>
```

as shown under [Examples](#) of the [FORMSELECTOPEN Function](#).

## FORMSELECTOPEN Function

This function generates the <SELECT> tags which creates a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the end of Select form element by means of the [FORMSELECTCLOSE Function](#). The values are inserted using [FORMSELETOPTION Function](#).

### Syntax

```
HTF.FORMSELECTOPEN (
  cname          IN          VARCHAR2,
  cprompt        IN          VARCHAR2  DEFAULT NULL,
  nsize          IN          INTEGER   DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–38** *FORMSELECTOPEN Function Parameters*

Parameter	Description
cname	The value for the NAME attribute.
cprompt	The string preceding the list box.
nsize	The value for the SIZE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
</SELECT>
```

so that

```
HTF.FORMSELECTOPEN('greatest_player';
  'Pick the greatest player:');
HTF.FORMSELETOPTION('Messier');
HTF.FORMSELETOPTION('Howe');
HTF.FORMSELETOPTION('Gretzky');.
HTF.FORMSELECTCLOSE;
```

generates

```
Pick the greatest player:
<SELECT NAME="greatest_player">
<OPTION>Messier
<OPTION>Howe
<OPTION>Gretzky
</SELECT>
```

## FORMSELETOPTION Function

This function generates the <OPTION> tag which represents one choice in a Select element.

### Syntax

```
HTF.FORMSELETOPTION(  
    cvalue          IN          VARCHAR2,  
    cselected       IN          VARCHAR2  DEFAULT NULL,  
    cattributes     IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–39** *FORMSELETOPTION Function Parameters*

Parameter	Description
cvalue	The text for the option.
cvalue	If the value for this parameter is not NULL, the SELECTED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<OPTION SELECTED cattributes>cvalue
```

as shown under [Examples](#) of the [FORMSELETOPTION Function](#).



## FORMSUBMIT Function

This function generates the <INPUT> tag with `TYPE="submit"` which creates a button that, when clicked, submits the form. If the button has a `NAME` attribute, the button contributes a name/value pair to the submitted data.

### Syntax

```
HTF.FORMSUBMIT(
  cname          IN          VARCHAR2  DEFAULT NULL,
  cvalue         IN          VARCHAR2  DEFAULT 'Submit',
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–40 FORMSUBMIT Function Parameters**

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

## FORMTEXT Function

This function generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text.

### Syntax

```
HTF.FORMTEXT(  
  cname          IN          VARCHAR2,  
  csize          IN          VARCHAR2  DEFAULT NULL,  
  cmaxlength     IN          VARCHAR2  DEFAULT NULL,  
  cvalue         IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–41** FORMTEXT Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmmaxlength" VALUE="cvalue"  
cattributes>
```

## FORMTEXTAREA Function

This function generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text. The same operation is performed by the [FORMTEXTAREA2 Function](#) which in addition has the `cwrap` parameter that lets you specify a wrap style.

### Syntax

```
HTF.FORMTEXTAREA(
  cname          IN          VARCHAR2,
  nrows          IN          INTEGER,
  ncolumns       IN          INTEGER,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–42** *FORMTEXTAREA Function Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>nrows</code>	The value for the <code>ROWS</code> attribute. This is an integer.
<code>ncolumns</code>	The value for the <code>COLS</code> attribute. This is an integer.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
cattributes></TEXTAREA>
```

## FORMTEXTAREA2 Function

This function generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text. The same operation is performed by the [FORMTEXTAREA Function](#) except that in that case you cannot specify a wrap style.

### Syntax

```
HTF.FORMTEXTAREA2 (
    cname          IN          VARCHAR2,
    nrows          IN          INTEGER,
    ncolumns       IN          INTEGER,
    calign         IN          VARCHAR2  DEFAULT NULL,
    cwrap          IN          VARCHAR2  DEFAULT NULL,
    cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–43 FORMTEXTAREA2 Function Parameters**

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute. This is an integer.
ncolumns	The value for the COLS attribute. This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP="cwrap"
cattributes></TEXTAREA>
```

## FORMTEXTAREACLOSE Function

This function generates the `</TEXTAREA>` tag which ends a text area form element. You open a text area element by means of either [FORMTEXTAREAOPEN Function](#) or [FORMTEXTAREAOPEN2 Function](#).

### Syntax

```
HTF.FORMTEXTAREACLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</TEXTAREA>
```

## FORMTEXTAREAOPEN Function

This function generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the [FORMTEXTAREAOPEN2 Function](#) which in addition has the `cwrap` parameter that lets you specify a wrap style. You mark the end of a text area form element by means of the [FORMTEXTAREACLOSE Function](#).

### Syntax

```
HTF.FORMTEXTAREAOPEN (
  cname          IN          VARCHAR2,
  nrows          IN          INTEGER,
  ncolumns       IN          INTEGER,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–44** *FORMTEXTAREAOPEN Function Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>nrows</code>	The value for the <code>ROWS</code> attribute. This is an integer.
<code>ncolumns</code>	The value for the <code>COLS</code> attribute. This is an integer.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes>
```

## FORMTEXTAREAOPEN2 Function

This function generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the [FORMTEXTAREAOPEN Function](#) except that in that case you cannot specify a wrap style. You mark the end of a text area form element by means of the [FORMTEXTAREACLOSE Function](#).

### Syntax

```
HTF.FORMTEXTAREAOPEN2 (
  cname          IN          VARCHAR2,
  nrows          IN          INTEGER,
  ncolumns       IN          INTEGER,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cwrap         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–45** *FORMTEXTAREAOPEN2 Function Parameters*

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute. This is an integer.
ncolumns	The value for the COLS attribute. This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP = "cwrap"
cattributes>
```

## FRAME Function

This function generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag.

### Syntax

```
HTF.FRAME (
  csrc          IN          VARCHAR2,
  cname         IN          VARCHAR2  DEFAULT NULL,
  cmarginwidth IN          VARCHAR2  DEFAULT NULL,
  cmarginheight IN        VARCHAR2  DEFAULT NULL,
  cscrolling    IN          VARCHAR2  DEFAULT NULL,
  cnoresize     IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–46** *FRAME Function Parameters*

Parameter	Description
csrc	The URL to display in the frame.
cname	The value for the NAME attribute.
cmarginwidth	The value for the MARGINWIDTH attribute.
cscrolling	The value for the SCROLLING attribute.
cnoresize	If the value for this parameter is not NULL, the NORESIZE attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth"
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```



## FRAMESETCLOSE Function

This function generates the `</FRAMESET>` tag which ends a frameset section. You mark the beginning of a frameset section by means of the [FRAMESETOPEN Function](#).

### Syntax

```
HTF.FRAMESETCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</FRAMESET>
```

## FRAMESETOPEN Function

This function generates the <FRAMESET> tag which define a frameset section. You mark the end of a frameset section by means of the [FRAMESETCLOSE Function](#).

### Syntax

```
HTF.FRAMESETOPEN(  
  crows          IN          VARCHAR2  DEFAULT NULL,  
  ccols          IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–47** *FRAMESETOPEN Function Parameters*

Parameter	Description
<code>crows</code>	The value for the <code>ROWS</code> attribute.
<code>ccols</code>	The value for the <code>COLS</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

## HEADCLOSE Function

This function generates the `</HEAD>` tag which marks the end of an HTML document head section. You mark the beginning of an HTML document head section by means of the [HEADOPEN Function](#).

### Syntax

```
HTF.HEADCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</HEAD>
```

## HEADER Function

This function generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>).

### Syntax

```
HTF.HEADER(  
  nsize      IN      INTEGER,  
  cheader    IN      VARCHAR2,  
  calign     IN      VARCHAR2  DEFAULT NULL,  
  cnowrap    IN      VARCHAR2  DEFAULT NULL,  
  cclear     IN      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–48** HEADER Function Parameters

Parameter	Description
nsize	The the heading level. This is an integer between 1 and 6.
cheader	The text to display in the heading.
calign	The value for the ALIGN attribute.
cnowrap	The value for the NOWRAP attribute.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

```
HTF.header (1, 'Overview')  
  RETURN VARCHAR2;
```

produces:

```
<H1>Overview</H1>
```

## HEADOPEN Function

This function generates the <HEAD> tag which marks the beginning of the HTML document head section. You mark the end of an HTML document head section by means of the [HEADCLOSE Function](#).

### Syntax

```
HTF.HEADOPEN  
RETURN VARCHAR2;
```

### Examples

This function generates

```
<HEAD>
```

## HR Function

This function generates the <HR> tag, which generates a line in the HTML document. This subprogram performs the same operation as the [LINE Function](#).

### Syntax

```
HTF.HR (  
  cclear      IN      VARCHAR2  DEFAULT NULL,  
  csrc        IN      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–49** HR Function Parameters

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>csrc</code>	The value for the SRC attribute which specifies a custom image as the source of the line.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

## HTMLCLOSE Function

This function generates the `</HTML>` tag which marks the end of an HTML document. You use the [HTMLOPEN Function](#) to mark the beginning of an HTML document.

### Syntax

```
HTF.HTMLCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</HTML>
```

## HTMLOPEN Function

This function generates the <HTML> tag which marks the beginning of an HTML document. You use the [HTMLCLOSE Function](#) to mark the end of the an HTML document.

### Syntax

```
HTF.HTMLOPEN  
RETURN VARCHAR2;
```

### Examples

This function generates

```
<HTML>
```



## IMG Function

This function generates the <IMG> tag which directs the browser to load an image onto the HTML page. The [IMG2 Function](#) performs the same operation but additionally uses the `cusemap` parameter.

### Syntax

```
HTF.IMG (
  curl          IN          VARCHAR2  DEFAULT NULL,
  calign        IN          VARCHAR2  DEFAULT NULL,
  calt          IN          VARCHAR2  DEFAULT NULL,
  cismap        IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–50** *IMG Function Parameters*

Parameter	Description
<code>curl</code>	The value for the SRC attribute.
<code>calign</code>	The value for the ALIGN attribute.
<code>calt</code>	The value for the ALT attribute which specifies alternative text to display if the browser does not support images.
<code>cismap</code>	If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

## IMG2 Function

This function generates the <IMG> tag, which directs the browser to load an image onto the HTML page. The [IMG Function](#) performs the same operation but does not use the `cusemap` parameter.

### Syntax

```
HTF.IMG2 (
    curl          IN          VARCHAR2    DEFAULT NULL,
    calign        IN          VARCHAR2    DEFAULT NULL,
    calt          IN          VARCHAR2    DEFAULT NULL,
    cismap        IN          VARCHAR2    DEFAULT NULL,
    cusemap       IN          VARCHAR2    DEFAULT NULL,
    cattributes   IN          VARCHAR2    DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–51** *IMG2 Function Parameters*

Parameter	Description
<code>curl</code>	The value for the SRC attribute.
<code>calign</code>	The value for the ALIGN attribute.
<code>calt</code>	The value for the ALT attribute which specifies alternative text to display if the browser does not support images.
<code>cismap</code>	If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.
<code>cusemap</code>	The value for the USEMAP attribute which specifies a client-side image map.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap" cattributes>
```

## ISINDEX Function

This function creates a single entry field with a prompting text, such as "*enter value*," then sends that value to the URL of the page or program.

### Syntax

```
HTF.ISINDEX (
  cprompt      IN      VARCHAR2      DEFAULT NULL,
  curl         IN      VARCHAR2      DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–52** *ISINDEX Function Parameters*

Parameter	Description
cprompt	The value for the PROMPT attribute.
curl	The value for the HREF attribute.

### Examples

This function generates

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```

## ITALIC Function

This function generates the `<I>` and `</I>` tags which direct the browser to render the text in italics.

### Syntax

```
HTF.ITALIC(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–53** *ITALIC Function Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in italics.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates  
`<I cattributes>ctext</I>`

## KBD Function

This function generates the <KBD> and </KBD> tags which direct the browser to render the text in monospace font. This subprogram performs the same operation as the [KEYBOARD Function](#).

### Syntax

```
HTF.KBD(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–54** *KBD Function Parameters*

Parameter	Description
ctext	The text to be rendered in monospace.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<KBD cattributes>ctext</KBD>
```

## KEYBOARD Function

This function generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font. This subprogram performs the same operation as the [KBD Function](#).

### Syntax

```
HTF.KEYBOARD(  
  ctext          IN          VARCHAR2,  
  cattributes   IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–55** *KEYBOARD Function Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in monospace.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<KBD cattributes>ctext</KBD>
```

## LINE Function

This function generates the <HR> tag, which generates a line in the HTML document. This subprogram performs the same operation as the [HR Function](#).

### Syntax

```
HTF.LINE(
  cclear      IN      VARCHAR2  DEFAULT NULL,
  csrc        IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–56** *LINE Function Parameters*

Parameter	Description
cclear	The value for the CLEAR attribute.
csrc	The value for the SRC attribute which specifies a custom image as the source of the line.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

## LINKREL Function

This function generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present. This is the opposite of [LINKREV Function](#). This tag indicates a relationship between documents but does not create a link. To create a link, use the [ANCHOR Function](#).

### Syntax

```
HTF.LINKREL(  
    crel          IN          VARCHAR2,  
    curl          IN          VARCHAR2,  
    ctitle       IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–57 LINKREL Function Parameters**

Parameter	Description
crel	The value for the REL attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

### Examples

This function generates

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```



## LINKREV Function

This function generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor. This is the opposite of the [LINKREL Function](#). This tag indicates a relationship between documents, but does not create a link. To create a link, use the [ANCHOR Function](#).

### Syntax

```
HTF.LINKREV(  
  crev          IN          VARCHAR2,  
  curl          IN          VARCHAR2,  
  ctitle       IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–58** LINKREV Function Parameters

Parameter	Description
crev	The value for the REV attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

### Examples

This function generates

```
<LINK REV="crev" HREF="curl" TITLE="ctitle">
```

## LISTHEADER Function

This function generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list.

### Syntax

```
HTF.LISTHEADER(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–59 LISTHEADER Function Parameters**

Parameter	Description
ctext	The text to place between <LH> and </LH>.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<LH cattributes>ctext</LH>
```

## LISTINGCLOSE Function

This function generates the `</LISTING>` tags which marks the end of a section of fixed-width text in the body of an HTML page. To mark the beginning of a section of fixed-width text in the body of an HTML page, use the [LISTINGOPEN Function](#).

### Syntax

```
HTF.LISTINGCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</LISTING>
```

## LISTINGOPEN Function

This function generates the <LISTING> tag which marks the beginning of a section of fixed-width text in the body of an HTML page. To mark the end of a section of fixed-width text in the body of an HTML page, use the [LISTINGCLOSE Function](#).

### Syntax

```
HTF.LISTINGOPEN  
RETURN VARCHAR2;
```

### Examples

This function generates

```
<LISTING>
```

## LISTITEM Function

This function generates the <LI> tag, which indicates a list item.

### Syntax

```
HTF.LISTITEM(
  ctext          IN          VARCHAR2  DEFAULT NULL,
  cclear         IN          VARCHAR2  DEFAULT NULL,
  cdingbat       IN          VARCHAR2  DEFAULT NULL,
  csrc           IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–60** LISTITEM Function Parameters

Parameter	Description
ctext	The text for the list item.
cclear	The value for the CLEAR attribute.
cdingbat	The value for the DINGBAT attribute.
csrc	The value for the SRC attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

## MAILTO Function

This function generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument.

### Syntax

```
HTF.MAILTO(
  address      IN      VARCHAR2,
  ctext       IN      VARCHAR2,
  cname       IN      VARCHAR2,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–61 MAILTO Function Parameters**

Parameter	Description
address	The email address of the recipient.
ctext	The clickable portion of the link.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<A HREF="mailto:address" NAME="cname" cattributes>ctext</A>
```

so that

```
HTF.mailto('pres@white_house.gov','Send Email to the President');
```

generates:

```
<A HREF="mailto:pres@white_house.gov">Send Email to the President</A>
```

## MAPCLOSE Function

This function generates the `</MAP>` tag which marks the end of a set of regions in a client-side image map. To mark the beginning of a set of regions in a client-side image map, use the [MAPOPEN Function](#).

### Syntax

```
HTF.MAPCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</MAP>
```

## MAOPEN Function

This function generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map. To mark the end of a set of regions in a client-side image map, use the [MAPCLOSE Function](#).

### Syntax

```
HTF.MAOPEN(  
  cname          IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–62** MAOPEN Function Parameters

Parameter	Description
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<MAP NAME="cname" cattributes>
```



## MENULISTCLOSE Function

This function generates the `</MENU>` tag which ends a list that presents one line for each item. To begin a list of this kind, use the [MENULISTOPEN Function](#). The items in the list appear more compact than an unordered list. The [LISTITEM Function](#) defines the list items in a menu list.

### Syntax

```
HTF.MENULISTCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</MENU>
```

## MENULISTOPEN Function

This function generates the <MENU> tag which create a list that presents one line for each item. To end a list of this kind, use the [MENULISTCLOSE Function](#). The items in the list appear more compact than an unordered list. The [LISTITEM Function](#) defines the list items in a menu list.

### Syntax

```
HTF.MENULISTOPEN  
    RETURN VARCHAR2;
```

### Examples

This function generates

```
<MENU>
```

## META Function

This function generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers. For example, you can specify the expiration date, keywords, and author name.

### Syntax

```
HTF.META(
  chttp_equiv    IN      VARCHAR2,
  cname          IN      VARCHAR2,
  ccontent       IN      VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 182–63 META Function Parameters**

Parameter	Description
chttp_equiv	The value for the CHTTP_EQUIV attribute.
cname	The value for the NAME attribute.
ccontent	The value for the CONTENT attribute.

### Examples

This function generates

```
<META HTTP-EQUIV="chttp_equiv" NAME ="cname" CONTENT="ccontent">
```

so that

```
HTF.meta ('Refresh', NULL, 120);
```

generates

```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```

On some Web browsers, this causes the current URL to be reloaded automatically every 120 seconds.

## NL Function

This function generates the <BR> tag which begins a new line of text. It performs the same operation as the [BR Function](#).

### Syntax

```
HTF.NL(  
  cclear      IN      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–64 NL Function Parameters**

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<BR CLEAR="cclear" cattributes>
```

## NOBR Function

This function generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text.

### Syntax

```
HTF.NOBR (  
  ctext          IN          VARCHAR2)  
  RETURN VARCHAR2;
```

### Parameters

**Table 182–65** *NOBR Function Parameters*

Parameter	Description
c text	The text that is to be rendered on one line.

### Examples

This function generates

```
<NOBR>c text</NOBR>
```

## NOFRAMESCLOSE Function

This function generates the `</NOFRAMES>` tag which marks the end of a no-frames section. To mark the beginning of a no-frames section, use the [FRAMESETOPEN Function](#). See also [FRAME Function](#), [FRAMESETOPEN Function](#) and [FRAMESETCLOSE Function](#).

### Syntax

```
HTF.NOFRAMESCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</NOFRAMES>
```

## NOFRAMESOPEN Function

This function generates the <NOFRAMES> tag which mark the beginning of a no-frames section. To mark the end of a no-frames section, use the [FRAMESETCLOSE Function](#). See also [FRAME Function](#), [FRAMESETOPEN Function](#) and [FRAMESETCLOSE Function](#).

### Syntax

```
HTF.NOFRAMESOPEN  
    RETURN VARCHAR2;
```

### Examples

This function generates

```
<NOFRAMES>
```

## OLISTCLOSE Function

This function generates the `</OL>` tag which defines the end of an ordered list. An ordered list presents a list of numbered items. To mark the beginning of a list of this kind, use the [OLISTOPEN Function](#). Numbered items are added using [LISTITEM Function](#).

### Syntax

```
HTF.OLISTCLOSE  
    RETURN VARCHAR2;
```

### Examples

This function generates

```
</OL>
```



## OLISTOPEN Function

This function generates the <OL> tag which marks the beginning of an ordered list. An ordered list presents a list of numbered items. To mark the end of a list of this kind, use the [OLISTCLOSE Function](#). Numbered items are added using [LISTITEM Function](#).

### Syntax

```
HTF.OLISTOPEN(
  cclear      IN      VARCHAR2  DEFAULT NULL,
  cwrap       IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–66** *OLISTOPEN Function Parameters*

Parameter	Description
cclear	The value for the CLEAR attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

## PARA Function

This function generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph. You can add attributes to the tag by means of the [PARAGRAPH Function](#).

### Syntax

```
HTF.PARA
RETURN VARCHAR2;
```

### Examples

This function generates

```
<P>
```

## PARAGRAPH Function

You can use this function to add attributes to the <P> tag created by the [PARA Function](#).

### Syntax

```
HTF.PARAGRAPH(
  calign          IN          VARCHAR2  DEFAULT NULL,
  cnowrap        IN          VARCHAR2  DEFAULT NULL,
  cclear         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–67 PARAGRAPH Function Parameters**

Parameter	Description
calign	The value for the ALIGN attribute.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

## PARAM Function

This function generates the <PARAM> tag which specifies parameter values for Java applets. The values can reference HTML variables. To invoke a Java applet from a Web page, use [APPLETOPEN Function](#) to begin the invocation. Use one [PARAM Function](#) for each desired name-value pair, and use [APPLETCLOSE Function](#) to end the applet invocation.

### Syntax

```
HTF.PARAM(  
  cname          IN          VARCHAR2  
  cvalue         IN          VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–68** *PARAM Function Parameters*

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.

### Examples

This function generates

```
<PARAM NAME=cname VALUE="cvalue">
```

## PLAINTEXT Function

This function generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type.

### Syntax

```
HTF.PLAINTEXT(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–69** *PLAINTEXT Function Parameters*

Parameter	Description
ctext	The text to be rendered in fixed-width font.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<PLAINTEXT cattributes>ctext</PLAINTEXT>
```

## PRECLOSE Function

This function generates the `</PRE>` tag which marks the end of a section of preformatted text in the body of the HTML page. To mark the beginning of a section of preformatted text in the body of the HTML page, use the [PREOPEN Function](#).

### Syntax

```
HTF.PRECLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</PRE>
```

## PREOPEN Function

This function generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page. To mark the end of a section of preformatted text in the body of the HTML page, use the [PRECLOSE Function](#).

### Syntax

```
HTF.PREOPEN(
  cclear      IN      VARCHAR2  DEFAULT NULL,
  cwidth      IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–70** *PREOPEN Function Parameters*

Parameter	Description
cclear	The value for the CLEAR attribute.
cwidth	The value for the WIDTH attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

## PRINT Functions

These functions generate the specified parameter as a string terminated with the \n newline character. The [PRN Functions](#) performs the same operation but does not terminate with a newline character.

### Syntax

```
HTF.PRINT (  
    cbuf      IN      VARCHAR2)  
RETURN VARCHAR2;
```

```
HTF.PRINT (  
    dbuf      IN      DATE)  
RETURN VARCHAR2;
```

```
HTF.PRINT (  
    nbuf      IN      NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–71 PRINT Function Parameters**

Parameter	Description
cbuf	The string to generate terminated by a newline.
dbuf	The string to generate terminated by a newline.
nbuf	The string to generate terminated by a newline.

### Usage Notes

- The \n character is not the same as <BR>. The \n character formats the HTML source but it does not affect how the browser renders the HTML source. Use <BR> to control how the browser renders the HTML source.
- These functions do not have function equivalents.



## PRN Functions

These functions generate the specified parameter as a string. Unlike the [PRINT Functions](#) the string is not terminated with the \n newline character.

### Syntax

```
HTF.PRN (  
    cbuf      IN      VARCHAR2)  
RETURN VARCHAR2;
```

```
HTF.PRN (  
    dbuf      IN      DATE)  
RETURN VARCHAR2;
```

```
HTF.PRN (  
    nbuf      IN      NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–72 PRN Function Parameters**

Parameter	Description
cbuf	The string to generate (not terminated by a newline).
dbuf	The string to generate (not terminated by a newline).
nbuf	The string to generate (not terminated by a newline).

### Usage Notes

These functions do not have function equivalents.

## S Function

This function generates the `<S>` and `</S>` tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as [STRIKE Function](#).

### Syntax

```
HTF.S (  
    ctext          IN          VARCHAR2,  
    cattributes   IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–73 S Function Parameters**

Parameter	Description
<code>ctext</code>	The text to be rendered in strikethrough type.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<S cattributes>ctext</S>
```

## SAMPLE Function

This function generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically.

### Syntax

```
HTF.SAMPLE (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–74** *SAMPLE Function Parameters*

Parameter	Description
ctext	The text to be rendered in monospace font.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<SAMP cattributes>ctext</SAMP>
```

## SCRIPT Function

This function generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript.

### Syntax

```
HTF.SCRIPT (  
    cscript          IN          VARCHAR2,  
    clanguage        IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–75** *SCRIPT Function Parameters*

Parameter	Description
cscript	The text of the script. This is the text that makes up the script itself, not the name of a file containing the script.
clanguage	The language in which the script is written. If this parameter is omitted, the user's browser determines the scripting language.

### Examples

This function generates

```
<SCRIPT LANGUAGE=clanguage>cscript</SCRIPT>
```

so that

```
HTF.script ('Erupting_Volcano', 'Javascript');
```

generates

```
<SCRIPT LANGUAGE=Javascript>"script text here"</SCRIPT>
```

This causes the browser to run the script enclosed in the tags.

## SMALL Function

This function generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font.

### Syntax

```
HTF.SMALL (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–76** *SMALL Function Parameters*

Parameter	Description
ctext	The text to be rendered in small font.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<SMALL cattributes>ctext</SMALL>
```

## STRIKE Function

This function generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as [S Function](#).

### Syntax

```
STRIKE (  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–77** *STRIKE Function Parameters*

Parameter	Description
ctext	The text to be rendered in strikethrough type.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<STRIKE cattributes>ctext</STRIKE>
```

## STRONG Function

This function generates the `<STRONG>` and `</STRONG>` tags which direct the browser to render the text they surround in bold or however "strong" is defined.

### Syntax

```
HTF.STRONG(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–78** *STRONG Function Parameters*

Parameter	Description
<code>ctext</code>	The text to be emphasized.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<STRONG cattributes>ctext</STRONG>
```

## STYLE Function

This function generates the `<STYLE>` and `</STYLE>` tags which include a style sheet in a Web page. You can get more information about style sheets at <http://www.w3.org>. This feature is not compatible with browsers that support only HTML versions 2.0 or earlier. Such browsers will ignore this tag.

### Syntax

```
HTF.STYLE(  
    cstyle          IN          VARCHAR2)  
    RETURN VARCHAR2;
```

### Parameters

**Table 182–79** *STYLE Function Parameters*

Parameter	Description
<code>cstyle</code>	The the style information to include.

### Examples

This function generates

```
<STYLE>cstyle</STYLE>
```



## SUB Function

This function generates the <SUB> and </SUB> tags which direct the browser to render the text they surround as subscript.

### Syntax

```
HTF.SUB (
  ctext          IN          VARCHAR2,
  calign         in          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–80 SUB Function Parameters**

Parameter	Description
ctext	The text to render in subscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<SUB ALIGN="calign" cattributes>ctext</SUB>
```

## SUP Function

This function generates the <SUP> and </SUP> tags which direct the browser to render the text they surround as superscript.

### Syntax

```
HTF.SUP(  
  ctext          IN          VARCHAR2,  
  calign         in          VARCHAR2  DEFAULT NULL,  
  cattributes   IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–81** SUP Function Parameters

Parameter	Description
ctext	The text to render in superscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

## TABLECAPTION Function

This function generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table.

### Syntax

```
HTF.TABLECAPTION(
  ccaption      IN      VARCHAR2,
  calign        in      VARCHAR2  DEFAULT NULL,
  cattributes   IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–82** TABLECAPTION Function Parameters

Parameter	Description
cctext	The text for the caption.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

## TABLECLOSE Function

This function generates the `</TABLE>` tag which marks the end of an HTML table. To define the beginning of an HTML table, use the [TABLEOPEN Function](#) on page 182-127.

### Syntax

```
HTF.TABLECLOSE  
    RETURN VARCHAR2;
```

### Examples

This function generates

```
</TABLE>
```

## TABLEDATA Function

This function generates the <TD> and </TD> tags which insert data into a cell of an HTML table.

### Syntax

```
HTF.TABLEDATA(
  cvalue      IN      VARCHAR2  DEFAULT NULL,
  calign      IN      VARCHAR2  DEFAULT NULL,
  cdp         IN      VARCHAR2  DEFAULT NULL,
  cnowrap     IN      VARCHAR2  DEFAULT NULL,
  crowspan    IN      VARCHAR2  DEFAULT NULL,
  ccolspan    IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–83 TABLEDATA Function Parameters**

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TD>
```

## TABLEHEADER Function

This function generates the <TH> and </TH> tags which insert a header cell in an HTML table. The <TH> tag is similar to the <TD> tag except that the text in this case the rows are usually rendered in bold type.

### Syntax

```
HTF.TABLEHEADER (
    cvalue      IN      VARCHAR2  DEFAULT NULL,
    calign      IN      VARCHAR2  DEFAULT NULL,
    cdp         IN      VARCHAR2  DEFAULT NULL,
    cnowrap     IN      VARCHAR2  DEFAULT NULL,
    crowspan    IN      VARCHAR2  DEFAULT NULL,
    ccolspan    IN      VARCHAR2  DEFAULT NULL,
    cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–84** TABLEHEADER Function Parameters

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
crispen	The value for the ROWSPAN attribute.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TH>
```

## TABLEOPEN Function

This function generates the <TABLE> tag which marks the beginning of an HTML table. To define the end of an HTML table, use the [TABLECLOSE Function](#).

### Syntax

```
HTF.TABLEOPEN(
  cborder      IN      VARCHAR2  DEFAULT NULL
  calign       IN      VARCHAR2  DEFAULT NULL,
  cnowrap     IN      VARCHAR2  DEFAULT NULL,
  cclear       IN      VARCHAR2  DEFAULT NULL
  cattributes  IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–85** TABLEOPEN Function Parameters

Parameter	Description
border	The value for the BORDER attribute.
calign	The value for the ALIGN attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

## TABLEROWCLOSE Function

This function generates the `</TR>` tag which marks the end of a new row in an HTML table. To mark the beginning of a new row, use the [TABLEROWOPEN Function](#).

### Syntax

```
HTF.TABLEROWCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</TABLE>
```



## TABLEROWOPEN Function

This function generates the <TR> tag which marks the beginning of a new row in an HTML table. To mark the end of a new row, use the [TABLEROWCLOSE Function](#).

### Syntax

```
HTF.TABLEROWOPEN(
  calign          IN          VARCHAR2  DEFAULT NULL,
  cvalign         IN          VARCHAR2  DEFAULT NULL,
  cdp             IN          VARCHAR2  DEFAULT NULL,
  nowrap          IN          VARCHAR2  DEFAULT NULL,
  cattributes     IN          VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–86** *TABLEROWOPEN Function Parameters*

Parameter	Description
calign	The value for the ALIGN attribute.
cvalign	The value for the VALIGN attribute.
cdp	The value for the DP attribute.
nowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```

## TELETYPE Function

This function generates the `<TT>` and `</TT>` tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font.

### Syntax

```
HTF.TELETYPE(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–87 TELETYPE Function Parameters**

Parameter	Description
<code>ctext</code>	The text to render in a fixed width typewriter font.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<TT cattributes>ctext</TT>
```

## TITLE Function

This function generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window.

### Syntax

```
HTF.TITLE(  
    ctitle          IN          VARCHAR2)  
    RETURN VARCHAR2;
```

### Parameters

**Table 182–88** *TITLE Function Parameters*

Parameter	Description
ctitle	The text to display in the titlebar of the browser window.

### Examples

This function generates

```
<TITLE>ctitle</TITLE>
```

## ULISTCLOSE Function

This function generates the `</UL>` tag which marks the end of an unordered list. An unordered list presents items with bullets. To mark the beginning of an unordered list, use the [ULISTOPEN Function](#). Add list items with [LISTITEM Function](#).

### Syntax

```
HTF.ULISTCLOSE  
RETURN VARCHAR2;
```

### Examples

This function generates

```
</UL>
```

## ULISTOPEN Function

This function generates the <UL> tag which marks the beginning of an unordered list. An unordered list presents items with bullets. To mark the end of an unordered list, use the [ULISTCLOSE Function](#). Add list items with [LISTITEM Function](#).

### Syntax

```
HTF.ULISTOPEN(
  cclear      IN      VARCHAR2  DEFAULT NULL,
  cwrap       IN      VARCHAR2  DEFAULT NULL,
  cdingbat    IN      VARCHAR2  DEFAULT NULL,
  csrc        IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 182–89** *ULISTOPEN Function Parameters*

Parameter	Description
cclear	The value for the CLEAR attribute.
cwrap	The value for the WRAP attribute.
cdingbat	The value for the DINGBAT attribute.
csrc	The value for the SRC attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

## UNDERLINE Function

This function generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline.

### Syntax

```
HTF.UNDERLINE(  
  ctext          IN          VARCHAR2,  
  cattributes   IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–90** *UNDERLINE Function Parameters*

Parameter	Description
<code>ctext</code>	The text to render with an underline.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<U cattributes>ctext</U>
```

## VARIABLE Function

This function generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.

### Syntax

```
HTF.VARIABLE(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 182–91** VARIABLE Function Parameters

Parameter	Description
ctext	The text to render in italics.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This function generates

```
<VAR cattributes>ctext</VAR>
```

## WBR Function

This function generates the <WBR> tag, which inserts a soft line break within a section of NOBR text.

### Syntax

```
HTF.WBR  
RETURN VARCHAR2;
```

### Examples

This function generates

```
<WBR>
```



The HTP (hypertext procedures) and HTF (hypertext functions) packages generate HTML tags. For example, the HTP.ANCHOR procedure generates the HTML anchor tag, <A>.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

This chapter contains the following topics:

- [Using HTP](#)
  - Operational Notes
  - Rules and Limits
  - Examples
- [Summary of Tags](#)
- [Summary of HTP Subprograms](#)

## Using HTP

- [Operational Notes](#)
- [Rules and Limits](#)
- [Examples](#)

## Operational Notes

For every HTP procedure that generates one or more HTML tags, there is a corresponding HTF function with identical parameters with the following exception:

- The [PRINTS Procedure](#) and the [PS Procedure](#) do not have HTF function equivalents. Use the [ESCAPE\\_SC Function](#) or the [ESCAPE\\_URL Function](#) if you need a string conversion function. Note that while there is a [ESCAPE\\_SC Procedure](#) that performs the same operation as the [PRINTS Procedure](#) and the [PS Procedure](#), there is no procedural equivalent for the [ESCAPE\\_URL Function](#).
- The [FORMAT\\_CELL Function](#) does not have an HTP equivalent. The function formats column values inside an HTML table using [TABLEDATA Function](#) which does have an HTP equivalent in the [TABLEDATA Procedure](#). The advantage of this using the [FORMAT\\_CELL Function](#) is that it allows for better control over the HTML tables.

The function versions do not directly generate output in your Web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls. To print the output of HTF functions, call the functions from within the HTP.PRINT procedure. It then prints its parameters to the generated Web page.

## Rules and Limits

If you use values of the LONG data type in procedures such as `HTP.PRINT`, `HTP.PRN`, `HTP.PRINTS`, `HTP.PA` or `OWA_UTIL.CELLSPRINT`, only the first 32 K of the LONG data is used. The LONG data is bound to a VARCHAR2 data type in the procedure.

## Examples

The following commands generate a simple HTML document:

```
CREATE OR REPLACE PROCEDURE hello AS
BEGIN
    HTP.HTMLOPEN;           -- generates <HTML>
    HTP.HEADOPEN;          -- generates <HEAD>
    HTP.TITLE('Hello');    -- generates <TITLE>Hello</TITLE>
    HTP.HEADCLOSE;        -- generates </HEAD>
    HTP.BODYOPEN;          -- generates <BODY>
    HTP.HEADER(1, 'Hello'); -- generates <H1>Hello</H1>
    HTP.BODYCLOSE;         -- generates </BODY>
    HTP.HTMLCLOSE;         -- generates </HTML>
END;
```

## Summary of Tags

### HTML, HEAD, and BODY Tags

[HTMLOPEN Procedure](#), [HTMLCLOSE Procedure](#) - generate <HTML> and </HTML>

[HEADOPEN Procedure](#), [HEADCLOSE Procedure](#) - generate <HEAD> and </HEAD>

[BODYOPEN Procedure](#), [BODYCLOSE Procedure](#) - generate <BODY> and </BODY>

### Comment Tag

[COMMENT Procedure](#) - generates <!-- and -->

### Tags in the <HEAD> Area

[BASE Procedure](#) - generates <BASE>

[LINKREL Procedure](#) - generates <LINK> with the REL attribute

[LINKREV Procedure](#) - generates <LINK> with the REV attribute

[TITLE Procedure](#) - generates <TITLE>

[META Procedure](#) - generates <META>

[SCRIPT Procedure](#) - generates <SCRIPT>

[STYLE Procedure](#) - generates <STYLE>

[ISINDEX Procedure](#) - generates <ISINDEX>

### Applet Tags

[APPLETOPEN Procedure](#), [APPLETCLOSE Procedure](#) - generate <APPLET> and </APPLET>

[PARAM Procedure](#) - generates <PARAM>

### List Tags

[OLISTOPEN Procedure](#), [OLISTCLOSE Procedure](#) - generate <OL> and </OL>

[ULISTOPEN Procedure](#), [ULISTCLOSE Procedure](#) - generate <UL> and </UL>

[DLISTOPEN Procedure](#), [DLISTCLOSE Procedure](#) - generate <DL> and </DL>

[DLISTTERM Procedure](#) - generates <DT>

[DLISTDEF Procedure](#) - generates <DD>

[DIRLISTOPEN Procedure](#), [DIRLISTCLOSE Procedure](#) - generate <DIR> and </DIR>

[LISTHEADER Procedure](#) - generates <LH>

[LISTINGOPEN Procedure](#), [LISTINGCLOSE Procedure](#) - generate <LISTING> and </LISTING>

[MENULISTOPEN Procedure](#) - generate <MENU> and </MENU>

[LISTITEM Procedure](#) - generates <LI>

### Form Tags

[FORMOPEN Procedure](#), [FORMCLOSE Procedure](#) - generate <FORM> and </FORM>

[FORMCHECKBOX Procedure](#) - generates `<INPUT TYPE="CHECKBOX">`  
[FORMHIDDEN Procedure](#) - generates `<INPUT TYPE="HIDDEN">`  
[FORMIMAGE Procedure](#) - generates `<INPUT TYPE="IMAGE">`  
[FORMPASSWORD Procedure](#) - generates `<INPUT TYPE="PASSWORD">`  
[FORMRADIO Procedure](#) - generates `<INPUT TYPE="RADIO">`  
[FORMSELECTOPEN Procedure](#), [FORMSELECTCLOSE Procedure](#) - generate `<SELECT>` and `</SELECT>`  
[FORMSELECTOPTION Procedure](#) - generates `<OPTION>`  
[FORMTEXT Procedure](#) - generates `<INPUT TYPE="TEXT">`  
[FORMTEXTAREA Procedure](#) - generate `<TEXTAREA>`  
[FORMTEXTAREAOPEN Procedure](#), [FORMTEXTAREACLOSE Procedure](#) - generate `<TEXTAREA>` and `</TEXTAREA>`  
[FORMRESET Procedure](#) - generates `<INPUT TYPE="RESET">`  
[FORMSUBMIT Procedure](#) - generates `<INPUT TYPE="SUBMIT">`

## Table Tags

[TABLEOPEN Procedure](#), [TABLECLOSE Procedure](#) - generate `<TABLE>` and `</TABLE>`  
[TABLECAPTION Procedure](#) - generates `<CAPTION>`  
[TABLEROWOPEN Procedure](#), [TABLEROWCLOSE Procedure](#) - generate `<TR>` and `</TR>`  
[TABLEHEADER Procedure](#) - generates `<TH>`  
[TABLEDATA Procedure](#) - generates `<TD>`

## IMG, HR, and A Tags

[HR Procedure](#), [LINE Procedure](#) - generate `<HR>`  
[IMG Procedure](#), [IMG2 Procedure](#) - generate `<IMG>`  
[ANCHOR Procedure](#), [ANCHOR2 Procedure](#) - generate `<A>`  
[MAPOPEN Procedure](#), [MAPCLOSE Procedure](#) - generate `<MAP>` and `</MAP>`

## Paragraph Formatting Tags

[HEADER Procedure](#) - generates heading tags (`<H1>` to `<H6>`)  
[PARA Procedure](#), [PARAGRAPH Procedure](#) - generate `<P>`  
[PRN Procedures](#), [PRINT Procedures](#) - generate any text that is passed in  
[PRINTS Procedure](#), [PS Procedure](#) - generate any text that is passed in; special characters in HTML are escaped  
[PREOPEN Procedure](#), [PRECLOSE Procedure](#) - generate `<PRE>` and `</PRE>`  
[BLOCKQUOTEOPEN Procedure](#), [BLOCKQUOTECLOSE Procedure](#) - generate `<BLOCKQUOTE>` and `</BLOCKQUOTE>`  
[DIV Procedure](#) - generates `<DIV>`  
[NL Procedure](#), [BR Procedure](#) - generate `<BR>`

[NOBR Procedure](#) - generates <NOBR>  
[WBR Procedure](#) - generates <WBR>  
[PLAINTEXT Procedure](#) - generates <PLAINTEXT>  
[ADDRESS Procedure](#) - generates <ADDRESS>  
[MAILTO Procedure](#) - generates <A> with the MAILTO attribute  
[AREA Procedure](#) - generates <AREA>  
[BGSOUND Procedure](#) - generates <BGSOUND>

## Character Formatting Tags

[BASEFONT Procedure](#) - generates <BASEFONT>  
[BIG Procedure](#) - generates <BIG>  
[BOLD Procedure](#) - generates <B>  
[CENTER Procedure](#) - generates <CENTER> and </CENTER>  
[CENTEROPEN Procedure](#), [CENTERCLOSE Procedure](#) - generate <CENTER> and </CENTER>  
[CITE Procedure](#) - generates <CITE>  
[CODE Procedure](#) - generates <CODE>  
[DFN Procedure](#) - generates <DFN>  
[EM Procedure](#), [EMPHASIS Procedure](#) - generate <EM>  
[FONTOPEN Procedure](#), [FONTCLOSE Procedure](#) - generate <FONT> and </FONT>  
[ITALIC Procedure](#) - generates <I>  
[KBD Procedure](#), [KEYBOARD Procedure](#) - generate <KBD> and </KBD>  
[S Procedure](#) - generates <S>  
[SAMPLE Procedure](#) - generates <SAMP>  
[SMALL Procedure](#) - generates <SMALL>  
[STRIKE Procedure](#) - generates <STRIKE>  
[STRONG Procedure](#) - generates <STRONG>  
[SUB Procedure](#) - generates <SUB>  
[SUP Procedure](#) - generates <SUP>  
[TELETYPE Procedure](#) - generates <TT>  
[UNDERLINE Procedure](#) - generates <U>  
[VARIABLE Procedure](#) - generates <VAR>

## Frame Tags

[FRAME Procedure](#) - generates <FRAME>  
[FRAMESETOPEN Procedure](#), [FRAMESETCLOSE Procedure](#) - generate <FRAMESET> and </FRAMESET>  
[NOFRAMESOPEN Procedure](#), [NOFRAMESCLOSE Procedure](#) - generate <NOFRAMES> and </NOFRAMES>



---

## Summary of HTP Subprograms

**Table 183–1 HTP Package Subprograms**

Subprogram	Description
<a href="#">ADDRESS Procedure</a> on page 183-15	Generate s the <code>&lt;ADDRESS&gt;</code> and <code>&lt;/ADDRESS&gt;</code> tags which specify the address, author and signature of a document
<a href="#">ANCHOR Procedure</a> on page 183-16	Generate s the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags which specify the source or destination of a hypertext link
<a href="#">ANCHOR2 Procedure</a> on page 183-17	Generates the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags which specify the source or destination of a hypertext link
<a href="#">APPLETCLOSE Procedure</a> on page 183-18	Closes the applet invocation with the <code>&lt;/APPLET&gt;</code> tag
<a href="#">APPLETOPEN Procedure</a> on page 183-19	Generates the <code>&lt;APPLET&gt;</code> tag which begins the invocation of a Java applet
<a href="#">AREA Procedure</a> on page 183-20	Generates the <code>&lt;AREA&gt;</code> tag, which defines a client-side image map
<a href="#">BASE Procedure</a> on page 183-21	Generates the <code>&lt;BASE&gt;</code> tag which records the URL of the document
<a href="#">BASEFONT Procedure</a> on page 183-22	Generates the <code>&lt;BASEFONT&gt;</code> tag which specifies the base font size for a Web page
<a href="#">BGSOUND Procedure</a> on page 183-23	Generates the <code>&lt;BGSOUND&gt;</code> tag which includes audio for a Web page
<a href="#">BIG Procedure</a> on page 183-24	Generates the <code>&lt;BIG&gt;</code> and <code>&lt;/BIG&gt;</code> tags which direct the browser to render the text in a bigger font
<a href="#">BLOCKQUOTECLOSE Procedure</a> on page 183-25	Generates the <code>&lt;/BLOCKQUOTE&gt;</code> tag which mark the end of a section of quoted text
<a href="#">BLOCKQUOTEOPEN Procedure</a> on page 183-26	Generates the <code>&lt;BLOCKQUOTE&gt;</code> tag, which marks the beginning of a section of quoted text
<a href="#">BODYCLOSE Procedure</a> on page 183-27	Generates the <code>&lt;/BODY&gt;</code> tag which marks the end of a body section of an HTML document
<a href="#">BODYOPEN Procedure</a> on page 183-28	Generates the <code>&lt;BODY&gt;</code> tag which marks the beginning of the body section of an HTML document
<a href="#">BOLD Procedure</a> on page 183-29	Generates the <code>&lt;B&gt;</code> and <code>&lt;/B&gt;</code> tags which direct the browser to display the text in boldface
<a href="#">BR Procedure</a> on page 183-30	Generates the <code>&lt;BR&gt;</code> tag which begins a new line of text
<a href="#">CENTER Procedure</a> on page 183-31	Generates the <code>&lt;CENTER&gt;</code> and <code>&lt;/CENTER&gt;</code> tags which center a section of text within a Web page
<a href="#">CENTERCLOSE Procedure</a> on page 183-32	Generates the <code>&lt;/CENTER&gt;</code> tag which marks the end of a section of text to center

**Table 183–1 (Cont.) HTP Package Subprograms**

Subprogram	Description
<a href="#">CENTEROPEN Procedure</a> on page 183-33	Generates the <CENTER> tag which mark the beginning of a section of text to center
<a href="#">CITE Procedure</a> on page 183-34	Generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation
<a href="#">CODE Procedure</a> on page 183-35	Generates the <CODE> and </CODE> tags which direct the browser to render the text in monospace font or however "code" is defined stylistically
<a href="#">COMMENT Procedure</a> on page 183-36	Generates This procedure generates the comment tags  <!-- ctext -->
<a href="#">DFN Procedure</a> on page 183-37	Generates the <DFN> and </DFN> tags which direct the browser to mark the text as italics or however "definition" is defined stylistically
<a href="#">DIRLISTCLOSE Procedure</a> on page 183-38	Generates the </DIR> tag which ends a directory list section
<a href="#">DIRLISTOPEN Procedure</a> on page 183-39	Generates the <DIR> which starts a directory list section
<a href="#">DIV Procedure</a> on page 183-40	Generates the <DIV> tag which creates document divisions
<a href="#">DLISTCLOSE Procedure</a> on page 183-41	Generates the </DL> tag which ends a definition list
<a href="#">DLISTDEF Procedure</a> on page 183-42	Generates the <DD> tag, which inserts definitions of terms
<a href="#">DLISTOPEN Procedure</a> on page 183-43	Generates the <DL> tag which starts a definition list
<a href="#">DLISTTERM Procedure</a> on page 183-44	Generates the <DT> tag which defines a term in a definition list <DL>
<a href="#">EM Procedure</a> on page 183-45	Generates the <EM> and </EM> tags, which define text to be emphasized
<a href="#">EMPHASIS Procedure</a> on page 183-46	Generates the <EM> and </EM> tags, which define text to be emphasized
<a href="#">ESCAPE_SC Procedure</a> on page 183-47	Replaces characters that have special meaning in HTML with their escape sequences
<a href="#">FONTCLOSE Procedure</a> on page 183-48	Generates the </FONT> tag which marks the end of a section of text with the specified font characteristics
<a href="#">FONTOPEN Procedure</a> on page 183-49	Generates the <FONT> which marks the beginning of section of text with the specified font characteristics
<a href="#">FORMCHECKBOX Procedure</a> on page 183-50	Generates the <INPUT> tag with TYPE="checkbox" which inserts a checkbox element in a form
<a href="#">FORMCLOSE Procedure</a> on page 183-51	Generates the </FORM> tag which marks the end of a form section in an HTML document
<a href="#">FORMOPEN Procedure</a> on page 183-52	Generates the <FORM> tag which marks the beginning of a form section in an HTML document
<a href="#">FORMFILE Procedure</a> on page 183-53	Generates the <INPUT> tag with TYPE="file" which inserts a file form element, and is used for file uploading for a given page
<a href="#">FORMHIDDEN Procedure</a> on page 183-54	Generates the <INPUT> tag with TYPE="hidden" which inserts a hidden form element

**Table 183-1 (Cont.) HTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">FORMIMAGE Procedure</a> on page 183-55	Generates the <INPUT> tag with TYPE="image" which creates an image field that the user clicks to submit the form immediately
<a href="#">FORMPASSWORD Procedure</a> on page 183-56	Generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field
<a href="#">FORMRADIO Procedure</a> on page 183-57	Generates the <INPUT> tag with TYPE="radio", which creates a radio button on the HTML form
<a href="#">FORMRESET Procedure</a> on page 183-58	Generates the <INPUT> tag with TYPE="reset" which creates a button that, when selected, resets the form fields to their initial values
<a href="#">FORMSELECTCLOSE Procedure</a> on page 183-59	Generates the </SELECT> tag which marks the end of a Select form element
<a href="#">FORMSELECTOPEN Procedure</a> on page 183-60	Generates the </SELECT> tag which marks the beginning of a Select form element
<a href="#">FORMSELETOPTION Procedure</a> on page 183-61	Generates the <OPTION> tag which represents one choice in a Select element
<a href="#">FORMSUBMIT Procedure</a> on page 183-62	Generates the <INPUT> tag with TYPE="submit" which creates a button that, when clicked, submits the form
<a href="#">FORMTEXT Procedure</a> on page 183-63	Generates the <INPUT> tag with TYPE="text", which creates a field for a single line of text
<a href="#">FORMTEXTAREA Procedure</a> on page 183-64	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area
<a href="#">FORMTEXTAREA2 Procedure</a> on page 183-65	Generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area with the ability to specify a wrap style
<a href="#">FORMTEXTAREACLOSE Procedure</a> on page 183-66	Generates the </TEXTAREA> tag which ends a text area form element
<a href="#">FORMTEXTAREAOPEN Procedure</a> on page 183-67	Generates the <TEXTAREA> which marks the beginning of a text area form element
<a href="#">FORMTEXTAREAOPEN2 Procedure</a> on page 183-68	Generates the <TEXTAREA> which marks the beginning of a text area form element with the ability to specify a wrap style
<a href="#">FRAME Procedure</a> on page 183-69	Generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag
<a href="#">FRAMESETCLOSE Procedure</a> on page 183-70	Generates the </FRAMESET> tag which ends a frameset section
<a href="#">FRAMESETOPEN Procedure</a> on page 183-71	Generates the </FRAMESET> tag which begins a frameset section
<a href="#">HEADCLOSE Procedure</a> on page 183-72	Generates the </HEAD> tag which marks the end of an HTML document head section
<a href="#">HEADER Procedure</a> on page 183-73	Generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>)
<a href="#">HEADOPEN Procedure</a> on page 183-74	Generates the <HEAD> tag which marks the beginning of the HTML document head section
<a href="#">HR Procedure</a> on page 183-75	Generates the <HR> tag, which generates a line in the HTML document

**Table 183–1 (Cont.) HTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">HTMLCLOSE Procedure</a> on page 183-76	Generates the <code>&lt;/HTML&gt;</code> tag which marks the end of an HTML document
<a href="#">HTMLOPEN Procedure</a> on page 183-77	Generates the <code>&lt;HTML&gt;</code> tag which marks the beginning of an HTML document
<a href="#">IMG Procedure</a> on page 183-78	Generates the <code>&lt;IMG&gt;</code> tag which directs the browser to load an image onto the HTML page
<a href="#">IMG2 Procedure</a> on page 183-79	Generates the <code>&lt;IMG&gt;</code> tag which directs the browser to load an image onto the HTML page with the option of specifying values for the <code>USEMAP</code> attribute
<a href="#">ISINDEX Procedure</a> on page 183-80	Creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program
<a href="#">ITALIC Procedure</a> on page 183-81	Generates the <code>&lt;I&gt;</code> and <code>&lt;/I&gt;</code> tags which direct the browser to render the text in italics
<a href="#">KBD Procedure</a> on page 183-82	Generates the <code>&lt;KBD&gt;</code> and <code>&lt;/KBD&gt;</code> tags which direct the browser to render the text in monospace font
<a href="#">KEYBOARD Procedure</a> on page 183-83	Generates the <code>&lt;KBD&gt;</code> and <code>&lt;/KBD&gt;</code> tags, which direct the browser to render the text in monospace font
<a href="#">LINE Procedure</a> on page 183-84	Generates the <code>&lt;HR&gt;</code> tag, which generates a line in the HTML document
<a href="#">LINKREL Procedure</a> on page 183-85	Generates the <code>&lt;LINK&gt;</code> tag with the <code>REL</code> attribute which delineates the relationship described by the hypertext link from the anchor to the target
<a href="#">LINKREV Procedure</a> on page 183-86	Generates the <code>&lt;LINK&gt;</code> tag with the <code>REV</code> attribute which delineates the relationship described by the hypertext link from the target to the anchor
<a href="#">LISTHEADER Procedure</a> on page 183-87	Generates the <code>&lt;LH&gt;</code> and <code>&lt;/LH&gt;</code> tags which print an HTML tag at the beginning of the list
<a href="#">LISTINGCLOSE Procedure</a> on page 183-88	Generates the <code>&lt;/LISTING&gt;</code> tags which marks the end of a section of fixed-width text in the body of an HTML page
<a href="#">LISTINGOPEN Procedure</a> on page 183-89	Generates the <code>&lt;LISTING&gt;</code> tag which marks the beginning of a section of fixed-width text in the body of an HTML page
<a href="#">LISTITEM Procedure</a> on page 183-90	Generates the <code>&lt;LI&gt;</code> tag, which indicates a list item
<a href="#">MAILTO Procedure</a> on page 183-91	Generates the <code>&lt;A&gt;</code> tag with the <code>HREF</code> set to 'mailto' prepended to the mail address argument
<a href="#">MAPCLOSE Procedure</a> on page 183-92	Generates the <code>&lt;/MAP&gt;</code> tag which marks the end of a set of regions in a client-side image map
<a href="#">MAPOPEN Procedure</a> on page 183-93	Generates the <code>&lt;MAP&gt;</code> tag which mark the beginning of a set of regions in a client-side image map
<a href="#">MENULISTCLOSE Procedure</a> on page 183-94	Generates the <code>&lt;/MENU&gt;</code> tag which ends a list that presents one line for each item
<a href="#">MENULISTOPEN Procedure</a> on page 183-95	Generates the <code>&lt;MENU&gt;</code> tag which create a list that presents one line for each item

**Table 183-1 (Cont.) HTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">META Procedure</a> on page 183-96	Generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers
<a href="#">NL Procedure</a> on page 183-97	Generates the   tag which begins a new line of text
<a href="#">NOBR Procedure</a> on page 183-98	Generates the <NOBR> and </NOBR> tags which turn off line-breaking in a section of text
<a href="#">NOFRAMESCLOSE Procedure</a> on page 183-99	Generates the </NOFRAMES> tag which marks the end of a no-frames section
<a href="#">NOFRAMESOPEN Procedure</a> on page 183-100	Generates the <NOFRAMES> tag which mark the beginning of a no-frames section
<a href="#">OLISTCLOSE Procedure</a> on page 183-101	Generates the </OL> tag which defines the end of an ordered list
<a href="#">OLISTOPEN Procedure</a> on page 183-102	Generates the <OL> tag which marks the beginning of an ordered list
<a href="#">PARA Procedure</a> on page 183-103	Generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph
<a href="#">PARAGRAPH Procedure</a> on page 183-104	Adds attributes to the <P> tag
<a href="#">PARAM Procedure</a> on page 183-105	Generates the <PARAM> tag which specifies parameter values for Java applets
<a href="#">PLAINTEXT Procedure</a> on page 183-106	Generates the <PLAINTEXT> and </PLAINTEXT> tags which direct the browser to render the text they surround in fixed-width type
<a href="#">PRECLOSE Procedure</a> on page 183-107	Generates the </PRE> tag which marks the end of a section of preformatted text in the body of the HTML page
<a href="#">PREOPEN Procedure</a> on page 183-108	Generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page
<a href="#">PRINT Procedures</a> on page 183-109	Generates the specified parameter as a string terminated with the \n newline character
<a href="#">PRINTS Procedure</a> on page 183-110	Generates a string and replaces the following characters with the corresponding escape sequence
<a href="#">PRN Procedures</a> on page 183-111	Generates the specified parameter as a string
<a href="#">PS Procedure</a> on page 183-112	Generates a string and replaces the following characters with the corresponding escape sequence.
<a href="#">S Procedure</a> on page 183-113	Generates the <S> and </S> tags which direct the browser to render the text they surround in strikethrough type
<a href="#">SAMPLE Procedure</a> on page 183-114	Generates the <SAMP> and </SAMP> tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically
<a href="#">SCRIPT Procedure</a> on page 183-115	Generates the <SCRIPT> and </SCRIPT> tags which contain a script written in languages such as JavaScript and VBscript
<a href="#">SMALL Procedure</a> on page 183-116	Generates the <SMALL> and </SMALL> tags, which direct the browser to render the text they surround using a small font

**Table 183–1 (Cont.) HTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">STRIKE Procedure</a> on page 183-117	Generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type
<a href="#">STRONG Procedure</a> on page 183-118	Generates the <STRONG> and </STRONG> tags which direct the browser to render the text they surround in bold or however "strong" is defined stylistically
<a href="#">STYLE Procedure</a> on page 183-119	Generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page
<a href="#">SUB Procedure</a> on page 183-120	Generates the <SUB> and </SUB> tags which direct the browser to render the text they surround as subscript
<a href="#">SUP Procedure</a> on page 183-121	Generates the <SUP> and </SUP> tags which direct the browser to render the text they surround as superscript
<a href="#">TABLECAPTION Procedure</a> on page 183-122	Generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table
<a href="#">TABLECLOSE Procedure</a> on page 183-123	Generates the </TABLE> tag which marks the end of an HTML table
<a href="#">TABLEDATA Procedure</a> on page 183-124	Generates the <TD> and </TD> tags which insert data into a cell of an HTML table
<a href="#">TABLEHEADER Procedure</a> on page 183-125	Generates the <TH> and </TH> tags which insert a header cell in an HTML table.
<a href="#">TABLEOPEN Procedure</a> on page 183-126	Generates the <TABLE> tag which marks the beginning of an HTML table
<a href="#">TABLEROWCLOSE Procedure</a> on page 183-127	Generates the </TR> tag which marks the end of a new row in an HTML table
<a href="#">TABLEROWOPEN Procedure</a> on page 183-128	Generates the <TR> tag which marks the beginning of a new row in an HTML table
<a href="#">TELETYPE Procedure</a> on page 183-129	Generates the <TT> and </TT> tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font
<a href="#">TITLE Procedure</a> on page 183-130	Generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window
<a href="#">ULISTCLOSE Procedure</a> on page 183-131	Generates the </UL> tag which marks the end of an unordered list
<a href="#">ULISTOPEN Procedure</a> on page 183-132	Generates the <UL> tag which marks the beginning of an unordered list
<a href="#">UNDERLINE Procedure</a> on page 183-133	Generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline
<a href="#">VARIABLE Procedure</a> on page 183-134	Generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.
<a href="#">WBR Procedure</a> on page 183-135	Generates the <WBR> tag, which inserts a soft line break within a section of NOBR text

## ADDRESS Procedure

This procedure generates the <ADDRESS> and </ADDRESS> tags which specify the address, author and signature of a document.

### Syntax

```
HTP.ADDRESS (
  cvalue      IN      VARCHAR2
  cnowrap     IN      VARCHAR2  DEFAULT NULL
  cclear      IN      VARCHAR2  DEFAULT NULL
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–2 ADDRESS Procedure Parameters**

Parameter	Description
cvalue	The string that goes between the <ADDRESS> and </ADDRESS> tags.
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is included in the tag
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag

### Examples

This procedure generates

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```

## ANCHOR Procedure

This procedure and the [ANCHOR2 Procedure](#) procedures generate the `<A>` and `</A>` HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that the [ANCHOR2 Procedure](#) provides a target and therefore can be used for a frame.

### Syntax

```
HTP.ANCHOR (  
    curl          IN          VARCHAR2,  
    ctext         IN          VARCHAR2,  
    cname         IN          VARCHAR2  DEFAULT NULL,  
    cattributes  IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–3 ANCHOR Procedure Parameters**

Parameter	Description
<code>curl</code>	The value for the <code>HREF</code> attribute.
<code>ctext</code>	The string that goes between the <code>&lt;A&gt;</code> and <code>&lt;/A&gt;</code> tags.
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

### Usage Notes

This tag accepts several attributes, but either `HREF` or `NAME` is required. `HREF` specifies to where to link. `NAME` allows this tag to be a target of a hypertext link.



## ANCHOR2 Procedure

This procedure and the [ANCHOR Procedure](#) generate the <A> and </A> HTML tags which specify the source or destination of a hypertext link. The difference between these subprograms is that this procedure provides a target and therefore can be used for a frame.

### Syntax

```
HTP.ANCHOR2 (
    curl          IN          VARCHAR2,
    ctext         IN          VARCHAR2,
    cname         IN          VARCHAR2  DEFAULT NULL,
    ctarget       IN          varchar2  DEFAULT NULL,
    cattributes   IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–4 ANCHOR2 Procedure Parameters**

Parameter	Description
curl	The value for the HREF attribute.
ctext	The string that goes between the <A> and </A> tags.
cname	The value for the NAME attribute
ctarget	The value for the TARGET attribute.
cattributes	The other attributes to be included as-is in the tag

### Examples

This procedure generates

```
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

## APPLETCLOSE Procedure

This procedure closes the applet invocation with the `</APPLET>` tag. You must first invoke the a Java applet using [APPLETOPEN Procedure](#).

### Syntax

```
HTP.APPLETCLOSE;
```

## APPLETOPEN Procedure

This procedure generates the <APPLET> tag which begins the invocation of a Java applet. You close the applet invocation with [APPLETCLOSE Procedure](#) which generates the </APPLET> tag.

### Syntax

```
HTP.APPLETOPEN (
  ccode          IN          VARCHAR2,
  cheight        IN          NUMBER,
  cwidth         IN          NUMBER,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–5** *APPLETOPEN Procedure Parameters*

Parameter	Description
<code>ccode</code>	The the value for the CODE attribute which specifies the name of the applet class.
<code>cheight</code>	The value for the HEIGHT attribute.
<code>cwidth</code>	The value for the WIDTH attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<APPLET CODE=ccode HEIGHT=cheight WIDTH=cwidth cattributes>
```

so that, for example,

```
HTP.appletopen('testclass.class', 100, 200, 'CODEBASE="/ows-applets"')
```

generates

```
<APPLET CODE="testclass.class" height=100 width=200 CODEBASE="/ows-applets">
```

### Usage Notes

- Specify parameters to the Java applet using the [PARAM Procedure](#) procedure.
- Use the `cattributes` parameter to specify the CODEBASE attribute since the PL/SQL cartridge does not know where to find the class files. The CODEBASE attribute specifies the virtual path containing the class files.

## AREA Procedure

This procedure generates the <AREA> tag, which defines a client-side image map. The <AREA> tag defines areas within the image and destinations for the areas.

### Syntax

```
HTP.AREA (
  ccoords      IN      VARCHAR2
  cshape       IN      VARCHAR2  DEFAULT NULL,
  chref        IN      VARCHAR2  DEFAULT NULL,
  cnohref      IN      VARCHAR2  DEFAULT NULL,
  ctarget      IN      VARCHAR2  DEFAULT NULL,
  cattributes  IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–6 AREA Procedure Parameters**

Parameter	Description
<code>ccords</code>	The the value for the COORDS attribute.
<code>cshape</code>	The value for the SHAPE attribute.
<code>chref</code>	The value for the HREF attribute.
<code>cnohref</code>	If the value for this parameter is not NULL, the NOHREF attribute is added to the tag.
<code>ctarget</code>	The value for the TARGET attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<AREA COORDS="ccords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget"
cattributes>
```

## BASE Procedure

This procedure generates the <BASE> tag which records the URL of the document.

### Syntax

```
HTP.BASE (
  ctarget      IN      VARCHAR2  DEFAULT NULL,
  cattributes  IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–7** *BASE Procedure Parameters*

Parameter	Description
ctarget	The value for the TARGET attribute which establishes a window name to which all links in this document are targeted.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BASE HREF="<current URL>" TARGET="ctarget" cattributes>
```

## BASEFONT Procedure

This procedure generates the `<BASEFONT>` tag which specifies the base font size for a Web page.

### Syntax

```
HTP.BASEFONT (  
    nsize    IN    INTEGER);
```

### Parameters

**Table 183–8** *BASEFONT Procedure Parameters*

Parameter	Description
<code>nsize</code>	The value for the <code>SIZE</code> attribute.

### Examples

This procedure generates

```
<BASEFONT SIZE="nsize">
```

## BGSOUND Procedure

This procedure generates the <BGSOUND> tag which includes audio for a Web page.

### Syntax

```
HTP.BGSOUND (
  csrc          IN          VARCHAR2,
  cloop        IN          VARCHAR2  DEFAULT NULL,
  cattributes  IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–9 BGSOUND Procedure Parameters**

Parameter	Description
csrc	The value for the SRC attribute.
clloop	The value for the LOOP attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BGSOUND SRC="csrc" LOOP="clloop" cattributes>
```

## BIG Procedure

This procedure generates the <BIG> and </BIG> tags which direct the browser to render the text in a bigger font.

### Syntax

```
HTP.BIG (  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–10** *BIG Procedure Parameters*

Parameter	Description
ctext	The the text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BIG cattributes>ctext</BIG>
```



## BLOCKQUOTECLOSE Procedure

This procedure generates the `</BLOCKQUOTE>` tag which mark the end of a section of quoted text. You mark the beginning of a section of text by means of the [BLOCKQUOTEOPEN Procedure](#).

### Syntax

```
HTP.BLOCKQUOTECLOSE;
```

### Examples

This procedure generates

```
</BLOCKQUOTE>
```

## BLOCKQUOTEOPEN Procedure

This procedure generates the <BLOCKQUOTE> tag, which marks the beginning of a section of quoted text. You mark the end of a section of text by means of the [BLOCKQUOTECLOSE Procedure](#).

### Syntax

```
HTP.BLOCKQUOTEOPEN (  
    cnowrap          IN          VARCHAR2  DEFAULT NULL,  
    cclear           IN          VARCHAR2  DEFAULT NULL,  
    cattributes      IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–11** *BLOCKQUOTEOPEN Procedure Parameters*

Parameter	Description
cnowrap	If the value for this parameter is not NULL, the NOWRAP attribute is added to the tag.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

## BODYCLOSE Procedure

This procedure generates the `</BODY>` tag which marks the end of a body section of an HTML document. You mark the beginning of a body section by means of the [BODYOPEN Procedure](#).

### Syntax

```
HTP.BODYCLOSE;
```

### Examples

This procedure generates

```
</BODY>
```

## BODYOPEN Procedure

This procedure generates the <BODY> tag which marks the beginning of the body section of an HTML document. You mark the end of a body section by means of the [BODYCLOSE Procedure](#).

### Syntax

```
HTP.BODYOPEN (  
    cbackground IN VARCHAR2 DEFAULT NULL,  
    cattributes IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 183–12 BODYOPEN Procedure Parameters**

Parameter	Description
<code>cbackground</code>	The value for the <code>BACKGROUND</code> attribute which specifies a graphic file to use for the background of the document.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BODY background="cbackground" cattributes>
```

so that

```
HTP.BODYOPEN('/img/background.gif');
```

generates:

```
<BODY background="/img/background.gif">
```

## BOLD Procedure

This procedure generates the <B> and </B> tags which direct the browser to display the text in boldface.

### Syntax

```
HTP.BOLD (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–13** *BOLD Procedure Parameters*

Parameter	Description
ctext	The text that goes between the tags.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<B cattributes>ctext</B>
```

## BR Procedure

This procedure generates the <BR> tag which begins a new line of text. It performs the same operation as the [NL Procedure](#).

### Syntax

```
HTP.BR(  
    cclear          IN          VARCHAR2  DEFAULT NULL,  
    cattributes     IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–14 BR Procedure Parameters**

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BR CLEAR="cclear" cattributes>
```

## CENTER Procedure

This procedure generates the <CENTER> and </CENTER> tags which center a section of text within a Web page.

### Syntax

```
HTP.CENTER (  
    ctext          IN          VARCHAR2);
```

### Parameters

**Table 183–15** *CENTER Parameters*

Parameter	Description
<code>ctext</code>	The text that goes between the tags.

### Examples

This procedure generates

```
<CENTER>ctext</CENTER>
```

## CENTERCLOSE Procedure

This procedure generates the `</CENTER>` tag which marks the end of a section of text to center. You mark the beginning of a section of text to center by means of the [CENTEROPEN Procedure](#).

### Syntax

```
HTP.CENTERCLOSE;
```

### Examples

This procedure generates

```
</CENTER>
```



## CENTEROPEN Procedure

This procedure generates the <CENTER> tag which mark the beginning of a section of text to center. You mark the beginning of a of a section of text to center by means of the [CENTERCLOSE Procedure](#).

### Syntax

```
HTP.CENTEROPEN;
```

### Examples

This procedure generates

```
<CENTER>
```

## CITE Procedure

This procedure generates the <CITE> and </CITE> tags which direct the browser to render the text as a citation.

### Syntax

```
HTP.CITE (  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–16 CITE Procedure Parameters**

Parameter	Description
ctext	The text to render as citation.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<CITE cattributes>ctext</CITE>
```

## CODE Procedure

This procedure generates the `<CODE>` and `</CODE>` tags which direct the browser to render the text in monospace font or however "code" is defined stylistically.

### Syntax

```
HTP.CODE (  
  ctext      IN      VARCHAR2,  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–17** *CODE Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to render as code.
<code>cattributes</code>	The other attributes to be included as-is in the tag

### Examples

This procedure generates

```
<CODE cattributes>ctext</CODE>
```

## COMMENT Procedure

This procedure generates the comment tags.

### Syntax

```
HTP.COMMENT (  
    ctext          IN          VARCHAR2);
```

### Parameters

**Table 183–18** *COMMENT Procedure Parameters*

Parameter	Description
<code>ctext</code>	The comment.

### Examples

This procedure generates

```
<!-- ctext -->
```

## DFN Procedure

This procedure generates the <DFN> and </DFN> tags which direct the browser to mark the text in italics or however "definition" is described stylistically.

### Syntax

```
HTP.DFN (  
    ctext          IN          VARCHAR2);
```

### Parameters

**Table 183–19** *DFN Procedure Parameters*

Parameter	Description
ctext	The text to render in italics.

### Examples

This procedure generates

```
<DFN>ctext</DFN>
```

## DIRLISTCLOSE Procedure

This procedure generates the `</DIR>` tag which ends a directory list section. You start a directory list section with the [DIRLISTOPEN Procedure](#).

### Syntax

```
HTP.DIRLISTCLOSE;
```

### Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the `<LI>` tag directly or invoke the [LISTITEM Procedure](#) so that the `<LI>` tag appears directly after the `</DIR>` tag to define the items as a list.

### Examples

This procedure generates

```
</DIR>
```

## DIRLISTOPEN Procedure

This procedure generates the <DIR> which starts a directory list section. You end a directory list section with the [DIRLISTCLOSE Procedure](#).

### Syntax

```
HTP.DIRLISTOPEN;
```

### Usage Notes

A directory list presents a list of items that contains up to 20 characters. Items in this list are typically arranged in columns, 24 characters wide. Insert the <LI> tag directly or invoke the [LISTITEM Procedure](#) so that the <LI> tag appears directly after the </DIR> tag to define the items as a list.

### Examples

This procedure generates

```
<DIR>
```

## DIV Procedure

This procedure generates the <DIV> tag which creates document divisions.

### Syntax

```
HTP.DIV (  
    calign          IN          VARCHAR2  DEFAULT NULL,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–20 DIV Procedure Parameters**

Parameter	Description
<code>calign</code>	The value for the ALIGN attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<DIV ALIGN="calign" cattributes>
```



## DLISTCLOSE Procedure

This procedure generates the `</DL>` tag which ends a definition list. You start a definition list by means of the [DLISTOPEN Procedure](#).

### Syntax

```
HTP.DLISTCLOSE;
```

### Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the [DLISTTERM Procedure](#) and definitions are inserted using the [DLISTDEF Procedure](#).

### Examples

This procedure generates

```
</DL>
```

## DLISTDEF Procedure

This procedure generates the <DD> tag, which inserts definitions of terms. Use this tag for a definition list <DL>. Terms are tagged <DT> and definitions are tagged <DD>.

### Syntax

```
HTP.DLISTDEF (  
    ctext          IN          VARCHAR2    DEFAULT NULL,  
    cclear         IN          VARCHAR2    DEFAULT NULL,  
    cattributes    IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 183–21** *DLISTDEF Procedure Parameters*

Parameter	Description
<code>ctext</code>	The definition of the term.
<code>cclear</code>	The value for the CLEAR attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<DD CLEAR="cclear" cattributes>ctext
```

## DLISTOPEN Procedure

This procedure generates the <DL> tag which starts a definition list. You end a definition list by means of the [DLISTCLOSE Procedure](#).

### Syntax

```
HTP.DLISTOPEN (
  cclear      IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–22** *DLISTOPEN Procedure Parameters*

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Usage Notes

A definition list looks like a glossary: it contains terms and definitions. Terms are inserted using the [DLISTTERM Procedure](#) and definitions are inserted using the [DLISTDEF Procedure](#).

### Examples

This procedure generates

```
<DL CLEAR="cclear" cattributes>
```

## DLISTTERM Procedure

This procedure generates the <DT> tag which defines a term in a definition list <DL>.

### Syntax

```
HTP.DLISTTERM (  
    ctext          IN          VARCHAR2    DEFAULT NULL,  
    cclear         IN          VARCHAR2    DEFAULT NULL,  
    cattributes    IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 183–23** *DLISTTERM Procedure Parameters*

Parameter	Description
<code>ctext</code>	The term.
<code>cclear</code>	The value for the CLEAR attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<DT CLEAR="cclear" cattributes>ctext
```

## EM Procedure

This procedure generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the [EMPHASIS Procedure](#).

### Syntax

```
HTP.EM(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 183–24** *EM Procedure Parameters*

Parameter	Description
ctext	The text to emphasize.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates  
<EM *cattributes*>ctext</EM>

## EMPHASIS Procedure

This procedure generates the <EM> and </EM> tags, which define text to be emphasized. It performs the same task as the [EM Procedure](#).

### Syntax

```
HTP.EMPHASIS(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–25 EMPHASIS Procedure Parameters**

Parameter	Description
<code>ctext</code>	The text to emphasize.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates  
<EM *cattributes*>*ctext*</EM>

## ESCAPE\_SC Procedure

This procedure replaces characters that have special meaning in HTML with their escape sequences. The following characters are converted:

- & to &amp;
- " to &quot;
- < to &lt;
- > to &gt;

This procedure performs the same operation as [PRINTS Procedures](#) and [PS Procedure](#).

### Syntax

```
HTP.ESCAPE_SC(  
    ctext          IN          VARCHAR2);
```

### Parameters

**Table 183–26** *ESCAPE\_SC Procedure Parameters*

Parameter	Description
ctext	The text string to convert.

## FONTCLOSE Procedure

This procedure generates the `</FONT>` tag which marks the end of a section of text with the specified font characteristics. You mark the beginning of the section text by means of the [FONTOPEN Procedure](#).

### Syntax

```
HTP.FONTCLOSE;
```

### Examples

This procedure generates

```
</FONT>
```



## FONTOPEN Procedure

This procedure generates the <FONT> which marks the beginning of section of text with the specified font characteristics. You mark the end of the section text by means of the [FONTCLOSE Procedure](#).

### Syntax

```
HTP.FONTOPEN(
  ccolor      IN      VARCHAR2  DEFAULT NULL,
  cface       IN      VARCHAR2  DEFAULT NULL,
  csize       IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–27** *FONTOPEN Procedure Parameters*

Parameter	Description
<code>ccolor</code>	The value for the <code>COLOR</code> attribute.
<code>cface</code>	The value for the <code>FACE</code> attribute
<code>csize</code>	The value for the <code>SIZE</code> attribute
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

## FORMCHECKBOX Procedure

This procedure generates the `<INPUT>` tag with `TYPE="checkbox"` which inserts a checkbox element in a form. A checkbox element is a button that the user toggles on or off.

### Syntax

```
HTP.FORMCHECKBOX(  
  cname          IN          VARCHAR2,  
  cvalue         IN          VARCHAR2  DEFAULT 'ON',  
  cchecked       IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–28** *FORMCHECKBOX Procedure Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cchecked</code>	If the value for this parameter is not <code>NULL</code> , the <code>CHECKED</code> attribute is added to the tag.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

## FORMCLOSE Procedure

This procedure generates the `</FORM>` tag which marks the end of a form section in an HTML document. You mark the beginning of the form section by means of the [FORMOPEN Procedure](#).

### Syntax

```
HTP.FORMCLOSE;
```

### Examples

This procedure generates

```
</FORM>
```

## FORMOPEN Procedure

This procedure generates the <FORM> tag which marks the beginning of a form section in an HTML document. You mark the end of the form section by means of the [FORMCLOSE Procedure](#).

### Syntax

```
HTP.FORMOPEN (
    curl          IN          VARCHAR2,
    cmethod      IN          VARCHAR2  DEFAULT 'POST',
    ctargget     IN          VARCHAR2  DEFAULT NULL,
    cenctype     IN          VARCHAR2  DEFAULT NULL,
    cattributes  IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–29 FORMOPEN Procedure Parameters**

Parameter	Description
<code>curl</code>	The URL of the WRB or CGI script where the contents of the form is sent. This parameter is required.
<code>cmethod</code>	The value for the METHOD attribute. The value can be "GET" or "POST".
<code>ctargget</code>	The value for the TARGET attribute.
<code>cenctype</code>	The value for the ENCTYPE attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctargget" ENCTYPE="cenctype"
cattributes>
```

## FORMFILE Procedure

This procedure generates the <INPUT> tag with `TYPE="file"` which inserts a file form element. This is used for file uploading for a given page.

### Syntax

```
HTP.FORMFILE(
  cname          IN          VARCHAR2,
  caccept        IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–30 FORMFILE Procedure Parameters**

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>caccept</code>	A comma-delimited list of MIME types for upload.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes>
```

## FORMHIDDEN Procedure

This procedure generates the <INPUT> tag with TYPE="hidden", which inserts a hidden form element. This element is not seen by the user. It submits additional values to the script.

### Syntax

```
HTP.FORMHIDDEN(  
  cname          IN          VARCHAR2,  
  cvalue         IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–31** *FORMHIDDEN Procedure Parameters*

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

## FORMIMAGE Procedure

This procedure generates the <INPUT> tag with `TYPE="image"` which creates an image field that the user clicks to submit the form immediately. The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with `.x` appended, and the y coordinate with `.y` appended. Any `VALUE` attribute is ignored.

### Syntax

```
HTP.FORMIMAGE (
  cname          IN          VARCHAR2,
  csrc           IN          VARCHAR2,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–32 FORMIMAGE Procedure Parameters**

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>csrc</code>	The value for the <code>SRC</code> attribute that specifies the image file.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

## FORMPASSWORD Procedure

This procedure generates the <INPUT> tag with TYPE="password" which creates a single-line text entry field. When the user enters text in the field, each character is represented by one asterisk. This is used for entering passwords.

### Syntax

```
HTP.FORMPASSWORD(  
  cname          IN          VARCHAR2,  
  csize          IN          VARCHAR2,  
  cmaxlength     IN          VARCHAR2  DEFAULT NULL,  
  cvalue         IN          VARCHAR2  DEFAULT NULL,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–33** *FORMPASSWORD Procedure Parameters*

Parameter	Description
cname	The value for the NAME attribute.
csize	The value for the SIZE attribute.
cmmaxlength	The value for the MAXLENGTH attribute.
cvalue	The value for the VALUE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmmaxlength"  
VALUE="cvalue" cattributes>
```



## FORMRADIO Procedure

This procedure generates the `<INPUT>` tag with `TYPE="radio"`, which creates a radio button on the HTML form. Within a set of radio buttons, the user selects only one. Each radio button in the same set has the same name, but different values. The selected radio button generates a name/value pair.

### Syntax

```
HTP.FORMRADIO (
  cname          IN          VARCHAR2,
  cvalue         IN          VARCHAR2,
  cchecked       IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–34 FORMRADIO Procedure Parameters**

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cchecked</code>	If the value for this parameter is not <code>NULL</code> , the <code>CHECKED</code> attribute is added to the tag.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

## FORMRESET Procedure

This procedure generates the `<INPUT>` tag with `TYPE="reset"` which creates a button that, when selected, resets the form fields to their initial values.

### Syntax

```
HTP.FORMRESET(  
  cvalue      IN      VARCHAR2  DEFAULT 'Reset',  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–35** *FORMRESET Procedure Parameters*

Parameter	Description
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

## FORMSELECTCLOSE Procedure

This procedure generates the `</SELECT>` tag which marks the end of a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the beginning of Select form element by means of the [FORMSELECTOPEN Procedure](#). The values are inserted using [FORMSELECTOPTION Procedure](#).

### Syntax

```
HTP.FORMSELECTCLOSE;
```

### Examples

This procedure generates

```
</SELECT>
```

as shown under Examples of the [FORMSELECTOPEN Procedure](#).

## FORMSELECTOPEN Procedure

This procedure generates the <SELECT> tags which creates a Select form element. A Select form element is a listbox where the user selects one or more values. You mark the end of Select form element by means of the [FORMSELECTCLOSE Procedure](#). The values are inserted using [FORMSELETOPTION Procedure](#).

### Syntax

```
FORMSELECTOPEN(
  cname          IN          VARCHAR2,
  cprompt        IN          VARCHAR2  DEFAULT NULL,
  nsize          IN          INTEGER   DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–36 FORMSELECTOPEN Procedure Parameters**

Parameter	Description
cname	The value for the NAME attribute.
cprompt	The string preceding the list box.
nsize	The value for the SIZE attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
</SELECT>
```

so that

```
HTP.FORMSELECTOPEN('greatest_player';
  'Pick the greatest player:');
HTP.FORMSELETOPTION('Messier');
HTP.FORMSELETOPTION('Howe');
HTP.FORMSELETOPTION('Gretzky');
HTP.FORMSELECTCLOSE;
```

generates

```
Pick the greatest player:
<SELECT NAME="greatest_player">
<OPTION>Messier
<OPTION>Howe
<OPTION>Gretzky
</SELECT>
```

## FORMSELECTOPTION Procedure

This procedure generates the <OPTION> tag which represents one choice in a Select element.

### Syntax

```
HTP.FORMSELECTOPTION(
  cvalue      IN      VARCHAR2,
  cselected   IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–37 FORMSELECTOPTION Procedure Parameters**

Parameter	Description
cvalue	The text for the option.
cvalue	If the value for this parameter is not NULL, the SELECTED attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<OPTION SELECTED cattributes>cvalue
```

as shown under Examples of the [FORMSELECTOPEN Procedure](#).

## FORMSUBMIT Procedure

This procedure generates the `<INPUT>` tag with `TYPE="submit"` which creates a button that, when clicked, submits the form. If the button has a `NAME` attribute, the button contributes a name/value pair to the submitted data.

### Syntax

```
HTP.FORMSUBMIT(  
  cname          IN          VARCHAR2  DEFAULT NULL,  
  cvalue         IN          VARCHAR2  DEFAULT 'Submit',  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–38 FORMSUBMIT Procedure Parameters**

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

## FORMTEXT Procedure

This procedure generates the <INPUT> tag with `TYPE="text"`, which creates a field for a single line of text.

### Syntax

```
HTP.FORMTEXT (
  cname          IN          VARCHAR2,
  csize          IN          VARCHAR2  DEFAULT NULL,
  cmaxlength     IN          VARCHAR2  DEFAULT NULL,
  cvalue         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–39** *FORMTEXT Procedure Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>csize</code>	The value for the <code>SIZE</code> attribute.
<code>cmmaxlength</code>	The value for the <code>MAXLENGTH</code> attribute.
<code>cvalue</code>	The value for the <code>VALUE</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmmaxlength" VALUE="cvalue"
cattributes>
```

## FORMTEXTAREA Procedure

This procedure generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text. The same operation is performed by the [FORMTEXTAREA2 Procedure](#) which in addition has the *cwrap* parameter that lets you specify a wrap style.

### Syntax

```
HTP.FORMTEXTAREA(
  cname          IN          VARCHAR2,
  nrows         IN          INTEGER,
  ncolumns      IN          INTEGER,
  calign        , IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–40 FORMTEXTAREA Procedure Parameters**

Parameter	Description
<i>cname</i>	The value for the <i>NAME</i> attribute.
<i>nrows</i>	The value for the <i>ROWS</i> attribute. This is an integer.
<i>ncolumns</i>	The value for the <i>COLS</i> attribute. This is an integer.
<i>calign</i>	The value for the <i>ALIGN</i> attribute.
<i>cattributes</i>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
cattributes></TEXTAREA>
```



## FORMTEXTAREA2 Procedure

This procedure generates the <TEXTAREA> tag, which creates a text field that has no predefined text in the text area. This field enables entering several lines of text. The same operation is performed by the [FORMTEXTAREA Procedure](#) except that in that case you cannot specify a wrap style.

### Syntax

```
HTP.FORMTEXTAREA2 (
  cname          IN          VARCHAR2,
  nrows          IN          INTEGER,
  ncolumns       IN          INTEGER,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cwrap          IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–41 FORMTEXTAREA2 Procedure Parameters**

Parameter	Description
cname	The value for the NAME attribute.
nrows	The value for the ROWS attribute. This is an integer.
ncolumns	The value for the COLS attribute. This is an integer.
calign	The value for the ALIGN attribute.
cwrap	The value for the WRAP attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP="cwrap"
cattributes></TEXTAREA>
```

## FORMTEXTAREACLOSE Procedure

This procedure generates the `</TEXTAREA>` tag which ends a text area form element. You open a text area element by means of either [FORMTEXTAREAOPEN Procedure](#) or [FORMTEXTAREAOPEN2 Procedure](#).

### Syntax

```
HTP.FORMTEXTAREACLOSE;
```

### Examples

This procedure generates

```
</TEXTAREA>
```

## FORMTEXTAREAOPEN Procedure

This procedure generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the [FORMTEXTAREAOPEN2 Procedure](#) which in addition has the `cwrap` parameter that lets you specify a wrap style. You mark the end of a text area form element by means of the [FORMTEXTAREACLOSE Procedure](#).

### Syntax

```
HTP.FORMTEXTAREAOPEN (
  cname          IN          VARCHAR2,
  nrows          IN          INTEGER,
  ncolumns       IN          INTEGER,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–42** *FORMTEXTAREAOPEN Procedure Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>nrows</code>	The value for the <code>ROWS</code> attribute. This is an integer.
<code>ncolumns</code>	The value for the <code>COLS</code> attribute. This is an integer.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" cattributes>
```

## FORMTEXTAREAOPEN2 Procedure

This procedure generates the <TEXTAREA> which marks the beginning of a text area form element. The same operation is performed by the [FORMTEXTAREAOPEN Procedure](#) except that in that case you cannot specify a wrap style. You mark the end of a text area form element by means of the [FORMTEXTAREACLOSE Procedure](#).

### Syntax

```
HTP.FORMTEXTAREAOPEN2 (
  cname          IN          VARCHAR2,
  nrows          IN          INTEGER,
  ncolumns       IN          INTEGER,
  calign         IN          VARCHAR2  DEFAULT NULL,
  cwrap          IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–43** *FORMTEXTAREAOPEN2 Procedure Parameters*

Parameter	Description
<code>cname</code>	The value for the <code>NAME</code> attribute.
<code>nrows</code>	The value for the <code>ROWS</code> attribute. This is an integer.
<code>ncolumns</code>	The value for the <code>COLS</code> attribute. This is an integer.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cwrap</code>	The value for the <code>WRAP</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP = "cwrap"
cattributes>
```

## FRAME Procedure

This procedure generates the <FRAME> tag which defines the characteristics of a frame created by a <FRAMESET> tag.

### Syntax

```
HTP.FRAME (
  csrc          IN          VARCHAR2,
  cname         IN          VARCHAR2  DEFAULT NULL,
  cmarginwidth  IN          VARCHAR2  DEFAULT NULL,
  cmarginheight IN          VARCHAR2  DEFAULT NULL,
  cscrolling    IN          VARCHAR2  DEFAULT NULL,
  cnoresize     IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–44** *FRAME Procedure Parameters*

Parameter	Description
csrc	The URL to display in the frame.
cname	The value for the NAME attribute.
cmarginwidth	The value for the MARGINWIDTH attribute.
cscrolling	The value for the SCROLLING attribute.
cnoresize	If the value for this parameter is not NULL, the NORESIZE attribute is added to the tag.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth"
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```

## FRAMESETCLOSE Procedure

This procedure generates the `</FRAMESET>` tag which ends a frameset section. You mark the beginning of a frameset section by means of the [FRAMESETOPEN Procedure](#).

### Syntax

```
HTP.FRAMESETCLOSE;
```

### Examples

This procedure generates

```
</FRAMESET>
```

## FRAMESETOPEN Procedure

This procedure generates the <FRAMESET> tag which define a frameset section. You mark the end of a frameset section by means of the [FRAMESETCLOSE Procedure](#).

### Syntax

```
HTP.FRAMESETOPEN(
  crows          IN          VARCHAR2  DEFAULT NULL,
  ccols          IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–45** *FRAMESETOPEN Procedure Parameters*

Parameter	Description
<code>crows</code>	The value for the <code>ROWS</code> attribute.
<code>ccols</code>	The value for the <code>COLS</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

## HEADCLOSE Procedure

This procedure generates the `</HEAD>` tag which marks the end of an HTML document head section. You mark the beginning of an HTML document head section by means of the [HEADOPEN Procedure](#).

### Syntax

```
HTP.HEADCLOSE;
```

### Examples

This procedure generates

```
</HEAD>
```



## HEADER Procedure

This procedure generates opening heading tags (<H1> to <H6>) and their corresponding closing tags (</H1> to </H6>).

### Syntax

```
HTP.HEADER(
  nsize      IN      INTEGER,
  cheader    IN      VARCHAR2,
  calign     IN      VARCHAR2  DEFAULT NULL,
  cnowrap    IN      VARCHAR2  DEFAULT NULL,
  cclear     IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–46** *HEADER Procedure Parameters*

Parameter	Description
nsize	The the heading level. This is an integer between 1 and 6.
cheader	The text to display in the heading.
calign	The value for the ALIGN attribute.
cnowrap	The value for the NOWRAP attribute.
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

```
HTP.header (1, 'Overview');
```

produces:

```
<H1>Overview</H1>
```

## HEADOPEN Procedure

This procedure generates the <HEAD> tag which marks the beginning of the HTML document head section. You mark the end of an HTML document head section by means of the [HEADCLOSE Procedure](#).

### Syntax

```
HTP.HEADOPEN;
```

### Examples

This procedure generates

```
<HEAD>
```

## HR Procedure

This procedure generates the <HR> tag, which generates a line in the HTML document. This subprogram performs the same operation as the [LINE Procedure](#).

### Syntax

```
HTP.HR (
  cclear      IN      VARCHAR2  DEFAULT NULL,
  csrc        IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–47 HR Procedure Parameters**

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>csrc</code>	The value for the SRC attribute which specifies a custom image as the source of the line.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

## HTMLCLOSE Procedure

This procedure generates the `</HTML>` tag which marks the end of an HTML document. You use the [HTMLOPEN Procedure](#) to mark the beginning of an HTML document.

### Syntax

```
HTP.HTMLCLOSE;
```

### Examples

This procedure generates

```
</HTML>
```

## HTMLOPEN Procedure

This procedure generates the <HTML> tag which marks the beginning of an HTML document. You use the [HTMLCLOSE Procedure](#) to mark the end of the an HTML document.

### Syntax

```
HTP.HTMLOPEN;
```

### Examples

This procedure generates

```
<HTML>
```

## IMG Procedure

This procedure generates the <IMG> tag which directs the browser to load an image onto the HTML page. The [IMG2 Procedure](#) performs the same operation but additionally uses the `cusemap` parameter.

### Syntax

```
HTP.IMG (
    curl          IN          VARCHAR2    DEFAULT NULL,
    calign        IN          VARCHAR2    DEFAULT NULL,
    calt          IN          VARCHAR2    DEFAULT NULL,
    cismap        IN          VARCHAR2
DEFAULT NULL,
    cattributes  IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 183–48** *IMG Procedure Parameters*

Parameter	Description
<code>curl</code>	The value for the SRC attribute.
<code>calign</code>	The value for the ALIGN attribute.
<code>calt</code>	The value for the ALT attribute which specifies alternative text to display if the browser does not support images.
<code>cismap</code>	If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

## IMG2 Procedure

This procedure generates the <IMG> tag, which directs the browser to load an image onto the HTML page. The [IMG Procedure](#) performs the same operation but does not use the `cusemap` parameter.

### Syntax

```
HTP.IMG2 (
  curl          IN          VARCHAR2  DEFAULT NULL,
  calign        IN          VARCHAR2  DEFAULT NULL,
  calt          IN          VARCHAR2  DEFAULT NULL,
  cismap        IN          VARCHAR2
DEFAULT NULL,
  cusemap       IN          VARCHAR2  DEFAULT NULL,
  cattributes   IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–49** *IMG2 Procedure Parameters*

Parameter	Description
<code>curl</code>	The value for the SRC attribute.
<code>calign</code>	The value for the ALIGN attribute.
<code>calt</code>	The value for the ALT attribute which specifies alternative text to display if the browser does not support images.
<code>cismap</code>	If the value for this parameter is not NULL, the ISMAP attribute is added to the tag. The attribute indicates that the image is an imagemap.
<code>cusemap</code>	The value for the USEMAP attribute which specifies a client-side image map.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap" cattributes>
```

## ISINDEX Procedure

This procedure creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program.

### Syntax

```
HTP.ISINDEX (  
  cprompt      IN      VARCHAR2  DEFAULT NULL,  
  curl         IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–50 ISINDEX Procedure Parameters**

Parameter	Description
cprompt	The value for the PROMPT attribute.
curl	The value for the HREF attribute.

### Examples

This procedure generates

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```



## ITALIC Procedure

This procedure generates the `<I>` and `</I>` tags which direct the browser to render the text in italics.

### Syntax

```
HTP.ITALIC(  
  ctext      IN      VARCHAR2,  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–51** *ITALIC Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in italics.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<I cattributes>ctext</I>
```

## KBD Procedure

This procedure generates the `<KBD>` and `</KBD>` tags which direct the browser to render the text in monospace font. This subprogram performs the same operation as the [KEYBOARD Procedure](#).

### Syntax

```
HTP.KBD(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–52** *KBD Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in monospace.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<KBD cattributes>ctext</KBD>
```

## KEYBOARD Procedure

This procedure generates the <KBD> and </KBD> tags, which direct the browser to render the text in monospace font. This subprogram performs the same operation as the [KBD Procedure](#).

### Syntax

```
HTP.KEYBOARD(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–53** *KEYBOARD Procedure Parameters*

Parameter	Description
ctext	The text to be rendered in monospace.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<KBD cattributes>ctext</KBD>
```

## LINE Procedure

This procedure generates the <HR> tag, which generates a line in the HTML document. This subprogram performs the same operation as the [HR Procedure](#).

### Syntax

```
HTP.LINE(  
  cclear      IN      VARCHAR2  DEFAULT NULL,  
  csrc        IN      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–54** *LINE Procedure Parameters*

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>csrc</code>	The value for the SRC attribute which specifies a custom image as the source of the line.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

## LINKREL Procedure

This procedure generates the <LINK> tag with the REL attribute which delineates the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present. This is the opposite of [LINKREV Procedure](#). This tag indicates a relationship between documents but does not create a link. To create a link, use the [ANCHOR Procedure](#).

### Syntax

```
HTP.LINKREL(  
  crel          IN          VARCHAR2,  
  curl          IN          VARCHAR2,  
  ctitle        IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–55 LINKREL Procedure Parameters**

Parameter	Description
crel	The value for the REL attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

### Examples

This procedure generates

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```

## LINKREV Procedure

This procedure generates the <LINK> tag with the REV attribute which delineates the relationship described by the hypertext link from the target to the anchor. This is the opposite of the [LINKREL Procedure](#). This tag indicates a relationship between documents, but does not create a link. To create a link, use the [ANCHOR Procedure](#).

### Syntax

```
HTP.LINKREV(  
  crev          IN          VARCHAR2,  
  curl          IN          VARCHAR2,  
  ctitle       IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–56 LINKREV Procedure Parameters**

Parameter	Description
crel	The value for the REV attribute.
curl	The value for the URL attribute.
ctitle	The value for the TITLE attribute.

### Examples

This procedure generates

```
<LINK REV="crev" HREF="curl" TITLE="ctitle">
```

## LISTHEADER Procedure

This procedure generates the <LH> and </LH> tags which print an HTML tag at the beginning of the list.

### Syntax

```
HTP.LISTHEADER(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–57 LISTHEADER Procedure Parameters**

Parameter	Description
ctext	The text to place between <LH> and </LH>.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates  
<LH cattributes>ctext</LH>

## LISTINGCLOSE Procedure

This procedure generates the `</LISTING>` tags which marks the end of a section of fixed-width text in the body of an HTML page. To mark the beginning of a section of fixed-width text in the body of an HTML page, use the [LISTINGOPEN Procedure](#).

### Syntax

```
HTP.LISTINGCLOSE;
```

### Examples

This procedure generates

```
</LISTING>
```



## LISTINGOPEN Procedure

This procedure generates the `<LISTING>` tag which marks the beginning of a section of fixed-width text in the body of an HTML page. To mark the end of a section of fixed-width text in the body of an HTML page, use the [LISTINGCLOSE Procedure](#).

### Syntax

```
HTP.LISTINGOPEN;
```

### Examples

This procedure generates

```
<LISTING>
```

## LISTITEM Procedure

This procedure generates the <LI> tag, which indicates a list item.

### Syntax

```
HTP.LISTITEM(  
  ctext          IN          VARCHAR2  DEFAULT NULL,  
  cclear        IN          VARCHAR2  DEFAULT NULL,  
  cdingbat      IN          VARCHAR2  DEFAULT NULL,  
  csrc          IN          VARCHAR2  DEFAULT NULL,  
  cattributes   IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–58 LISTITEM Procedure Parameters**

Parameter	Description
<code>ctext</code>	The text for the list item.
<code>cclear</code>	The value for the CLEAR attribute.
<code>cdingbat</code>	The value for the DINGBAT attribute.
<code>csrc</code>	The value for the SRC attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

## MAILTO Procedure

This procedure generates the <A> tag with the HREF set to 'mailto' prepended to the mail address argument.

### Syntax

```
HTP.MAILTO(
  caddress      IN      VARCHAR2,
  ctext         IN      VARCHAR2,
  cname         IN      VARCHAR2,
  cattributes   IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–59 MAILTO Procedure Parameters**

Parameter	Description
caddress	The email address of the recipient.
ctext	The clickable portion of the link.
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<A HREF="mailto:caddress" NAME="cname" cattributes>ctext</A>
```

so that

```
HTP.mailto('pres@white_house.gov','Send Email to the President');
```

generates:

```
<A HREF="mailto:pres@white_house.gov">Send Email to the President</A>
```

## MAPCLOSE Procedure

This procedure generates the `</MAP>` tag which marks the end of a set of regions in a client-side image map. To mark the beginning of a set of regions in a client-side image map, use the [MAOPEN Procedure](#).

### Syntax

```
HTP.MAPCLOSE;
```

### Examples

This procedure generates

```
</MAP>
```

## MAOPEN Procedure

This procedure generates the <MAP> tag which mark the beginning of a set of regions in a client-side image map. To mark the end of a set of regions in a client-side image map, use the [MAPCLOSE Procedure](#).

### Syntax

```
HTP.MAOPEN(
  cname          IN          VARCHAR2  DEFAULT NULL,
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–60 MAOPEN Procedure Parameters**

Parameter	Description
cname	The value for the NAME attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<MAP NAME="cname" cattributes>
```

## MENULISTCLOSE Procedure

This procedure generates the `</MENU>` tag which ends a list that presents one line for each item. To begin a list of this kind, use the [MENULISTOPEN Procedure](#). The items in the list appear more compact than an unordered list. The [LISTITEM Procedure](#) defines the list items in a menu list.

### Syntax

```
HTP.MENULISTCLOSE;
```

### Examples

This procedure generates

```
</MENU>
```

## MENULISTOPEN Procedure

This procedure generates the <MENU> tag which create a list that presents one line for each item. To end a list of this kind, use the [MENULISTCLOSE Procedure](#). The items in the list appear more compact than an unordered list. The [LISTITEM Procedure](#) defines the list items in a menu list.

### Syntax

```
HTP.MENULISTOPEN;
```

### Examples

This procedure generates

```
<MENU>
```

## META Procedure

This procedure generates the <META> tag, which embeds meta-information about the document and also specifies values for HTTP headers. For example, you can specify the expiration date, keywords, and author name.

### Syntax

```
HTP.META(  
  chttp_equiv IN VARCHAR2,  
  cname      IN VARCHAR2,  
  ccontent   IN VARCHAR2);
```

### Parameters

**Table 183–61 META Procedure Parameters**

Parameter	Description
chttp_equiv	The value for the CHTTP_EQUIV attribute.
cname	The value for the NAME attribute.
ccontent	The value for the CONTENT attribute.

### Examples

This procedure generates

```
<META HTTP-EQUIV="chttp_equiv" NAME ="cname" CONTENT="ccontent">
```

so that

```
HTP.meta ('Refresh', NULL, 120);
```

generates

```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```

On some Web browsers, this causes the current URL to be reloaded automatically every 120 seconds.



## NL Procedure

This procedure generates the <BR> tag which begins a new line of text. It performs the same operation as the [BR Procedure](#).

### Syntax

```
HTP.NL(
  cclear      IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–62 NL Procedure Parameters**

Parameter	Description
cclear	The value for the CLEAR attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<BR CLEAR="cclear" cattributes>
```

## NOBR Procedure

This procedure generates the `<NOBR>` and `</NOBR>` tags which turn off line-breaking in a section of text.

### Syntax

```
HTP.NOBR (  
  ctext          IN          VARCHAR2) ;
```

### Parameters

**Table 183–63** *NOBR Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text that is to be rendered on one line.

### Examples

This procedure generates

```
<NOBR>ctext</NOBR>
```

## NOFRAMESCLOSE Procedure

This procedure generates the `</NOFRAMES>` tag which marks the end of a no-frames section. To mark the beginning of a no-frames section, use the [FRAMESETOPEN Procedure](#). See also [FRAME Procedure](#), [FRAMESETOPEN Procedure](#) and [FRAMESETCLOSE Procedure](#).

### Syntax

```
HTP.NOFRAMESCLOSE;
```

### Examples

This procedure generates

```
</NOFRAMES>
```

## NOFRAMESOPEN Procedure

This procedure generates the <NOFRAMES> tag which mark the beginning of a no-frames section. To mark the end of a no-frames section, use the [FRAMESETCLOSE Procedure](#). See also [FRAME Procedure](#), [FRAMESETOPEN Procedure](#) and [FRAMESETCLOSE Procedure](#).

### Syntax

```
HTP.NOFRAMESOPEN;
```

### Examples

This procedure generates

```
<NOFRAMES>
```

## OLISTCLOSE Procedure

This procedure generates the `</OL>` tag which defines the end of an ordered list. An ordered list presents a list of numbered items. To mark the beginning of a list of this kind, use the [OLISTOPEN Procedure](#). Numbered items are added using [LISTITEM Procedure](#).

### Syntax

```
HTP.OLISTCLOSE;
```

### Examples

This procedure generates

```
</OL>
```

## OLISTOPEN Procedure

This procedure generates the <OL> tag which marks the beginning of an ordered list. An ordered list presents a list of numbered items. To mark the end of a list of this kind, use the [OLISTCLOSE Procedure](#). Numbered items are added using [LISTITEM Procedure](#).

### Syntax

```
HTP.OLISTOPEN(  
  cclear      IN      VARCHAR2  DEFAULT NULL,  
  cwrap       IN      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–64** *OLISTOPEN Procedure Parameters*

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>cwrap</code>	The value for the WRAP attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

## PARA Procedure

This procedure generates the <P> tag which indicates that the text that comes after the tag is to be formatted as a paragraph. You can add attributes to the tag by means of the [PARAGRAPH Procedure](#).

### Syntax

```
HTP.PARA;
```

### Examples

This procedure generates

```
<P>
```

## PARAGRAPH Procedure

You can use this procedure to add attributes to the <P> tag created by the [PARA Procedure](#).

### Syntax

```
HTP.PARAGRAPH(  
    calign          IN          VARCHAR2    DEFAULT NULL,  
    cnowrap        IN          VARCHAR2    DEFAULT NULL,  
    cclear         IN          VARCHAR2    DEFAULT NULL,  
    cattributes    IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 183–65 PARAGRAPH Procedure Parameters**

Parameter	Description
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cnowrap</code>	If the value for this parameter is not <code>NULL</code> , the <code>NOWRAP</code> attribute is added to the tag.
<code>cclear</code>	The value for the <code>CLEAR</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<P ALIGN="align" NOWRAP CLEAR="cclear" cattributes>
```



## PARAM Procedure

This procedure generates the <PARAM> tag which specifies parameter values for Java applets. The values can reference HTML variables. To invoke a Java applet from a Web page, use [APPLETOPEN Procedure](#) to begin the invocation. Use one [PARAM Procedure](#) for each desired name-value pair, and use [APPLETCLOSE Procedure](#) to end the applet invocation.

### Syntax

```
HTP.PARAM(  
  cname          IN          VARCHAR2  
  cvalue         IN          VARCHAR2);
```

### Parameters

**Table 183–66** *PARAM Procedure Parameters*

Parameter	Description
cname	The value for the NAME attribute.
cvalue	The value for the VALUE attribute.

### Examples

This procedure generates

```
<PARAM NAME=cname VALUE="cvalue">
```

## PLAINTEXT Procedure

This procedure generates the `<PLAINTEXT>` and `</PLAINTEXT>` tags which direct the browser to render the text they surround in fixed-width type.

### Syntax

```
HTP.PLAINTEXT(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–67** *PLAINTEXT Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in fixed-width font.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<PLAINTEXT cattributes>ctext</PLAINTEXT>
```

## PRECLOSE Procedure

This procedure generates the `</PRE>` tag which marks the end of a section of preformatted text in the body of the HTML page. To mark the beginning of a section of preformatted text in the body of the HTML page, use the [PREOPEN Procedure](#).

### Syntax

```
HTP.PRECLOSE;
```

### Examples

This procedure generates

```
</PRE>
```

## PREOPEN Procedure

This procedure generates the <PRE> tag which marks the beginning of a section of preformatted text in the body of the HTML page. To mark the end of a section of preformatted text in the body of the HTML page, use the [PRECLOSE Procedure](#).

### Syntax

```
HTP.PREOPEN(  
    cclear          IN          VARCHAR2  DEFAULT NULL,  
    cwidth         IN          VARCHAR2  DEFAULT NULL,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–68** *PREOPEN Procedure Parameters*

Parameter	Description
<code>cclear</code>	The value for the CLEAR attribute.
<code>cwidth</code>	The value for the WIDTH attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

## PRINT Procedures

These procedures generate the specified parameter as a string terminated with the \n newline character. The [PRN Procedures](#) performs the same operation but does not terminate with a newline character.

### Syntax

```
HTP.PRINT (
  cbuf      IN      VARCHAR2);
```

```
HTP.PRINT (
  dbuf      IN      DATE);
```

```
HTP.PRINT (
  nbuf      IN      NUMBER);
```

### Parameters

**Table 183–69 PRINT Procedure Parameters**

Parameter	Description
cbuf	The string to generate terminated by a newline.
dbuf	The string to generate terminated by a newline.
nbuf	The string to generate terminated by a newline.

### Usage Notes

- The \n character is not the same as <BR>. The \n character formats the HTML source but it does not affect how the browser renders the HTML source. Use <BR> to control how the browser renders the HTML source.
- These procedures do not have function equivalents.

## PRINTS Procedure

This procedure generates a string and replaces the following characters with the corresponding escape sequence.

- < to &lt;
- > to &gt;
- " to &quot;
- & to &amp;

If not replaced, the special characters are interpreted as HTML control characters and produce garbled output. This procedure an the [PS Procedure](#) perform the same operation as the [PRN Procedures](#) but with character substitution.

### Syntax

```
HTP.PRINTS (  
    ctext          IN          VARCHAR2);
```

### Parameters

**Table 183–70 PRINTS Procedure Parameters**

Parameter	Description
ctext	The string where to perform character substitution.

### Usage Notes

This procedure does not have an HTF function equivalent (see [Operational Notes](#) on page 183-3 for the HTF implementation).

## PRN Procedures

These procedures generate the specified parameter as a string. Unlike the [PRINT Procedures](#) the string is not terminated with the \n newline character.

### Syntax

```
HTP.PRN (
  cbuf      IN      VARCHAR2);
```

```
HTP.PRN (
  dbuf      IN      DATE);
```

```
HTP.PRN (
  nbuf      IN      NUMBER);
```

### Parameters

**Table 183–71 PRN Procedure Parameters**

Parameter	Description
cbuf	The string to generate (not terminated by a newline).
dbuf	The string to generate (not terminated by a newline).
nbuf	The string to generate (not terminated by a newline).

### Usage Notes

These procedures do not have function equivalents.

## PS Procedure

This procedure generates a string and replaces the following characters with the corresponding escape sequence.

- < to &lt;
- > to &gt;
- " to &quot;
- & to &amp;

If not replaced, the special characters are interpreted as HTML control characters and produce garbled output. This procedure and the [PRINTS Procedure](#) perform the same operation as the [PRN Procedures](#) but with character substitution.

### Syntax

```
HTP.PS (  
    ctext      IN      VARCHAR2);
```

### Parameters

**Table 183–72 PS Procedure Parameters**

Parameter	Description
ctext	The string where to perform character substitution.

### Usage Notes

This procedure does not have an HTF function equivalent (see [Operational Notes](#) on page 183-3 for the HTF implementation).



## S Procedure

This procedure generates the `<S>` and `</S>` tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as [STRIKE Procedure](#).

### Syntax

```
HTP.S (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–73 S Procedure Parameters**

Parameter	Description
<code>ctext</code>	The text to be rendered in strikethrough type.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<S cattributes>ctext</S>
```

## SAMPLE Procedure

This procedure generates the `<SAMP>` and `</SAMP>` tags which direct the browser to render the text they surround in monospace font or however "sample" is defined stylistically.

### Syntax

```
HTP.SAMPLE (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–74** *SAMPLE Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in monospace font.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<SAMP cattributes>ctext</SAMP>
```

## SCRIPT Procedure

This procedure generates the `<SCRIPT>` and `</SCRIPT>` tags which contain a script written in languages such as JavaScript and VBscript.

### Syntax

```
HTP.SCRIPT (
  cscript      IN      VARCHAR2,
  clanguage    IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–75** *SCRIPT Procedure Parameters*

Parameter	Description
<code>cscript</code>	The text of the script. This is the text that makes up the script itself, not the name of a file containing the script.
<code>clanguage</code>	The language in which the script is written. If this parameter is omitted, the user's browser determines the scripting language.

### Examples

This procedure generates

```
<SCRIPT LANGUAGE=clanguage>cscript</SCRIPT>
```

so that

```
HTP.script ('Erupting_Volcano', 'Javascript');
```

generates

```
<SCRIPT LANGUAGE=Javascript>"script text here"</SCRIPT>
```

This causes the browser to run the script enclosed in the tags.

## SMALL Procedure

This procedure generates the `<SMALL>` and `</SMALL>` tags, which direct the browser to render the text they surround using a small font.

### Syntax

```
HTP.SMALL (  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–76** *SMALL Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to be rendered in small font.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<SMALL cattributes>ctext</SMALL>
```

## STRIKE Procedure

This procedure generates the <STRIKE> and </STRIKE> tags which direct the browser to render the text they surround in strikethrough type. This performs the same operation as [S Procedure](#).

### Syntax

```
HTP.STRIKE (  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–77** *STRIKE Procedure Parameters*

Parameter	Description
ctext	The text to be rendered in strikethrough type.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<STRIKE cattributes>ctext</STRIKE>
```

## STRONG Procedure

This procedure generates the `<STRONG>` and `</STRONG>` tags which direct the browser to render the text they surround in bold or however "strong" is defined.

### Syntax

```
HTP.STRONG(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–78** *STRONG Procedure Parameters*

Parameter	Description
<code>ctext</code>	The text to be emphasized.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<STRONG cattributes>ctext</STRONG>
```

## STYLE Procedure

This procedure generates the <STYLE> and </STYLE> tags which include a style sheet in a Web page. You can get more information about style sheets at <http://www.w3.org>. This feature is not compatible with browsers that support only HTML versions 2.0 or earlier. Such browsers will ignore this tag.

### Syntax

```
HTP.STYLE(  
    cstyle          IN          VARCHAR2);
```

### Parameters

**Table 183–79** *STYLE Procedure Parameters*

Parameter	Description
cstyle	The the style information to include.

### Examples

This procedure generates

```
<STYLE>cstyle</STYLE>
```

## SUB Procedure

This procedure generates the `<SUB>` and `</SUB>` tags which direct the browser to render the text they surround as subscript.

### Syntax

```
HTP.SUB(  
    ctext          IN          VARCHAR2,  
    calign         in          VARCHAR2  DEFAULT NULL,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–80 SUB Procedure Parameters**

Parameter	Description
<code>ctext</code>	The text to render in subscript.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<SUB ALIGN="calign" cattributes>ctext</SUB>
```



## SUP Procedure

This procedure generates the <SUP> and </SUP> tags which direct the browser to render the text they surround as superscript.

### Syntax

```
HTP.SUP(  
  ctext      IN      VARCHAR2,  
  calign     in      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–81** *SUP Procedure Parameters*

Parameter	Description
ctext	The text to render in superscript.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

## TABLECAPTION Procedure

This procedure generates the <CAPTION> and </CAPTION> tags which place a caption in an HTML table.

### Syntax

```
HTP.TABLECAPTION(  
    ccaption      IN      VARCHAR2,  
    calign        in      VARCHAR2  DEFAULT NULL,  
    cattributes   IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–82 TABLECAPTION Procedure Parameters**

Parameter	Description
cctext	The text for the caption.
calign	The value for the ALIGN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<CAPTION ALIGN="calign" cattributes>cctext</CAPTION>
```

## TABLECLOSE Procedure

This procedure generates the `</TABLE>` tag which marks the end of an HTML table. To define the beginning of an HTML table, use the [TABLEOPEN Procedure](#).

### Syntax

```
HTP.TABLECLOSE;
```

### Examples

This procedure generates

```
</TABLE>
```

## TABLEDATA Procedure

This procedure generates the <TD> and </TD> tags which insert data into a cell of an HTML table.

### Syntax

```
HTP.TABLEDATA (
    cvalue      IN      VARCHAR2  DEFAULT NULL,
    calign      IN      VARCHAR2  DEFAULT NULL,
    cdp         IN      VARCHAR2  DEFAULT NULL,
    cnowrap     IN      VARCHAR2  DEFAULT NULL,
    crowspan    IN      VARCHAR2  DEFAULT NULL,
    ccolspan    IN      VARCHAR2  DEFAULT NULL,
    cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–83 TABLEDATA Procedure Parameters**

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
crowspan	The value for the COLSPAN attribute.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TD>
```

## TABLEHEADER Procedure

This procedure generates the <TH> and </TH> tags which insert a header cell in an HTML table. The <TH> tag is similar to the <TD> tag except that the text in this case the rows are usually rendered in bold type.

### Syntax

```
HTP.TABLEHEADER (
  cvalue      IN      VARCHAR2  DEFAULT NULL,
  calign      IN      VARCHAR2  DEFAULT NULL,
  cdp         IN      VARCHAR2  DEFAULT NULL,
  cnowrap     IN      VARCHAR2  DEFAULT NULL,
  crowspan    IN      VARCHAR2  DEFAULT NULL,
  ccolspan    IN      VARCHAR2  DEFAULT NULL,
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–84 TABLEHEADER Procedure Parameters**

Parameter	Description
cvalue	The data for the cell in the table.
calign	The value for the ALIGN attribute.
cdp	The value for the DP attribute.
cnowrap	If the value of this parameter is not NULL, the NOWRAP attribute is added to the tag.
crispen	The value for the ROWSPAN attribute.
ccolspan	The value for the COLSPAN attribute.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP
cattributes>cvalue</TH>
```

## TABLEOPEN Procedure

This procedure generates the <TABLE> tag which marks the beginning of an HTML table. To define the end of an HTML table, use the [TABLECLOSE Procedure](#).

### Syntax

```
HTP.TABLEOPEN (
    cborder      IN      VARCHAR2  DEFAULT NULL
    calign       IN      VARCHAR2  DEFAULT NULL,
    cnowrap     IN      VARCHAR2  DEFAULT NULL,
    cclear       IN      VARCHAR2  DEFAULT NULL
    cattributes  IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–85 TABLEOPEN Procedure Parameters**

Parameter	Description
<code>border</code>	The value for the <code>BORDER</code> attribute.
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cnowrap</code>	If the value of this parameter is not <code>NULL</code> , the <code>NOWRAP</code> attribute is added to the tag.
<code>cclear</code>	The value for the <code>CLEAR</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

## TABLEROWCLOSE Procedure

This procedure generates the `</TR>` tag which marks the end of a new row in an HTML table. To mark the beginning of a new row, use the [TABLEROWOPEN Procedure](#).

### Syntax

```
HTP.TABLEROWCLOSE;
```

### Examples

This procedure generates

```
</TABLE>
```

## TABLEROWOPEN Procedure

This procedure generates the <TR> tag which marks the beginning of a new row in an HTML table. To mark the end of a new row, use the [TABLEROWCLOSE Procedure](#).

### Syntax

```
HTP.TABLEROWOPEN(
    calign          IN          VARCHAR2    DEFAULT NULL,
    cvalign         IN          VARCHAR2    DEFAULT NULL,
    cdp             IN          VARCHAR2    DEFAULT NULL,
    cnowrap         IN          VARCHAR2    DEFAULT NULL,
    cattributes     IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 183–86** *TABLEROWOPEN Procedure Parameters*

Parameter	Description
<code>calign</code>	The value for the <code>ALIGN</code> attribute.
<code>cvalign</code>	The value for the <code>VALIGN</code> attribute.
<code>cdp</code>	The value for the <code>DP</code> attribute.
<code>cnowrap</code>	If the value of this parameter is not <code>NULL</code> , the <code>NOWRAP</code> attribute is added to the tag.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```



## TELETYPE Procedure

This procedure generates the `<TT>` and `</TT>` tags which direct the browser to render the text they surround in a fixed width typewriter font, for example, the courier font.

### Syntax

```
HTP.TELETYPE(  
  ctext          IN          VARCHAR2,  
  cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–87 TELETYPE Procedure Parameters**

Parameter	Description
<code>ctext</code>	The text to render in a fixed width typewriter font.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<TT cattributes>ctext</TT>
```

## TITLE Procedure

This procedure generates the <TITLE> and </TITLE> tags which specify the text to display in the titlebar of the browser window.

### Syntax

```
HTP.TITLE(  
    ctitle          IN          VARCHAR2);
```

### Parameters

**Table 183–88** TITLE Procedure Parameters

Parameter	Description
ctitle	The text to display in the titlebar of the browser window.

### Examples

This procedure generates

```
<TITLE>ctitle</TITLE>
```

## ULISTCLOSE Procedure

This procedure generates the `</UL>` tag which marks the end of an unordered list. An unordered list presents items with bullets. To mark the beginning of an unordered list, use the [ULISTOPEN Procedure](#). Add list items with [LISTITEM Procedure](#).

### Syntax

```
HTP.ULISTCLOSE;
```

### Examples

This procedure generates

```
</TABLE>
```

## ULISTOPEN Procedure

This procedure generates the <UL> tag which marks the beginning of an unordered list. An unordered list presents items with bullets. To mark the end of an unordered list, use the [ULISTCLOSE Procedure](#). Add list items with [LISTITEM Procedure](#).

### Syntax

```
HTP.ULISTOPEN(  
  cclear      IN      VARCHAR2  DEFAULT NULL,  
  cwrap       IN      VARCHAR2  DEFAULT NULL,  
  cdingbat    IN      VARCHAR2  DEFAULT NULL,  
  csrc        IN      VARCHAR2  DEFAULT NULL,  
  cattributes IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–89** *ULISTOPEN Procedure Parameters*

Parameter	Description
<code>cclear</code>	The value for the <code>CLEAR</code> attribute.
<code>cwrap</code>	The value for the <code>WRAP</code> attribute.
<code>cdingbat</code>	The value for the <code>DINGBAT</code> attribute.
<code>csrc</code>	The value for the <code>SRC</code> attribute.
<code>cattributes</code>	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

## UNDERLINE Procedure

This procedure generates the <U> and </U> tags, which direct the browser to render the text they surround with an underline.

### Syntax

```
HTP.UNDERLINE (  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–90** *UNDERLINE Procedure Parameters*

Parameter	Description
ctext	The text to render with an underline.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<U cattributes>ctext</U>
```

## VARIABLE Procedure

This procedure generates the <VAR> and </VAR> tags which direct the browser to render the text they surround in italics or however "variable" is defined stylistically.

### Syntax

```
HTP.VARIABLE(  
    ctext          IN          VARCHAR2,  
    cattributes    IN          VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 183–91** VARIABLE Procedure Parameters

Parameter	Description
ctext	The text to render in italics.
cattributes	The other attributes to be included as-is in the tag.

### Examples

This procedure generates

```
<VAR cattributes>ctext</VAR>
```

## WBR Procedure

This procedure generates the <WBR> tag, which inserts a soft line break within a section of NOBR text.

### Syntax

```
HTP.WBR;
```

### Examples

This procedure generates

```
<WBR>
```





This Oracle Multimedia package supports the management and manipulation of Digital Imaging and Communications in Medicine (DICOM) content stored in BLOBs or BFILEs rather than in an ORDDicom object type. See *Oracle Multimedia DICOM Developer's Guide* for a complete description of the ORDDicom object type.

The DICOM standard is the dominant standard for radiology imaging and communication, to which all major manufacturers of radiological devices must conform. Oracle Multimedia DICOM provides native support for DICOM format medical images and other content, such as single-frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports.

The Oracle Multimedia `ORD_DICOM` package provides functions and procedures in the DICOM relational interface to extract standard and private DICOM metadata from DICOM content into customizable XML documents, to perform image processing operations such as format conversion and thumbnail image generation, and to create new DICOM content. This Oracle Multimedia package also provides functions and procedures to check DICOM content for conformance based on a set of user-specified conformance rules, and to make DICOM content anonymous based on user-defined rules that specify the set of attributes to be made anonymous and the actions to be taken to make those attributes anonymous.

The Oracle Multimedia `ORD_DICOM` package also provides functions and procedures in the DICOM data model utility interface to operate on the DICOM data model repository. See *Oracle Multimedia DICOM Developer's Guide* for a complete description of the DICOM data model utility interface.

- [Documentation of ORD\\_DICOM](#)

---

## Documentation of ORD\_DICOM

For a complete description of this package within the context of Oracle Multimedia, see ORD\_DICOM in the *Oracle Multimedia DICOM Developer's Guide*.

---

---

## ORD\_DICOM\_ADMIN

This Oracle Multimedia package is used by Oracle Multimedia Digital Imaging and Communications in Medicine (DICOM) administrators to maintain the Oracle Multimedia DICOM data model repository.

The DICOM data model repository is a collection of documents. An initial set of documents is loaded during installation. After installation, DICOM administrators can use the procedures and functions provided in the data model repository API of the Oracle Multimedia ORD\_DICOM\_ADMIN package to obtain document content as well as to insert, edit, and delete documents in the data model repository.

- [Documentation of ORD\\_DICOM\\_ADMIN](#)

---

## Documentation of ORD\_DICOM\_ADMIN

For a complete description of this package within the context of Oracle Multimedia, see ORD\_DICOM\_ADMIN in the *Oracle Multimedia DICOM Developer's Guide*.

The OWA\_CACHE package provides an interface that enables the PL/SQL Gateway cache to improve the performance of PL/SQL Web applications.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using OWA\\_CACHE](#)
  - Constants
- [Summary of OWA\\_CACHE Subprograms](#)

## Using OWA\_CACHE

---

- [Constants](#)

## Constants

- `system_level CONSTANT VARCHAR(6) := 'SYSTEM';`
- `user_level CONSTANT VARCHAR(4) := 'USER';`

---

## Summary of OWA\_CACHE Subprograms

**Table 186–1 OWA\_CACHE Package Subprograms**

Subprogram	Description
<a href="#">DISABLE Procedure</a> on page 186-5	Disables the cache for this particular request
<a href="#">GET_ETAG Function</a> on page 186-6	Returns the tag associated with the cached content (used in the Validation technique model only)
<a href="#">GET_LEVEL Function</a> on page 186-7	Returns the caching level (used in the Validation technique model only)
<a href="#">SET_CACHE Procedure</a> on page 186-8	Sets up the cache headers for validation model cache type
<a href="#">SET_EXPIRES Procedure</a> on page 186-9	Sets up the cache headers for expires model cache type
<a href="#">SET_NOT_MODIFIED Procedure</a> on page 186-10	Sets up the headers for a not modified cache hit (used in the Validation technique model only)
<a href="#">SET_SURROGATE_CONTROL Procedure</a> on page 186-11	Sets up the headers for a surrogate-control header for Web cache



## **DISABLE Procedure**

This procedure disables the cache for this particular request.

### **Syntax**

```
OWA_CACHE.DISABLE;
```

## GET\_ETAG Function

This function returns the tag associated with the cached content. It is used in the Validation technique only.

### Syntax

```
OWA_CACHE.GET_ETAG  
RETURN VARCHAR2;
```

### Return Values

The tag for cache hit, otherwise NULL.

## GET\_LEVEL Function

This returns the caching level. It is used in the Validation technique model only.

### Syntax

```
OWA_CACHE.GET_LEVEL  
RETURN VARCHAR2;
```

### Return Values

The caching level string ('USER' or 'SYSTEM') for cache hit, otherwise NULL.

## SET\_CACHE Procedure

This sets up the cache headers for validation model cache type.

### Syntax

```
OWA_CACHE.SET_CACHE (  
    p_etag      IN      VARCHAR2,  
    p_level     IN      VARCHAR2);
```

### Parameters

**Table 186–2 SET\_CACHE Procedure Parameters**

Parameter	Description
p_etag	The etag associated with this content
p_level	The caching level ('USER' or 'SYSTEM').

### Exceptions

VALUE\_ERROR is thrown if

- p\_etag is greater than 55
- p\_level is not 'USER' or 'SYSTEM'

## SET\_EXPIRES Procedure

This procedure sets up the cache headers for expires model cache type.

### Syntax

```
OWA_CACHE.SET_EXPIRES(
  p_expires    IN    NUMBER,
  p_level      IN    VARCHAR2);
```

### Parameters

**Table 186–3 SET\_EXPIRES Procedure Parameters**

Parameter	Description
p_expires	The number of minutes this content is valid.
p_level	The caching level ('USER' or 'SYSTEM').

### Exceptions

VALUE\_ERROR is thrown if

- p\_expires is negative or zero
- p\_level is not 'USER' or 'SYSTEM'
- p\_expires is > 525600 (1 year)

## SET\_NOT\_MODIFIED Procedure

This procedure sets up the headers for a not-modified cache hit. It is used in the Validation technique only.

### Syntax

```
OWA_CACHE.SET_NOT_MODIFIED;
```

### Exceptions

VALUE\_ERROR is thrown if If the etag was not passed in

---

## SET\_SURROGATE\_CONTROL Procedure

This procedure sets the headers for a surrogate-control header for Web cache

### Syntax

```
OWA_CACHE.SET_SURROGATE_CONTROL(  
    p_value          IN          VARCHAR2);
```

### Parameters

**Table 186-4 SET\_SURROGATE\_CONTROL Procedure Parameters**

Parameter	Description
p_value	The value to be passed as the Surrogate-Control header

### Exceptions

VALUE\_ERROR is thrown if If p\_value is greater than 55 in length.





The OWA\_COOKIE package provides an interface for sending and retrieving HTTP cookies from the client's browser.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using OWA\\_COOKIE](#)
  - Overview
  - Types
  - Rules and Limits
- [Summary of OWA\\_COOKIE Subprograms](#)

## Using OWA\_COOKIE

- [Overview](#)
- [Types](#)
- [Rules and Limits](#)

## Overview

Cookies are opaque strings sent to the browser to maintain state between HTTP calls. State can be maintained throughout the client's sessions, or longer if an expiration date is included. The system date is calculated with reference to the information specified in the OWA\_CUSTOM package.

## Types

This data type contains cookie name-value pairs. Since the HTTP standard allows cookie names to be overloaded (that is, multiple values can be associated with the same cookie name), there is a PL/SQL `RECORD` holding all values associated with a given cookie name.

```
TYPE vc_arr IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER.
```

```
TYPE COOKIE IS RECORD (  
    name          VARCHAR2(4000),  
    vals          vc_arr,  
    num_vals      INTEGER);
```

## Rules and Limits

All HTTP headers must be in English and the ASCII character set. If the headers are generated from the database, verify they are created in the English language.

## Summary of OWA\_COOKIE Subprograms

**Table 187–1 OWA\_COOKIE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET Function</a> on page 187-7	Gets the value of the specified cookie
<a href="#">GET_ALL Procedure</a> on page 187-8	Gets all cookie name-value pairs
<a href="#">REMOVE Procedure</a> on page 187-9	Removes the specified cookie
<a href="#">SEND procedure</a> on page 187-10	Generates a "Set-Cookie" line in the HTTP header

## GET Function

This function returns the values associated with the specified cookie. The values are returned in a OWA\_COOKIE.COOKIE DATA TYPE.

### Syntax

```
OWA_COOKIE.GET (  
    name          IN          VARCHAR2)  
RETURN COOKIE;
```

### Parameters

**Table 187–2** *GET Procedure Parameters*

Parameter	Description
name	The name of the cookie.

### Return Values

OWA\_COOKIE.COOKIE DATA TYPE.

## GET\_ALL Procedure

This procedure returns all cookie names and their values from the client's browser. The values appear in the order in which they were sent from the browser.

### Syntax

```
OWA_COOKIE.GET_ALL(  
  names          OUT      vc_arr,  
  vals           OUT      vc_arr,  
  num_vals       OUT      INTEGER);
```

### Parameters

**Table 187-3** GET\_ALL Procedure Parameters

Parameter	Description
names	The names of the cookies.
vals	The values of the cookies.
num_vals	The number of cookie-value pairs.



## REMOVE Procedure

This procedure forces a cookie to expire immediately by setting the "expires" field of a Set-Cookie line in the HTTP header to "01-Jan-1990". This procedure must be called within the context of an HTTP header.

### Syntax

```
OWA_COOKIE.REMOVE(  
  name          IN          VARCHAR2,  
  val           IN          VARCHAR2,  
  path          IN          VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 187-4 REMOVE Procedure Parameters**

Parameter	Description
name	The name of the cookie to expire.
val	The value of the cookie.
path	[Currently unused]

## SEND procedure

This procedure generates a Set-Cookie line, which transmits a cookie to the client. This procedure must occur in the context of an HTTP header.

### Syntax

```
OWA_COOKIE.SEND(  
  name          in          varchar2,  
  value         in          varchar2,  
  expires       in          date          DEFAULT NULL,  
  path          in          varchar2     DEFAULT NULL,  
  domain        in          varchar2     DEFAULT NULL,  
  secure        in          varchar2     DEFAULT NULL);
```

### Parameters

**Table 187–5 SEND Procedure Parameters**

Parameter	Description
name	The name of the cookie.
value	The value of the cookie.
expires	The date at which the cookie will expire
path	The value for the path field.
domain	The value for the domain field.
secure	If the value of this parameter is not NULL, the "secure" field is added to the line.

The OWA\_CUSTOM package provides a Global PLSQL Agent Authorization callback function. It is used when PLSQL Agent's authorization scheme is set to GLOBAL or CUSTOM when there is no overriding OWA\_CUSTOM package.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using OWA\\_CUSTOM](#)
  - Constants
- [Summary of OWA\\_CUSTOM Subprograms](#)

## Using OWA\_CUSTOM

- [Constants](#)

## Constants

- `dbms_server_timezone` CONSTANT VARCHAR2(3) := 'PST';
- `dbms_server_gmtdiff` CONSTANT NUMBER := NULL;

---

## Summary of OWA\_CUSTOM Subprograms

**Table 188–1 OWA\_CUSTOM Package Subprograms**

Subprogram	Description
<a href="#">AUTHORIZE Function</a> on page 188-5	Provides a Global PLSQL Agent Authorization callback function

## AUTHORIZE Function

This function is used when PLSQL Agent's authorization scheme is set to GLOBAL or CUSTOM when there is no overriding OWA\_CUSTOM package.

### Syntax

```
OWA_CUSTOM.AUTHORIZE  
RETURN BOOLEAN;
```





The OWA\_IMAGE package provides an interface to access the coordinates where a user clicked on an image.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Summary of OWA\\_IMAGE Subprograms](#)
  - Overview
  - Types
  - Variables
  - Examples
- [Summary of OWA\\_IMAGE Subprograms](#)

## Using OWA\_IMAGE

- [Overview](#)
- [Types](#)
- [Variables](#)
- [Examples](#)

## Overview

Use this package when you have any image map whose destination links invoke the PL/SQL Gateway.

## Types

This data type (`point`) contain the X and Y values of a coordinate, and so provides the coordinates of a user's click on an imagemap. It is defined as:

```
TYPE POINT IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER
```

## Variables

This package variable (`null_point`) of TYPE `POINT` is used to default point parameters. Both the X and the Y fields of this variable are `NULL`.

## Examples

```
CREATE OR REPLACE PROCEDURE process_image
  (my_img in OWA_IMAGE.POINT)
AS
  x integer := OWA_IMAGE.GET_X(my_img);
  y integer := OWA_IMAGE.GET_Y(my_img);
BEGIN
  /* process the coordinate */
END
```

## Summary of OWA\_IMAGE Subprograms

**Table 189–1 OWA\_IMAGE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_X Function</a> on page 189-8	Gets the X value of a point type
<a href="#">GET_Y Function</a> on page 189-9	Gets the Y value of a point type

## GET\_X Function

This function returns the X coordinate of the point where the user clicked on an image map.

### Syntax

```
OWA_IMAGE.GET_X(  
    p          IN          point)  
RETURN INTEGER;
```

### Parameters

**Table 189–2** *GET\_X Procedure Parameters*

Parameter	Description
p	The point where the user clicked.

### Return Values

The X coordinate as an integer.



## GET\_Y Function

This function returns the Y coordinate of the point where the user clicked on an image map.

### Syntax

```
OWA_IMAGE.GET_Y (  
    p          IN          point)  
RETURN INTEGER;
```

### Parameters

**Table 189-3** *GET\_Y Procedure Parameters*

Parameter	Description
p	The point where the user clicked.

### Return Values

The Y coordinate as an integer.



The OWA\_OPT\_LOCK package contains subprograms that impose optimistic locking strategies so as to prevent lost updates.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

This chapter contains the following topics:

- [Using OWA\\_OPT\\_LOCK](#)
  - Overview
  - Types
- [Summary of OWA\\_OPT\\_LOCK Subprograms](#)

## Using OWA\_OPT\_LOCK

- [Overview](#)
- [Types](#)

## Overview

The OWA\_OPT\_LOCK package contains subprograms that impose optimistic locking strategies, so as to prevent lost updates.

It checks if the row that the user is interested in updating has been changed by someone else in the meantime.

The PL/SQL Gateway cannot use conventional database locking schemes because HTTP is a stateless protocol. The OWA\_OPT\_LOCK package gives you two ways of dealing with the lost update problem:

- The hidden fields method stores the previous values in hidden fields in the HTML page. When the user requests an update, the PL/SQL Gateway checks these values against the current state of the database. The update operation is performed only if the values match. To use this method, call the `owa_opt_lock.store_values` procedure.
- The checksum method stores a checksum rather than the values themselves. To use this method, call the `owa_opt_lock.checksum` function.

These methods are optimistic. They do not prevent other users from performing updates, but they do reject the current update if an intervening update has occurred.

## Types

This data type is a PL/SQL table intended to hold ROWIDs.

```
TYPE VCARRAY IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER
```

Note that this is different from the OWA\_TEXT.VC\_ARR DATA TYPE.

---

## Summary of OWA\_OPT\_LOCK Subprograms

**Table 190–1 OWA\_CACHE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">CHECKSUM Functions</a> on page 190-6	Returns the checksum value
<a href="#">GET_ROWID Function</a> on page 190-7	Returns the ROWID value
<a href="#">STORE_VALUES Procedure</a> on page 190-8	Stores unmodified values in hidden fields for later verification
<a href="#">VERIFY_VALUES Function</a> on page 190-9	Verifies the stored values against modified values

## CHECKSUM Functions

This function returns a `checksum` value for a specified string, or for a row in a table. For a row in a table, the function calculates the `checksum` value based on the values of the columns in the row. This function comes in two versions.

The first version returns a `checksum` based on the specified string. This is a "pure" 32-bit `checksum` executed by the database and based on the Internet 1 protocol.

The second version returns a `checksum` based on the values of a row in a table. This is a "impure" 32-bit `checksum` based on the Internet 1 protocol.

### Syntax

```
OWA_OPT_LOCK.CHECKSUM(
  p_buff      IN      VARCHAR2)
RETURN NUMBER;
```

```
OWA_OPT_LOCK.CHECKSUM(
  p_owner     IN      VARCHAR2,
  p_tname     IN      VARCHAR2,
  p_rowid     IN      ROWID)
RETURN NUMBER;
```

### Parameters

**Table 190–2** CHECKSUM Procedure Parameters

Parameter	Description
<code>p_buff</code>	The nstring where you want to calculate the <code>checksum</code> .
<code>p_owner</code>	The owner of the table.
<code>p_tname</code>	The table name.
<code>p_rowid</code>	The row in <code>p_tname</code> where you want to calculate the <code>checksum</code> value. Use the <a href="#">GET_ROWID Function</a> to convert VCARRAY values to proper rowids.



## GET\_ROWID Function

This function returns the ROWID data type from the specified OWA\_OPT\_LOCK.VCARRAY DATA TYPE.

### Syntax

```
OWA_OPT_LOCK.GET_ROWID(  
    p_old_values    IN    varray)  
RETURN ROWID;
```

### Parameters

**Table 190-3** *GET\_ROWID Procedure Parameters*

Parameter	Description
p_old_values	This parameter is usually passed in from an HTML form.

## STORE\_VALUES Procedure

This procedure stores the column values of the row that you want to update later. The values are stored in hidden HTML form elements.

### Syntax

```
OWA_OPT_LOCK.STORE_VALUES(  
  p_owner      IN      VARCHAR2,  
  p_tname      IN      VARCHAR2,  
  p_rowid      IN      ROWID);
```

### Parameters

**Table 190–4 STORE\_VALUES Procedure Parameters**

Parameter	Description
p_owner	The owner of the table.
p_tname	The name of the table.
p_rowid	The row where you want to store values.

### Usage Notes

Before updating the row, compare these values with the current row values to ensure that the values in the row have not been changed. If the values have changed, you can warn the users and let them decide if the update should take place.

The procedure generates series of hidden form elements:

- One hidden form element is created for the table owner. The name of the element is "old\_p\_tname", where p\_tname is the name of the table. The value of the element is the owner name.
- One hidden form element is created for the table name. The name of the element is "old\_p\_tname", where p\_tname is the name of the table. The value of the element is the table name.
- One element is created for each column in the row. The name of the element is "old\_p\_tname", where p\_tname is the name of the table. The value of the element is the column value.

See also the [VERIFY\\_VALUES Function](#).

## VERIFY\_VALUES Function

This function verifies whether values in the specified row have been updated since the last query. Use this function with the [STORE\\_VALUES Procedure](#).

### Syntax

```
OWA_OPT_LOCK.VERIFY_VALUES(
  p_old_values IN varray)
RETURN BOOLEAN;
```

### Parameters

**Table 190–5 VERIFY\_VALUES Procedure Parameters**

Parameter	Description
p_old_values	<p>A PL/SQL table containing the following information:</p> <ul style="list-style-type: none"> <li>▪ p_old_values (1) specifies the owner of the table.</li> <li>▪ p_old_values (2) specifies the table.</li> <li>▪ p_old_values (3) specifies the rowid of the row to verify.</li> </ul> <p>The remaining indexes contain values for the columns in the table.</p> <p>Typically, this parameter is passed in from the HTML form, where you have previously called the <a href="#">STORE_VALUES Procedure</a> to store the row values on hidden form elements.</p>

### Return Values

TRUE if no other update has been performed, otherwise FALSE.



The OWA\_PATTERN package provides an interface to locate text patterns within strings and replace the matched string with another string.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using OWA\\_PATTERN](#)
  - Types
  - Operational Notes
- [Summary of OWA\\_PATTERN Subprograms](#)

## Using OWA\_PATTERN

- [Types](#)
- [Operational Notes](#)

## Types

You can use a pattern as both an input and output parameter. Thus, you can pass the same regular expression to OWA\_PATTERN function calls, and it only has to be parsed once.

- OWA\_PATTERN.PATTERN

## Operational Notes

The OWA\_PATTERN subprograms are overloaded. Specifically, there are six versions of MATCH, and four each of AMATCH and CHANGE. The subprograms use the following parameters:

- `line` - This is the target to be examined for a match. It can be more than one line of text or a `owa_text.multi_line` data type.
- `pat` - This is the pattern that the subprograms attempt to locate in line. The pattern can contain regular expressions. In the `owa_pattern.change` function and procedure, this parameter is called `from_str`.
- `flags` - This specifies whether the search is case-sensitive or if substitutions are done globally.

Use regular expressions with the subprograms in this package. You Specify a regular expression by creating the string you want to match interspersed with various wildcard tokens and quantifiers.

- [Wildcards](#)
- [Quantifiers](#)
- [Flags](#)

### Wildcards

Wildcard tokens match something other than themselves:

**Table 191-1 Wildcard tokens recognized by OWA\_PATTERN package**

Token	Description
<code>^</code>	Matches newline or the beginning of the target
<code>\$</code>	Matches newline or the end of the target
<code>\n</code>	Matches newline
<code>.</code>	Matches any character except newline
<code>\t</code>	Matches tab
<code>\d</code>	Matches digits [0-9]
<code>\D</code>	Matches non-digits [not 0-9]
<code>\w</code>	Matches word characters (0-9, a-z, A-Z, or <code>_</code> )
<code>\W</code>	Matches non-word characters (not 0-9, a-z, A-Z, or <code>_</code> )
<code>\s</code>	Matches whitespace characters (blank, tab, or newline).
<code>\S</code>	Matches non-whitespace characters (not blank, tab, or newline)
<code>\b</code>	Matches "word" boundaries (between <code>\w</code> and <code>\W</code> )
<code>\x&lt;HEX&gt;</code>	Matches the value in the current character set of the two hexadecimal digits
<code>\&lt;OCT&gt;</code>	Matches the value in the current character set of the two or three octal digits
<code>\</code>	Followed by any character not covered by another case matches that character
<code>&amp;</code>	Applies only to CHANGE. This causes the string that matched the regular expression to be included in the string that replaces it. This differs from the other tokens in that it specifies how a target is changed rather than how it is matched. This is explained further under <a href="#">CHANGE Functions and Procedures</a> .



## Quantifiers

Any tokens except & can have their meaning extended by any of the following quantifiers. You can also apply these quantifiers to literals:

**Table 191–2 Quantifiers**

Quantifier	Description
?	0 or 1 occurrence(s)
*	0 or more occurrences
+	1 or more occurrence(s)
{n}	Exactly <i>n</i> occurrences
{n, }	At least <i>n</i> occurrences
{n, m}	At least <i>n</i> , but not more than <i>m</i> , occurrences

## Flags

In addition to targets and regular expressions, the OWA\_PATTERN functions and procedures use flags to affect how they are interpreted.

**Table 191–3 Flags**

Flag	Description
i	This indicates a case-insensitive search.
g	This applies only to CHANGE. It indicates a global replace. That is, all portions of the target that match the regular expression are replaced.

## Summary of OWA\_PATTERN Subprograms

**Table 191-4 OWA\_CACHE Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">AMATCH Function</a> on page 191-7	Determines if a string contains the specified pattern. It lets you specify where in the string the match has to occur
<a href="#">CHANGE Functions and Procedures</a> on page 191-9	Replaces a pattern within a string. If you call it as a function it returns the number of times the regular expression was found and replaced
<a href="#">GETPAT Procedure</a> on page 191-11	Generates a pattern data type from a VARCHAR2 type
<a href="#">MATCH Function</a> on page 191-12	Determines if a string contains the specified pattern

## AMATCH Function

This function specifies if a pattern occurs in a particular location in a string. There are four versions to this function:

- The first and second versions of the function do not save the matched tokens (these are saved in the `backrefs` parameters in the third and fourth versions). The difference between the first and second versions is the `pat` parameter, which can be a `VARCHAR2` or a pattern data type.
- The third and fourth versions of the function save the matched tokens in the `backrefs` parameter. The difference between the third and fourth versions is the `pat` parameter, which can be a `VARCHAR2` or a pattern data type.

---



---

**Note:** If multiple overlapping strings match the regular expression, this function takes the longest match.

---



---

### Syntax

```
OWA_PATTERN.AMATCH (
  line          IN          VARCHAR2,
  from_loc      IN          INTEGER,
  pat           IN          VARCHAR2,
  flags        IN          VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

```
OWA_PATTERN.AMATCH (
  line          IN          VARCHAR2,
  from_loc      IN          INTEGER,
  pat           IN OUT     PATTERN,
  flags        IN          VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

```
OWA_PATTERN.AMATCH (
  line          IN          VARCHAR2
  from_loc      IN          INTEGER
  pat           IN          varchar2
  backrefs      OUT         owa_text.vc_arr
  flags        IN          VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

```
OWA_PATTERN.AMATCH (
  line          IN          VARCHAR2
  from_loc      IN          INTEGER
  pat           IN OUT     PATTERN
  backrefs      OUT         owa_text.vc_arr
  flags        IN          VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

### Parameters

**Table 191–5** *AMATCH Procedure Parameters*

Parameter	Description
<code>line</code>	The text to search in.

**Table 191–5 (Cont.) AMATCH Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
from_loc	The location (in number of characters) in line where the search is to begin.
pat	The string to match. It can contain regular expressions. This can be either a VARCHAR2 or a pattern. If it is a pattern, the output value of this parameter is the pattern matched.
backrefs	The text that is matched. Each token that is matched is placed in a cell in the OWA_TEXT.VC_ARR DATA TYPE PL/SQL table.
flags	Whether or not the search is case-sensitive. If the value of this parameter is "i", the search is case-insensitive. Otherwise the search is case-sensitive.

**Return Values**

The index of the character after the end of the match, counting from the beginning of line. If there was no match, the function returns 0.

## CHANGE Functions and Procedures

This function or procedure searches and replaces a string or `multi_line` data type. If multiple overlapping strings match the regular expression, this subprogram takes the longest match.

### Syntax

```
OWA_PATTERN.CHANGE (
  line          IN OUT  VARCHAR2,
  from_str      IN      VARCHAR2,
  to_str        IN      VARCHAR2,
  flags         IN      VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

```
OWA_PATTERN.CHANGE (
  line          IN OUT  VARCHAR2,
  from_str      IN      VARCHAR2,
  to_str        IN      VARCHAR2,
  flags         IN      VARCHAR2  DEFAULT NULL);
```

```
owa_pattern.change (
  mline        IN OUT  owa_text.multi_line,
  from_str     IN      VARCHAR2,
  to_str       IN      VARCHAR2,
  flags        IN      VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

```
OWA_PATTERN.CHANGE (
  mline        IN OUT  owa_text.multi_line,
  from_str     IN      VARCHAR2,
  to_str       IN      VARCHAR2,
  flags        IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 191–6** *CHANGE Procedure Parameters*

Parameter	Description
<code>line</code>	The text to search in. The output value of this parameter is the altered string.
<code>mline</code>	The text to search in. This is a <code>owa_text.multi_line</code> data type. The output value of this parameter is the altered string.
<code>from_str</code>	The regular expression to replace.
<code>to_str</code>	The substitution pattern.
<code>flags</code>	Whether or not the search is case-sensitive, and whether or not changes are to be made globally. If "i" is specified, the search is case-insensitive. If "g" is specified, changes are made to all matches. Otherwise, the function stops after the first substitution is made.

### Return Values

As a function, it returns the number of substitutions made. If the flag "g" is not used, this number can only be 0 or 1 and only the first match is replaced. The flag "g" specifies to replace all matches with the regular expression.

## Examples

```
OWA_PATTERN.CHANGE('Cats in pajamas', 'C.+in', '& red ')
```

The regular expression matches the substring "Cats in". It then replaces this string with "& red". The ampersand character "&" indicates "Cats in" because that is what matched the regular expression. Thus, this procedure replaces the string "Cats in pajamas" with "Cats in red" If you call this as a function instead of a procedure, the value returned is 1, indicating that a single substitution has been made.

### Example 2:

```
CREATE OR REPLACE PROCEDURE test_pattern as theline VARCHAR2(256);
num_found      INTEGER;
BEGIN
    theline := 'what is the goal?';
    num_found := OWA_PATTERN.CHANGE(theline, 'goal', 'idea', 'g');
    HTP.PRINT(num_found); -- num_found is 1
    HTP.PRINT(theline); -- theline is 'what is the idea?'
END;
/
SHOW ERRORS
```

## GETPAT Procedure

This procedure converts a VARCHAR2 string into an OWA\_PATTERN.PATTERN DATA TYPE.

### Syntax

```
OWA_PATTERN.GETPAT (  
  arg      IN      VARCHAR2,  
  pat      IN OUT  pattern);
```

### Parameters

**Table 191-7 GETPAT Procedure Parameters**

Parameter	Description
arg	The string to convert.
pat	the OWA_PATTERN.PATTERN DATA TYPE initialized with arg.

## MATCH Function

This function determines if a string contains the specified pattern. The pattern can contain regular expressions. If multiple overlapping strings can match the regular expression, this function takes the longest match.

### Syntax

```
owa_pattern.match(
  line          IN          VARCHAR2,
  pat           IN          VARCHAR2,
  flags         IN          VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
```

```
owa_pattern.match(
  line          IN          VARCHAR2,
  pat           IN OUT     PATTERN,
  flags         IN          VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
```

```
owa_pattern.match(
  line          IN          VARCHAR2,
  pat           IN          VARCHAR2,
  backrefs      OUT         owa_text.vc_arr,
  flags         IN          VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
```

```
OWA_PATTERN.MATCH(
  line          IN          VARCHAR2,
  pat           IN OUT     PATTERN,
  backrefs      OUT         owa_text.vc_arr,
  flags         IN          VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
```

```
owa_pattern.match(
  mline         IN          owa_text.multi_line,
  pat           IN          VARCHAR2,
  rlist         OUT         owa_text.row_list,
  flags         IN          VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
```

```
OWA_PATTERN.MATCH(
  mline         IN          owa_text.multi_line,
  pat           IN OUT     pattern,
  rlist         OUT         owa_text.row_list,
  flags         IN          VARCHAR2  DEFAULT NULL)
RETURN BOOLEAN;
```

### Parameters

**Table 191–8** *CHANGE Procedure Parameters*

Parameter	Description
line	The line to search in.
mline	The text to search in. This is a <code>owa_text.multi_line</code> data type..



**Table 191–8 (Cont.) CHANGE Procedure Parameters**

Parameter	Description
pat	The pattern to match. This is either a VARCHAR2 or a OWA_PATTERN.PATTERN DATA TYPE. If it is a pattern, the output value of this parameter is the pattern matched.
backrefs	The text that is matched. Each token that is matched is placed in a cell in the OWA_TEXT.VC_ARR DATA TYPE PL/SQL table. This parameter is a row_list that holds each string in the target that was matched by a sequence of tokens in the regular expression.
rlist	An output parameter containing a list of matches.
flags	Whether or not the search is case-sensitive. If the value of this parameter is "i", the search is case-insensitive. Otherwise the search is case-sensitive.

## Return Values

TRUE if a match was found, FALSE otherwise.

## Examples

KAZOO is the target where it is searching for the zoo.\* regular expression. The period indicates any character other than newline, and the asterisk matches 0 or more of the preceding characters. In this case, it matches any character other than the newline.

Therefore, this regular expression specifies that a matching target consists of zoo, followed by any set of characters neither ending in nor including a newline (which does not match the period). The i flag indicates to ignore case in the search. In this case, the function returns TRUE, which indicates that a match had been found.

```
boolean foundMatch;
foundMatch := owa_pattern.match('KAZOO', 'zoo.*', 'i');
```

The following example searches for the string "goal" followed by any number of characters in sometext. If found,

```
sometext VARCHAR2(256);
pat      VARCHAR2(256);

sometext := 'what is the goal?';
pat      := 'goal.*';
IF OWA_PATTERN.MATCH(sometext, pat)
THEN
    HTP.PRINT('Match found');
ELSE
    HTP.PRINT('Match not found');
END IF;
```

## Operational Notes

- The regular expression in this function can be either a VARCHAR2 or an OWA\_PATTERN.PATTERN DATA TYPE. Create an OWA\_PATTERN.PATTERN DATA TYPE from a string using the OWA\_PATTERN.GETPAT procedure.
- Create a MULTI\_LINE DATA TYPE from a long string using the OWA\_TEXT.STREAM2MULTI procedure. If a multi\_line is used, the rlist parameter specifies a list of chunks where matches were found.

- If the line is a string and not a `multi_line`, you can add an optional output parameter called `backrefs`. This parameter is a `row_list` that holds each string in the target that was matched by a sequence of tokens in the regular expression.

The OWA\_SEC package provides an interface for custom authentication.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using OWA\\_SEC](#)
  - Operational Notes
- [Summary of OWA\\_SEC Subprograms](#)

## Using OWA\_SEC

---

- [Operational Notes](#)

## Operational Notes

Parameters that have default values are optional.

## Summary of OWA\_SEC Subprograms

**Table 192–1 OWA\_SEC Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_CLIENT_HOSTNAME Function</a> on page 192-5	Returns the client's hostname
<a href="#">GET_CLIENT_IP Function</a> on page 192-6	Returns the client's IP address
<a href="#">GET_PASSWORD Function</a> on page 192-7	Returns the password that the user entered
<a href="#">GET_USER_ID Function</a> on page 192-8	Returns the username that the user entered
<a href="#">SET_AUTHORIZATION Procedure</a> on page 192-9	Enables the PL/SQL application to use custom authentication
<a href="#">SET_PROTECTION_REALM Procedure</a> on page 192-10	Defines the realm that the page is in

## GET\_CLIENT\_HOSTNAME Function

This function returns the hostname of the client.

### Syntax

```
OWA_SEC.GET_CLIENT_HOSTNAME  
RETURN VARCHAR2;
```

### Return Values

The hostname.

## GET\_CLIENT\_IP Function

This function returns the IP address of the client.

### Syntax

```
OWA_SEC.GET_CLIENT_IP  
    RETURN OWA_UTIL.IP_ADDRESS;
```

### Return Values

The IP address. The `owa_util.ip_address` data type is a PL/SQL table where the first four elements contain the four numbers of the IP address. For example, if the IP address is 123.45.67.89 and the variable `ipaddr` is of the `owa_util.ip_address` data type, the variable would contain the following values:

```
ipaddr(1) = 123  
ipaddr(2) = 45  
ipaddr(3) = 67  
ipaddr(4) = 89
```



## GET\_PASSWORD Function

This function returns the password that the user used to log in.

### Syntax

```
OWA_SEC.GET_PASSWORD  
RETURN VARCHAR2;
```

### Return Values

The password.

### Usage Notes

For security reasons, this function returns a true value only when custom authentication is used. If you call this function when you are not using custom authentication, the function returns an undefined value. Thus, the database passwords are not exposed.

## GET\_USER\_ID Function

This function returns the username that the user used to log in.

### Syntax

```
OWA_SEC.GET_USER_ID  
RETURN VARCHAR2;
```

### Return Values

The username.

## SET\_AUTHORIZATION Procedure

This procedure, called in the initialization portion of the [OWA\\_CUSTOM](#) package, sets the authorization scheme for the PL/SQL Gateway. This implements your `authorize` function, which authorizes the user before his requested procedure is run. The placement of the `authorize` function depends on the scheme you select.

### Syntax

```
OWA_SEC.SET_AUTHORIZATION(
    scheme          IN          INTEGER);
```

### Parameters

**Table 192–2 SET\_AUTHORIZATION Procedure Parameters**

Parameter	Description
scheme	<p>The authorization scheme. It is one of the following schemes for <code>SET_AUTHORIZATION</code>:</p> <ul style="list-style-type: none"> <li>■ <code>OWA_SEC.NO_CHECK</code> - Specifies that the PL/SQL application is not to do any custom authentication. This is the default.</li> <li>■ <code>OWA_SEC.GLOBAL</code> - Defines an <code>authorize</code> function that is called for all users and all procedures. This is the <a href="#">OWA_CUSTOM.AUTHORIZE Function</a> in the "sys" schema.</li> <li>■ <code>OWA_SEC.PER_PACKAGE</code> - Define an <code>authorize</code> function that is called when procedures in a package or anonymous procedures are called. If the procedures are in a package, the <code>package.AUTHORIZE</code> function in the user's schema is called to authorize the user. If the procedures are not in a package, then the anonymous <code>authorize</code> function in the user's schema is called.</li> <li>■ <code>OWA_SEC.CUSTOM</code> - Implements different <code>authorize</code> functions for each user. The function <a href="#">OWA_CUSTOM.AUTHORIZE Function</a> in the user's schema is called to authorize the user. If the user's schema does not contain an <code>OWA_CUSTOM.AUTHORIZE Function</code>, the PL/SQL Gateway looks for it in the "sys" schema.</li> </ul> <p>The custom <code>authorize</code> function has the following signature:</p> <pre>FUNCTION AUTHORIZE     RETURN BOOLEAN;</pre> <p>If the function returns <code>TRUE</code>, authentication succeeded. If it returns <code>FALSE</code>, authentication failed. If the <code>authorize</code> function is not defined, the Gateway returns an error and fails.</p>

## SET\_PROTECTION\_REALM Procedure

This procedure sets the realm of the page that is returned to the user. The user enters a username and login that already exist in the realm.

### Syntax

```
OWA_SEC.SET_PROTECTION_REALM(  
    realm      IN      VARCHAR2);
```

### Parameters

**Table 192–3** *SET\_PROTECTION\_REALM Procedure Parameters*

Parameter	Description
realm	The realm where the page belongs. This string is displayed to the user.

The OWA\_TEXT package contains subprograms used by OWA\_PATTERN for manipulating strings. They are externalized so you can use them directly.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using OWA\\_TEXT](#)
  - Types
- [Summary of OWA\\_TEXT Subprograms](#)

## Using OWA\_TEXT

- [Types](#)

## Types

- [MULTI\\_LINE DATA TYPE](#)
- [ROW\\_LIST DATA TYPE](#)
- [VC\\_ARR DATA TYPE](#)

### MULTI\_LINE DATA TYPE

This data type is a PL/SQL record that holds large amounts of text. The rows field, of type OWA\_TEXT.VC\_ARR DATA TYPE, contains the text data in the record.

```
TYPE multi_line IS RECORD (  
    rows          vc_arr,  
    num_rows      INTEGER,  
    partial_row    BOOLEAN);
```

### ROW\_LIST DATA TYPE

This is the data type for holding data to be processed.

```
TYPE row_list IS RECORD (  
    rows          int_arr,  
    num_rows      INTEGER);
```

```
int_arr IS DEFINED AS:
```

```
TYPE int_arr IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
```

### VC\_ARR DATA TYPE

This is a component of the [MULTI\\_LINE DATA TYPE](#) and is used for holding large amounts of text.

```
TYPE vc_arr IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

## Summary of OWA\_TEXT Subprograms

**Table 193–1 OWA\_TEXT Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ADD2MULTI Procedure</a> on page 193-5	Adds text to an existing <code>multi_line</code> type
<a href="#">NEW_ROW_LIST Function and Procedure</a> on page 193-6	Creates a new <code>row_list</code>
<a href="#">PRINT_MULTI Procedure</a> on page 193-7	Prints out the contents of a <code>multi_list</code>
<a href="#">PRINT_ROW_LIST Procedure</a> on page 193-8	Prints out the contents of a <code>row_list</code>
<a href="#">STREAM2MULTI Procedure</a> on page 193-9	Converts a <code>varchar2</code> to a <code>multi_line</code> type



## ADD2MULTI Procedure

This procedure adds content to an existing [MULTI\\_LINE DATA TYPE](#).

### Syntax

```
OWA_TEXT.ADD2MULTI (  
    stream          IN          VARCHAR2,  
    mline          IN OUT    multi_line,  
    continue       IN          BOOLEAN DEFAULT TRUE);
```

### Parameters

**Table 193–2** *ADD2MULTI Procedure Parameters*

Parameter	Description
stream	The text to add.
mline	The OWA_TEXT.MULTI_LINE DATA TYPE. The output of this parameter contains <i>stream</i> .
continue	If TRUE, the procedure appends <i>stream</i> within the previous final row (assuming it is less than 32K). If FALSE, the procedure places <i>stream</i> in a new row.

## NEW\_ROW\_LIST Function and Procedure

This function or procedure creates a new `OWA_TEXT.ROW_LIST` DATA TYPE. The function version uses no parameters and returns a new empty `row_list`. The procedure version creates the `row_list` data type as an output parameter.

### Syntax

```
OWA_TEXT.NEW_ROW_LIST
  RETURN ROW_LIST;

OWA_TEXT.NEW_ROW_LIST(
  rlist OUT row_list);
```

### Parameters

**Table 193–3** *NEW\_ROW\_LIST Procedure Parameters*

Parameter	Description
<code>rlist</code>	This is an output parameter containing the new <code>row_list</code> data type

### Return Values

The function version returns the new `row_list` data type.

## PRINT\_MULTI Procedure

This procedure uses the [PRINT Procedures](#) or the [PRN Procedures](#) to print the "rows" field of the OWA\_TEXT.MULTI\_LINE DATA TYPE.

### Syntax

```
OWA_TEXT.PRINT_MULTI(  
    mline          IN          multi_line);
```

### Parameters

**Table 193–4 PRINT\_MULTI Procedure Parameters**

Parameter	Description
<code>mline</code>	The <code>multi_line</code> data type to print.

### Return Values

The contents of the `multi_line`.

## PRINT\_ROW\_LIST Procedure

This procedure uses the [PRINT Procedures](#) or the [PRN Procedures](#) to print the "rows" field of the OWA\_TEXT.ROW\_LIST DATA TYPE.

### Syntax

```
OWA_TEXT.PRINT_ROW_LIST(  
    rlist          IN          multi_line);
```

### Parameters

**Table 193–5 PRINT\_ROW\_LIST Procedure Parameters**

Parameter	Description
rlist	The row_list data type to print.

### Return Values

The contents of the row\_list.

## STREAM2MULTI Procedure

This procedure converts a string to a `multi_line` data type.

### Syntax

```
OWA_TEXT.STREAM2MULTI(  
  stream      IN      VARCHAR2  
  mline      OUT     multi_line);
```

### Parameters

**Table 193–6** *STREAM2MULTI Procedure Parameters*

Parameter	Description
<code>stream</code>	The string to convert.
<code>mline</code>	The stream in <code>OWA_TEXT.MULTI_LINE DATA TYPE</code> format



The OWA\_UTIL package contains utility subprograms for performing operations such as getting the value of CGI environment variables, printing the data that is returned to the client, and printing the results of a query in an HTML table.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

This chapter contains the following topics:

- [Using OWA\\_UTIL](#)
  - Overview
  - Types
- [Summary of OWA\\_UTIL Subprograms](#)

## Using OWA\_UTIL

- [Overview](#)
- [Types](#)



## Overview

The OWA\_UTIL package contains three types of utility subprograms.

- Dynamic SQL Utilities enable you to produce pages with dynamically generated SQL code.
- HTML utilities enable you to retrieve the values of CGI environment variables and perform URL redirects.
- Date utilities enable correct date-handling. Date values are simple strings in HTML, but are treated as a data type by the Oracle database.

## Types

- [DATETYPE Data Type](#)
- [IDENT\\_ARR Data Type](#)
- [IP\\_ADDRESS Data Type](#)

### DATETYPE Data Type

The [TODATE Function](#) converts an item of this type to the type DATE, which is understood and properly handled as data by the database. The procedure [CHOOSE\\_DATE Procedure](#) enables the user to select the desired date.

```
TYPE dateType IS TABLE OF VARCHAR2(10) INDEX BY BINARY_INTEGER;
```

### IDENT\_ARR Data Type

This data type is used for an array.

```
TYPE ident_arr IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

### IP\_ADDRESS Data Type

This data type is used by the [GET\\_CLIENT\\_IP Function](#) in the "OWA\_SEC" package on page 192-1.

```
TYPE ip_address IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
```

---

## Summary of OWA\_UTIL Subprograms

**Table 194–1 OWA\_UTIL Package Subprograms**

Subprogram	Description
<a href="#">BIND_VARIABLES Function</a> on page 194-6	Prepares a SQL query and binds variables to it
<a href="#">CALENDARPRINT Procedures</a> on page 194-7	Prints a calendar
<a href="#">CELLSPRINT Procedures</a> on page 194-8	Prints the contents of a query in an HTML table
<a href="#">CHOOSE_DATE Procedure</a> on page 194-10	Generates HTML form elements that allow the user to select a date
<a href="#">GET_CGI_ENV Function</a> on page 194-11	Returns the value of the specified CGI environment variable
<a href="#">GET_OWA_SERVICE_PATH Function</a> on page 194-12	Returns the full virtual path for the PL/SQL Gateway
<a href="#">GET_PROCEDURE Function</a> on page 194-13	Returns the name of the procedure that is invoked by the PL/SQL Gateway
<a href="#">HTTP_HEADER_CLOSE Procedure</a> on page 194-14	Closes the HTTP header
<a href="#">LISTPRINT Procedure</a> on page 194-15	Generates a HTML form element that contains data from a query
<a href="#">MIME_HEADER Procedure</a> on page 194-16	Generates the Content-type line in the HTTP header
<a href="#">PRINT_CGI_ENV Procedure</a> on page 194-17	Generates a list of all CGI environment variables and their values
<a href="#">REDIRECT_URL Procedure</a> on page 194-18	Generates the Location line in the HTTP header
<a href="#">SHOWPAGE Procedure</a> on page 194-19	Prints a page generated by the HTP and HTF packages in SQL*Plus
<a href="#">SHOWSOURCE Procedure</a> on page 194-20	Prints the source for the specified subprogram
<a href="#">SIGNATURE procedure</a> on page 194-21	Prints a line that says that the page is generated by the PL/SQL Agent
<a href="#">STATUS_LINE Procedure</a> on page 194-22	Generates the Status line in the HTTP header
<a href="#">TABLEPRINT Function</a> on page 194-23	Prints the data from a table in the database as an HTML table
<a href="#">TODATE Function</a> on page 194-26	Converts dateType data to the standard PL/SQL date type
<a href="#">WHO_CALLED_ME Procedure</a> on page 194-27	Returns information on the caller of the procedure.

## BIND\_VARIABLES Function

This function prepares a SQL query by binding variables to it, and stores the output in an opened cursor. Use this function as a parameter to a procedure sending a dynamically generated query. Specify up to 25 bind variables.

### Syntax

```
OWA_UTIL.BIND_VARIABLES (
    theQuery      IN      VARCHAR2  DEFAULT NULL,
    bv1Name       IN      VARCHAR2  DEFAULT NULL,
    bv1Value      IN      VARCHAR2  DEFAULT NULL,
    bv2Name       IN      VARCHAR2  DEFAULT NULL,
    bv2Value      IN      VARCHAR2  DEFAULT NULL,
    bv3Name       IN      VARCHAR2  DEFAULT NULL,
    bv3Value      IN      VARCHAR2  DEFAULT NULL,
    ...
    bv25Name      IN      VARCHAR2  DEFAULT NULL,
    bv25Value     IN      VARCHAR2  DEFAULT NULL)
RETURN INTEGER;
```

### Parameters

**Table 194–2 BIND\_VARIABLES Function Parameters**

Parameter	Description
theQuery	The SQL query statement which must be a SELECT statement
bv1Name	The name of the variable
bv1Value	The value of the variable

### Return Values

An integer identifying the opened cursor.

## CALENDARPRINT Procedures

These procedures creates a calendar in HTML with a visible border. Each date in the calendar can contain any number of hypertext links.

This procedure has 2 versions.

- Version 1 uses a hard-coded query stored in a varchar2 string.
- Version 2 uses a dynamic query prepared with the [BIND\\_VARIABLES Function](#).

### Syntax

```
OWA_UTIL.CALENDARPRINT (
  p_query          IN          VARCHAR2,
  p_mf_only        IN          VARCHAR2  DEFAULT 'N' );
```

```
OWA_UTIL.CALENDARPRINT (
  p_cursor         IN          INTEGER,
  p_mf_only        IN          VARCHAR2  DEFAULT 'N' );
```

### Parameters

**Table 194–3 CALENDARPRINT Procedure Parameters**

Parameter	Description
p_query	A PL/SQL query.
p_cursor	A PL/SQL cursor containing the same format as p_query.
p_mf_only	If "N" (the default), the generated calendar includes Sunday through Saturday. Otherwise, it includes Monday through Friday only.

### Usage Notes

Design your query as follows:

- The first column is a DATE. This correlates the information produced by the query with the calendar output generated by the procedure.
- The query output must be sorted on this column using ORDER BY.
- The second column contains the text, if any, that you want printed for that date.
- The third column contains the destination for generated links. Each item in the second column becomes a hypertext link to the destination given in this column. If this column is omitted, the items in the second column are simple text, not links.

## CELLSPRINT Procedures

This procedure generates an HTML table from the output of a SQL query. SQL atomic data items are mapped to HTML cells and SQL rows to HTML rows. You must write the code to begin and end the HTML table. There are nine versions of this procedure:

- The first version passes the results of a query into an index table. Perform the query and CELLSPRINT does the formatting. To have more control in generating an HTML table from the output of an SQL query, use the [FORMAT\\_CELL Function](#) in the "HTF" package on page 182-1.
- The second and third versions display rows (up to the specified maximum) returned by the query or cursor.
- The fourth and fifth versions exclude a specified number of rows from the HTML table. Use the fourth and fifth versions to scroll through result sets by saving the last row seen in a hidden form element.
- The sixth through ninth versions are the same as the first four versions, except that they return a row count output parameter.

### Syntax

```
OWA_UTIL.CELLSPRINT (
  p_colCnt      IN    INTEGER,
  p_resultTbl   IN    vc_arr,
  p_format_numbers IN  VARCHAR2  DEFAULT NULL);
```

```
OWA_UTIL.CELLSPRINT (
  p_theQuery     IN    VARCHAR2,
  p_max_rows     IN    NUMBER    DEFAULT 100,
  p_format_numbers IN  VARCHAR2  DEFAULT NULL);
```

```
OWA_UTIL.CELLSPRINT (
  p_theCursor    IN    INTEGER,
  p_max_rows     IN    NUMBER    DEFAULT 100,
  p_format_numbers IN  VARCHAR2  DEFAULT NULL);
```

```
OWA_UTIL.CELLSPRINT (
  p_theQuery     IN    VARCHAR2,
  p_max_rows     IN    NUMBER    DEFAULT 100,
  p_format_numbers IN  VARCHAR2  DEFAULT NULL,
  p_skip_rec     IN    NUMBER    DEFAULT 0,
  p_more_data    OUT   BOOLEAN);
```

```
OWA_UTIL.CELLSPRINT (
  p_theCursor    IN    INTEGER,
  p_max_rows     IN    NUMBER    DEFAULT 100,
  p_format_numbers IN  VARCHAR2  DEFAULT NULL,
  p_skip_rec     IN    NUMBER    DEFAULT 0,
  p_more_data    OUT   BOOLEAN);
```

```
OWA_UTIL.CELLSPRINT (
  p_theQuery     IN    VARCHAR2,
  p_max_rows     IN    NUMBER    DEFAULT 100,
  p_format_numbers IN  VARCHAR2  DEFAULT NULL,
  p_reccnt      OUT   NUMBER);
```

```
OWA_UTIL.CELLSPRINT (
```

```

p_theCursor      IN    INTEGER,
p_max_rows      IN    NUMBER    DEFAULT 100,
p_format_numbers IN    VARCHAR2  DEFAULT NULL,
p_reccnt       OUT   NUMBER);

OWA_UTIL.CELLSPRINT(
  p_theQuery      IN    VARCHAR2,
  p_max_rows      IN    NUMBER    DEFAULT 100,
  p_format_numbers IN    VARCHAR2  DEFAULT NULL,
  p_skip_rec      IN    NUMBER    DEFAULT 0,
  p_more_data     OUT   BOOLEAN,
  p_reccnt       OUT   NUMBER);

OWA_UTIL.CELLSPRINT(
  p_theCursor      IN    INTEGER,
  p_max_rows      IN    NUMBER    DEFAULT 100,
  p_format_numbers IN    VARCHAR2  DEFAULT NULL,
  p_skip_rec      IN    NUMBER    DEFAULT 0,
  p_more_data     OUT   BOOLEAN,
  p_reccnt       OUT   NUMBER);

```

## Parameters

**Table 194–4** *CELLSPRINT Procedure Parameters*

Parameter	Description
p_query	A PL/SQL query.
p_colCnt	The number of columns in the table.
p_theQuery	A SQL SELECT statement.
p_theCursor	A cursor ID. This can be the return value from the <a href="#">BIND_VARIABLES Function</a> .
p_max_rows	The maximum number of rows to print.
p_format_numbers	If the value of this parameter is not NULL, number fields are right justified and rounded to two decimal places.
p_skip_rec	The number of rows to exclude from the HTML table.
p_more_data	TRUE if there are more rows in the query or cursor, FALSE otherwise.
p_reccnt	The number of rows that have been returned by the query. This value does not include skipped rows (if any).
p_resultTbl	The index table which will contain the result of the query. Each entry in the query will correspond to one column value.

## Examples

This function generates

```
<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>...
```

## CHOOSE\_DATE Procedure

This procedure generates three HTML form elements that allow the user to select the day, the month, and the year.

### Syntax

```
OWA_UTIL.CHOOSE_DATE (
  p_name      IN      VARCHAR2,
  p_date      IN      DATE      DEFAULT SYSDATE) ;
```

### Parameters

**Table 194–5 CHOOSE\_DATE Procedure Parameters**

Parameter	Description
p_name	The name of the form elements.
p_date	The initial date that is selected when the HTML page is displayed.

### Usage Notes

- The parameter in the procedure that receives the data from these elements must be a [GET\\_CGI\\_ENV Function](#).
- Use the [TODATE Function](#) to convert the [GET\\_CGI\\_ENV Function](#) value to the standard Oracle DATE data type.

### Examples

```
<SELECT NAME="p_name" SIZE="1">
<OPTION value="01">1
...
<OPTION value="31">31
</SELECT>
-
<SELECT NAME="p_name" SIZE="1">
<OPTION value="01">JAN
...
<OPTION value="12">DEC
</SELECT>
-
<SELECT NAME="p_name" SIZE="1">
<OPTION value="1992">1992
...
<OPTION value="2002">2002
</SELECT>
```



## GET\_CGI\_ENV Function

This function returns the value of the specified CGI environment variable.

### Syntax

```
OWA_UTIL.GET_CGI_ENV(  
    param_name      IN      VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 194–6** GET\_CGI\_ENV Function Parameters

Parameter	Description
param_name	The name of the CGI environment variable. It is case-insensitive.

### Return Values

The value of the specified CGI environment variable. If the variable is not defined, the function returns NULL.

## GET\_OWA\_SERVICE\_PATH Function

This function returns the full virtual path of the PL/SQL Gateway that is handling the request.

### Syntax

```
OWA_UTIL.GET_OWA_SERVICE_PATH  
RETURN VARCHAR2;
```

### Return Values

A virtual path of the PL/SQL Gateway that is handling the request.

## GET\_PROCEDURE Function

This function returns the name of the procedure that is being invoked by the PL/SQL Gateway.

### Syntax

```
OWA_UTIL.GET_PROCEDURE  
RETURN VARCHAR2;
```

### Return Values

The name of a procedure, including the package name if the procedure is defined in a package.

## HTTP\_HEADER\_CLOSE Procedure

This procedure generates a newline character to close the HTTP header.

### Syntax

```
OWA_UTIL.HTTP_HEADER_CLOSE;
```

### Return Values

A newline character, which closes the HTTP header.

### Usage Notes

- Use this procedure if you have not closed the header by using the `bclose_header` parameter in calls such as [MIME\\_HEADER Procedure](#), [REDIRECT\\_URL Procedure](#), or [STATUS\\_LINE Procedure](#)
- The HTTP header must be closed before any `HTTP.PRINT` or `HTTP.PRN` calls.

## LISTPRINT Procedure

This procedure generates an HTML selection list form element from the output of a SQL query. There are two versions of this procedure.

- The first version contains a hard-coded SQL query.
- The second version uses a dynamic query prepared with the [BIND\\_VARIABLES Function](#).

### Syntax

```
OWA_UTIL.LISTPRINT (
  p_theQuery      IN      VARCHAR2,
  p_cname         IN      VARCHAR2,
  p_nsize         IN      NUMBER,
  p_multiple      IN      BOOLEAN  DEFAULT FALSE);
```

```
OWA_UTIL.LISTPRINT (
  p_theCursor     IN      INTEGER,
  p_cname         IN      VARCHAR2,
  p_nsize         IN      NUMBER,
  p_multiple      IN      BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 194–7 LISTPRINT Procedure Parameters**

Parameter	Description
p_theQuery	The SQL query.
p_theCursor	The cursor ID. This can be the return value from the <a href="#">BIND_VARIABLES Function</a> .
p_cname	The name of the HTML form element.
p_nsize	The size of the form element (this controls how many items the user can see without scrolling).
p_multiple	Whether multiple selection is permitted.

### Usage Notes

The columns in the output of the query are handled in the following manner:

- The first column specifies the values that are sent back. These values are for the VALUE attribute of the OPTION tag.
- The second column specifies the values that the user sees.
- The third column specifies whether or not the row is marked as SELECTED in the OPTION tag. If the value is not NULL, the row is selected.

### Examples

```
<SELECT NAME="p_cname" SIZE="p_nsize">
<OPTION SELECTED value='value_from_the_first_column'>value_from_the_second_column
<OPTION SELECTED value='value_from_the_first_column'>value_from_the_second_column
...
</SELECT>
```

## MIME\_HEADER Procedure

This procedure changes the default MIME header that the script returns. This procedure must come before any `HTP.PRINT` or `HTP.PRN` calls to direct the script not to use the default MIME header.

### Syntax

```
OWA_UTIL.MIME_HEADER(
  ccontent_type  IN      VARCHAR2  DEFAULT 'text/html',
  bclose_header  IN      BOOLEAN    DEFAULT TRUE,
  ccharset       IN      VARCHAR2  DEFAULT NULL);
```

### Parameters

**Table 194–8** *MIME\_HEADER Procedure Parameters*

Parameter	Description
<code>ccontent_type</code>	The MIME type to generate
<code>bclose_header</code>	Whether or not to close the HTTP header. If <code>TRUE</code> , two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.
<code>ccharset</code>	The character set to use. The character set only makes sense if the MIME type is of type 'text'. Therefore, the character set is only tagged on to the Content-Type header only if the MIME type passed in is of type 'text'. Any other MIME type, such as 'image', will not have any character set tagged on.

### Examples

```
Content-type: <ccontent_type>; charset=<ccharset>
```

so that

```
owa_util.mime_header('text/plain', false, 'ISO-8859-4')
```

generates

```
Content-type: text/plain; charset=ISO-8859-4\n
```

## PRINT\_CGI\_ENV Procedure

This procedure generates all the CGI environment variables and their values made available by the PL/SQL Gateway to the stored procedure.

### Syntax

```
OWA_UTIL.PRINT_CGI_ENV;
```

### Examples

This procedure generates a list in the following format:

```
cgi_env_var_name = value\n
```

## REDIRECT\_URL Procedure

This procedure specifies that the application server is to visit the specified URL. The URL may specify either a Web page to return or a program to execute.

### Syntax

```
OWA_UTIL.REDIRECT_URL(  
    curl          IN          VARCHAR2  
    bclose_header IN          BOOLEAN    DEFAULT TRUE);
```

### Parameters

**Table 194–9 REDIRECT\_URL Function Parameters**

Parameter	Description
curl	The URL to visit.
bclose_header	Whether or not to close the HTTP header. If TRUE, two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.

### Usage Notes

This procedure must come before any HTTP procedure or HTTP function call.

### Examples

This procedure generates

```
Location: <curl>\n\n
```



## SHOWPAGE Procedure

This procedure prints out the HTML output of a procedure in SQL\*Plus. The procedure must use the HTP or HTF packages to generate the HTML page, and this procedure must be issued after the HTP or HTF page-generating subprogram has been called and before any other HTP or HTF subprograms are directly or indirectly called.

### Syntax

```
OWA_UTIL.SHOWPAGE;
```

### Usage Notes

- This method is useful for generating pages filled with static data.
- This procedure uses the [DBMS\\_OUTPUT](#) package and is limited to 32767 characters for each line and an overall buffer size of 1,000,000 bytes.

### Examples

The output of htp procedure is displayed in SQL\*Plus, SQL\*DBA, or Oracle Server Manager. For example:

```
SQL> set serveroutput on
SQL> spool gretzky.html
SQL> execute hockey.pass("Gretzky")
SQL> execute owa_util.showpage
SQL> exit
```

This would generate an HTML page that could be accessed from Web browsers.

## SHOWSOURCE Procedure

This procedure prints the source of the specified procedure, function, or package. If a procedure or function which belongs to a package is specified, then the entire package is displayed.

### Syntax

```
OWA_UTIL.SHOWSOURCE (  
    cname          IN      VARCHAR2);
```

### Parameters

**Table 194–10** *SHOWSOURCE Procedure Parameters*

Parameter	Description
cname	The function or procedure whose source you want to show.

## SIGNATURE procedure

This procedure generates an HTML line followed by a signature line on the HTML document. If a parameter is specified, the procedure also generates a hypertext link to view the PL/SQL source for that procedure. The link calls the [SHOWSOURCE Procedure](#).

### Syntax

```
OWA_UTIL.SIGNATURE;  
  
OWA_UTIL.SIGNATURE (  
    cname          IN          VARCHAR2);
```

### Parameters

**Table 194–11 SIGNATURE Procedure Parameters**

Parameter	Description
cname	The function or procedure whose source you want to show.

### Examples

Without a parameter, the procedure generates a line that looks like the following:

```
This page was produced by the PL/SQL Agent on August 9, 2001 09:30.
```

With a parameter, the procedure generates a signature line in the HTML document that looks like the following:

```
This page was produced by the PL/SQL Agent on 8/09/01 09:30  
View PL/SQL Source
```

## STATUS\_LINE Procedure

This procedure sends a standard HTTP status code to the client. This procedure must come before any `http.print` or `http.prn` calls so that the status code is returned as part of the header, rather than as "content data".

### Syntax

```
OWA_UTIL.STATUS_LINE(  
  nstatus      IN      INTEGER,  
  creason      IN      VARCHAR2  DEFAULT NULL,  
  bclose_header IN      BOOLEAN  DEFAULT TRUE);
```

### Parameters

**Table 194–12 STATUS\_LINE Procedure Parameters**

Parameter	Description
<code>nstatus</code>	The status code.
<code>creason</code>	The string for the status code.
<code>bclose_header</code>	Whether or not to close the HTTP header. If <code>TRUE</code> , two newlines are sent, which closes the HTTP header. Otherwise, one newline is sent, and the HTTP header remains open.

### Examples

This procedure generates

```
Status: <nstatus> <creason>\n\n
```

## TABLEPRINT Function

This function generates either preformatted tables or HTML tables (depending on the capabilities of the user's browser) from database tables.

### Syntax

```
OWA_UTIL.TABLEPRINT (
  ctable          IN          VARCHAR2,
  cattributes     IN          VARCHAR2  DEFAULT NULL,
  ntable_type     IN          INTEGER   DEFAULT HTML_TABLE,
  ccolumns        IN          VARCHAR2  DEFAULT '*',
  cclauses        IN          VARCHAR2  DEFAULT NULL,
  ccol_aliases    IN          VARCHAR2  DEFAULT NULL,
  nrow_min        IN          NUMBER    DEFAULT 0,
  nrow_max        IN          NUMBER    DEFAULT NULL)
RETURN BOOLEAN;
```

### Parameters

**Table 194–13** TABLEPRINT Function Parameters

Parameter	Description
ctable	The database table.
cattributes	Other attributes to be included as-is in the tag.
ntable_type	How to generate the table. Specify <code>HTML_TABLE</code> to generate the table using <code>&lt;TABLE&gt;</code> tags or <code>PRE_TABLE</code> to generate the table using the <code>&lt;PRE&gt;</code> tags. These are constants: <ul style="list-style-type: none"> <li>■ <code>HTML_TABLE CONSTANT INTEGER := 1;</code></li> <li>■ <code>PRE_TABLE CONSTANT INTEGER := 2;</code></li> </ul>
ccolumns	A comma-delimited list of columns from <code>ctable</code> to include in the generated table.
cclauses	WHERE or ORDER BY clauses, which specify which rows to retrieve from the database table, and how to order them.
ccol_aliases	A comma-delimited list of headings for the generated table.
nrow_min	The first row, of those retrieved, to display.
nrow_max	The last row, of those retrieved, to display.

### Return Values

Returns `TRUE` if there are more rows beyond the `nrow_max` requested, `FALSE` otherwise.

### Usage Notes

- RAW columns are supported, but LONG RAW columns are not. References to LONG RAW columns will print the result 'Not Printable'.
- Note that in this function, `cattributes` is the second rather than the last parameter.

## Examples

For browsers that do not support HTML tables, create the following procedure:

```
CREATE OR REPLACE PROCEDURE showemps IS
  ignore_more BOOLEAN;
BEGIN
  ignore_more := OWA_UTIL.TABLEPRINT('emp', 'BORDER', OWA_UTIL.PRE_TABLE);
END;
```

Requesting a URL such as

<http://myhost:7777/pls/hr/showemps>

returns to the following to the client:

<PRE>

```
-----
| EMPNO | ENAME | JOB      | MGR  | HIREDATE | SAL  | COMM | DEPTNO |
-----
| 7369 | SMITH | CLERK    | 7902 | 17-DEC-80 | 800  |      | 20     |
| 7499 | ALLEN | SALESMAN | 7698 | 20-FEB-81 | 1600 | 300  | 30     |
| 7521 | WARD  | SALESMAN | 7698 | 22-FEB-81 | 1250 | 500  | 30     |
| 7566 | JONES | MANAGER  | 7839 | 02-APR-81 | 2975 |      | 20     |
| 7654 | MARTIN | SALESMAN | 7698 | 28-SEP-81 | 1250 | 1400 | 30     |
| 7698 | BLAKE | MANAGER  | 7839 | 01-MAY-81 | 2850 |      | 30     |
| 7782 | CLARK | MANAGER  | 7839 | 09-JUN-81 | 2450 |      | 10     |
| 7788 | SCOTT | ANALYST  | 7566 | 09-DEC-82 | 3000 |      | 20     |
| 7839 | KING  | PRESIDENT |      | 17-NOV-81 | 5000 |      | 10     |
| 7844 | TURNER | SALESMAN | 7698 | 08-SEP-81 | 1500 | 0    | 30     |
| 7876 | ADAMS | CLERK    | 7788 | 12-JAN-83 | 1100 |      | 20     |
| 7900 | JAMES | CLERK    | 7698 | 03-DEC-81 | 950  |      | 30     |
| 7902 | FORD  | ANALYST  | 7566 | 03-DEC-81 | 3000 |      | 20     |
| 7934 | MILLER | CLERK    | 7782 | 23-JAN-82 | 1300 |      | 10     |
-----
```

</PRE>

To view the employees in department 10, and only their employee ids, names, and salaries, create the following procedure:

```
CREATE OR REPLACE PROCEDURE showemps_10 IS
  ignore_more BOOLEAN;
begin
  ignore_more := OWA_UTIL.TABLEPRINT
    ('EMP', 'BORDER', OWA_UTIL.PRE_TABLE,
     'empno, ename, sal', 'WHERE deptno=10 ORDER BY empno',
     'Employee Number, Name, Salary');
END;
```

A request for a URL like

[http://myhost:7777/pls/hr/showemps\\_10](http://myhost:7777/pls/hr/showemps_10)

would return the following to the client:

<PRE>

```
-----
| Employee Number | Name   | Salary |
-----
| 7782            | CLARK | 2450   |
| 7839            | KING  | 5000   |
| 7934            | MILLER | 1300   |
-----
```

```
</PRE>
```

For browsers that support HTML tables, to view the department table in an HTML table, create the following procedure:

```
CREATE OR REPLACE PROCEDURE showdept IS
  ignore_more BOOLEAN;
BEGIN
  ignore_more := oWA_UTIL.TABLEPRINT('dept', 'BORDER');
END;
```

A request for a URL like

```
http://myhost:7777/pls/hr/showdept
```

would return the following to the client:

```
<TABLE BORDER>
<TR>
<TH>DEPTNO</TH>
<TH>DNAME</TH>
<TH>LOC</TH>
</TR>
<TR>
<TD ALIGN="LEFT">10</TD>
<TD ALIGN="LEFT">ACCOUNTING</TD>
<TD ALIGN="LEFT">NEW YORK</TD>
</TR>
<TR>
<TD ALIGN="LEFT">20</TD>
<TD ALIGN="LEFT">RESEARCH</TD>
<TD ALIGN="LEFT">DALLAS</TD>
</TR>
<TR>
<TD ALIGN="LEFT">30</TD>
<TD ALIGN="LEFT">SALES</TD>
<TD ALIGN="LEFT">CHICAGO</TD>
</TR>
<TR>
<TD ALIGN="LEFT">40</TD>
<TD ALIGN="LEFT">OPERATIONS</TD>
<TD ALIGN="LEFT">BOSTON</TD>
</TR>
</TABLE>
```

A Web browser would format this to look like the following table:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

## TODATE Function

This function converts the [DATETYPE Data Type](#) to the standard Oracle `DATE` type.

### Syntax

```
OWA_UTIL.TODATE (  
    p_dateArray      IN      dateType)  
RETURN DATE;
```

### Parameters

**Table 194–14** *TODATE Function Parameters*

Parameter	Description
<code>p_dateArray</code>	The value to convert.



## WHO\_CALLED\_ME Procedure

This procedure returns information (in the form of output parameters) about the PL/SQL code unit that invoked it.

### Syntax

```
OWA_UTIL.WHO_CALLED_ME (
  owner          OUT          VARCHAR2,
  name           OUT          VARCHAR2,
  lineno        OUT          NUMBER,
  caller_t      OUT          VARCHAR2);
```

### Parameters

**Table 194–15** WHO\_CALLED\_ME Procedure Parameters

Parameter	Description
owner	The owner of the program unit.
name	The name of the program unit. This is the name of the package, if the calling program unit is wrapped in a package, or the name of the procedure or function if the calling program unit is a standalone procedure or function. If the calling program unit is part of an anonymous block, this is NULL.
lineno	The line number within the program unit where the call was made.
caller_t	The type of program unit that made the call. The possibilities are: package body, anonymous block, procedure, and function. Procedure and function are only for standalone procedures and functions.



The `SDO_CS` package contains functions and procedures for working with coordinate systems. You can perform explicit coordinate transformations on a single geometry or an entire layer of geometries (that is, all geometries in a specified column in a table).

- [Documentation of SDO\\_CS](#)

---

## Documentation of SDO\_CS

For a complete description of this package within the context of Oracle Spatial, see SDO\_CS in the *Oracle Spatial Developer's Guide*.

---

---

## SDO\_CSW\_PROCESS

The SDO\_CSW\_PROCESS package contains subprograms for various processing operations related to support for Catalog Services for the Web (CSW).

- [Documentation of SDO\\_CSW\\_PROCESS](#)

---

## Documentation of SDO\_CSW\_PROCESS

For a complete description of this package within the context of Oracle Spatial, see SDO\_CSW\_PROCESS in the *Oracle Spatial Developer's Guide*.

The SDO\_GCDR package contains the Oracle Spatial geocoding subprograms, which let you geocode unformatted postal addresses.

- [Documentation of SDO\\_GCDR](#)

---

## Documentation of SDO\_GCDR

For a complete description of this package within the context of Oracle Spatial, see SDO\_GCDR in *Oracle Spatial Developer's Guide*.



The SDO\_GEOM package contains the geometry functions, which can be grouped into the following categories (with examples of each):

- Relationship (True/False) between two objects: RELATE, WITHIN\_DISTANCE
- Validation: VALIDATE\_GEOMETRY\_WITH\_CONTEXT, VALIDATE\_LAYER\_WITH\_CONTEXT
- Single-object operations: SDO\_ARC\_DENSIFY, SDO\_AREA, SDO\_BUFFER, SDO\_CENTROID, SDO\_CONVEXHULL, SDO\_LENGTH, SDO\_MBR, SDO\_POINTONSURFACE
- Two-object operations: SDO\_DISTANCE, SDO\_DIFFERENCE, SDO\_INTERSECTION, SDO\_UNION, SDO\_XOR

This chapter contains the following topic:

- [Documentation of SDO\\_GEOM](#)

---

## Documentation of SDO\_GEOM

For a complete description of this package within the context of Oracle Spatial, see SDO\_GEOM in the *Oracle Spatial Developer's Guide*.

The `SDO_GEOR` package contains functions and procedures for the Oracle Spatial GeoRaster feature, which lets you store, index, query, analyze, and deliver raster image data and its associated spatial vector geometry data and metadata.

- [Documentation of SDO\\_GEOR](#)

---

## Documentation of SDO\_GEOR

For complete description of this package within the context of Oracle Spatial, see SDO\_GEOR in the *Oracle Spatial GeoRaster Developer's Guide*.

---

---

## SDO\_GEOR\_ADMIN

The SDO\_GEOR\_ADMIN package contains subprograms for administrative operations related to GeoRaster.

- [Documentation of SDO\\_GEOR\\_ADMIN](#)

---

## Documentation of SDO\_GEOR\_ADMIN

For a complete description of this package within the context of Oracle Spatial, see SDO\_GEOR\_ADMIN in the *Oracle Spatial GeoRaster Developer's Guide*.

---

---

## SDO\_GEOR\_UTL

The SDO\_GEOR\_UTL package contains utility functions and procedures for the Oracle Spatial GeoRaster feature, including those related to using triggers with GeoRaster data.

- [Documentation of SDO\\_GEOR\\_UTL](#)

---

## Documentation of SDO\_GEOR\_UTL

For complete description of this package within the context of Oracle Spatial, see SDO\_GEOR\_UTL in the *Oracle Spatial GeoRaster Developer's Guide*.



The SDO\_LRS package contains functions that create, modify, query, and convert linear referencing elements.

- [Documentation of SDO\\_LRS](#)

---

## Documentation of SDO\_LRS

For a complete description of this package within the context of Oracle Spatial, see `SDO_LRS` in the *Oracle Spatial Developer's Guide*.

The SDO\_MIGRATE package lets you upgrade geometry tables from previous releases of Oracle Spatial.

- [Documentation of SDO\\_MIGRATE](#)

---

## Documentation of SDO\_MIGRATE

For a complete description of this package within the context of Oracle Spatial, see SDO\_MIGRATE in the *Oracle Spatial Developer's Guide*.

The SDO\_NET package contains functions and procedures for working with data modeled as nodes and links in a network.

- [Documentation of SDO\\_NET](#)

---

## Documentation of SDO\_NET

For a complete description of this package within the context of Oracle Spatial, see SDO\_NET in the *Oracle Spatial Topology and Network Data Models Developer's Guide*.

---

---

## SDO\_NET\_MEM

The SDO\_NET\_MEM package contains functions and procedures for performing editing and analysis operations on network data using a network memory object.

- [Documentation of SDO\\_NET\\_MEM](#)

---

## Documentation of SDO\_NET\_MEM

For a complete description of this package within the context of Oracle Spatial, see SDO\_NET\_MEM in the *Oracle Spatial Topology and Network Data Models Developer's Guide*.



The SDO\_OLS package contains subprograms for Spatial OpenLS support.

- [Documentation of SDO\\_OLS](#)

---

## Documentation of SDO\_OLS

For a complete description of this package within the context of Oracle Spatial, see SDO\_OLS in the *Oracle Spatial Developer's Guide*.

The SDO\_PC\_PKG package contains subprograms to support the use of point clouds in Spatial.

- [Documentation of SDO\\_PC\\_PKG](#)

---

## Documentation of SDO\_PC\_PKG

For a complete description of this package within the context of Oracle Spatial, see SDO\_PC\_PKG in the *Oracle Spatial Developer's Guide*.

The SDO\_SAM package contains functions and procedures for spatial analysis and data mining.

- [Documentation of SDO\\_SAM](#)

---

## Documentation of SDO\_SAM

For a complete description of this package within the context of Oracle Spatial, see SDO\_SAM in the *Oracle Spatial Developer's Guide*.

---

## SDO\_TIN\_PKG

The SDO\_TIN\_PKG package contains subprograms to support the use of triangulated irregular networks (TINs) in Spatial.

- [Documentation of SDO\\_TIN\\_PKG](#)

---

## Documentation of SDO\_TIN\_PKG

For a complete description of this package within the context of Oracle Spatial, see SDO\_TIN\_PKG in the *Oracle Spatial Developer's Guide*.



The SDO\_TOPO package contains subprograms for creating and managing Oracle Spatial topologies.

- [Documentation of SDO\\_TOPO](#)

---

## Documentation of SDO\_TOPO

For a complete description of this package within the context of Oracle Spatial, see SDO\_TOPO in the *Oracle Spatial Topology and Network Data Models Developer's Guide*.

---

---

## SDO\_TOPO\_MAP

The SDO\_TOPO\_MAP package contains subprograms for editing Oracle Spatial topologies using a cache (TopoMap object).

- [Documentation of SDO\\_TOPO\\_MAP](#)

---

## Documentation of SDO\_TOPO\_MAP

For a complete description of this package within the context of Oracle Spatial, see SDO\_TOPO\_MAP in the *Oracle Spatial Topology and Network Data Models Developer's Guide*.

The SDO\_TUNE package contains Spatial tuning functions and procedures.

- [Documentation of SDO\\_TUNE](#)

---

## Documentation of SDO\_TUNE

For complete description of this package within the context of Oracle Spatial, see SDO\_TUNE in the *Oracle Spatial Developer's Guide*.

The SDO\_UTIL package contains the utility functions and procedures for Oracle Spatial.

- [Documentation of SDO\\_UTIL](#)

---

## Documentation of SDO\_UTIL

For complete description of this package within the context of Oracle Spatial, see SDO\_UTIL in the *Oracle Spatial Developer's Guide*.



---

---

## SDO\_WFS\_LOCK

The SDO\_WFS\_LOCK package contains subprograms for WFS support for registering and unregistering feature tables. Registering a feature table enables the table for WFS transaction locking; unregistering a feature table disables the table for WFS transaction locking.

- [Documentation of SDO\\_WFS\\_LOCK](#)

---

## Documentation of SDO\_WFS\_LOCK

For a complete description of this package within the context of Oracle Spatial, see SDO\_WFS\_LOCK in the *Oracle Spatial Developer's Guide*.

---

---

## SDO\_WFS\_PROCESS

The SDO\_WFS\_PROCESS package contains subprograms for various processing operations related to support for Web Feature Services.

- [Documentation of SDO\\_WFS\\_PROCESS](#)

---

## Documentation of SDO\_WFS\_PROCESS

For a complete description of this package within the context of Oracle Spatial, see SDO\_WFS\_PROCESS in the *Oracle Spatial Developer's Guide*.

The SEM\_APIS package contains subprograms for working with the Resource Description Framework (RDF) and Web Ontology Language (OWL) in an Oracle database.

- [Documentation of SEM\\_APIS](#)

---

## Documentation of SEM\_APIS

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM\_APIS in the *Oracle Database Semantic Technologies Developer's Guide*.

The SEM\_PERF package contains subprograms for examining and enhancing the performance of the Resource Description Framework (RDF) and Web Ontology Language (OWL) support in an Oracle database.

- [Documentation of SEM\\_PERF](#)

---

## Documentation of SEM\_PERF

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM\_PERF in the *Oracle Database Semantic Technologies Developer's Guide*.



The SEM\_RDFCTX package contains subprograms for managing extractor policies and semantic indexes created for documents.

- [Documentation of SEM\\_RDFCTX](#)

---

## Documentation of SEM\_RDFCTX

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM\_RDFCTX in the *Oracle Database Semantic Technologies Developer's Guide*.

The SEM\_RDFSA package contains subprograms for providing fine-grained access control to RDF data, using either a virtual private database (VPD) or Oracle Label Security (OLS).

- [Documentation of SEM\\_RDFSA](#)

---

## Documentation of SEM\_RDFSA

For a complete description of this package within the context of Oracle Database semantic technology support, see SEM\_RDFSA in the *Oracle Database Semantic Technologies Developer's Guide*.

The UTL\_COLL package lets PL/SQL programs use collection locators to query and update.

This chapter contains the following topics:

- [Summary of UTL\\_COLL Subprograms](#)

---

## Summary of UTL\_COLL Subprograms

**Table 220–1 UTL\_COLL Package Subprograms**

Subprogram	Description
<a href="#">IS_LOCATOR Function</a> on page 220-3	Determines whether a collection item is actually a locator or not

## IS\_LOCATOR Function

This function determines whether a collection item is actually a locator or not.

### Syntax

```
UTL_COLL.IS_LOCATOR (
  coln IN STANDARD)
  RETURNS BOOLEAN;
```

### Pragmas

Asserts WNDS, WNPS and RNPS pragmas

### Parameters

**Table 220–2 IS\_LOCATOR Function Parameters**

Parameter	Description
coln	Nested table or varray item.

### Return Values

**Table 220–3 IS\_LOCATOR Function Return Values**

Return Value	Description
1	Collection item is indeed a locator.
0	Collection item is not a locator.

### Examples

```
CREATE OR REPLACE TYPE list_t as TABLE OF VARCHAR2(20);
/

CREATE OR REPLACE TYPE phone_book_t AS OBJECT (
  pno number,
  ph list_t );
/

CREATE TABLE phone_book OF phone_book_t
  NESTED TABLE ph STORE AS nt_ph;
CREATE TABLE phone_book1 OF phone_book_t
  NESTED TABLE ph STORE AS nt_ph_1 RETURN LOCATOR;

INSERT INTO phone_book VALUES(1, list_t('650-633-5707','650-323-0953'));
INSERT INTO phone_book1 VALUES(1, list_t('415-555-1212'));

CREATE OR REPLACE PROCEDURE chk_coll IS
  plist list_t;
  plist1 list_t;
BEGIN
  SELECT ph INTO plist FROM phone_book WHERE pno=1;

  SELECT ph INTO plist1 FROM phone_book1 WHERE pno=1;

  IF (UTL_COLL.IS_LOCATOR(plist)) THEN
    DBMS_OUTPUT.PUT_LINE('plist is a locator');
```

```
ELSE
  DBMS_OUTPUT.PUT_LINE('plist is not a locator');
END IF;

IF (UTL_COLL.IS_LOCATOR(plist1)) THEN
  DBMS_OUTPUT.PUT_LINE('plist1 is a locator');
ELSE
  DBMS_OUTPUT.PUT_LINE('plist1 is not a locator');
END IF;

END chk_coll;

SET SERVEROUTPUT ON
EXECUTE chk_coll;
```



The UTL\_COMPRESS package provides a set of data compression utilities.

This chapter contains the following topics:

- [Using UTL\\_COMPRESS](#)
  - Constants
  - Exceptions
  - Operational Notes
- [Summary of UTL\\_COMPRESS Subprograms](#)

## Using UTL\_COMPRESS

- [Constants](#)
- [Exceptions](#)
- [Operational Notes](#)

## Constants

Define max number of handles for piecewise operations:

```
UTLCOMP_MAX_HANDLE  CONSTANT  PLS_INTEGER := 5;
```

## Exceptions

**Table 221–1** *UTL\_COMPRESS Exceptions*

<b>Exception</b>	<b>Description</b>
BUFFER_TOO_SMALL	The compressed representation is too big.
DATA_ERROR	The input or output data stream was found to be an invalid format.
INVALID_ARGUMENT	One of the arguments was an invalid type or value.
INVALID_HANDLE	Invalid handle for piecewise compress or uncompress.
STREAM_ERROR	An error occurred during compression or uncompression of the data stream

## Operational Notes

- It is the caller's responsibility to free the temporary LOB returned by the LZ\* functions with `DBMS_LOB.FREETEMPORARY` call.
- A `BFILE` passed into `LZ_COMPRESS*` or `LZ_UNCOMPRESS*` has to be opened by `DBMS_LOB.FILEOPEN`.
- Under special circumstances (especially if the input has already been compressed) the output produced by one of the `UTL_COMPRESS` subprograms may be the same size, or even slightly larger than, the input.
- The output of the `UTL_COMPRESS` compressed data is compatible with `gzip`(with `-n` option)/`gunzip` on a single file.

---

## Summary of UTL\_COMPRESS Subprograms

**Table 221–2 UTL\_COMPRESS Package Subprograms**

Subprogram	Description
<a href="#">ISOPEN Function</a> on page 221-7	Checks to see if the handle to a piecewise (un)compress context is open or closed
<a href="#">LZ_COMPRESS Functions and Procedures</a> on page 221-8	Compresses data using Lempel-Ziv compression algorithm
<a href="#">LZ_COMPRESS_ADD Procedure</a> on page 221-10	Adds a piece of compressed data
<a href="#">LZ_COMPRESS_CLOSE</a> on page 221-11	Closes and finishes piecewise compress operation
<a href="#">LZ_COMPRESS_OPEN</a> on page 221-12	Initializes a piecewise context that maintains the compress state and data
<a href="#">LZ_UNCOMPRESS Functions and Procedures</a> on page 221-13	Accepts compressed input, verifies it to be a valid and uncompresses it
<a href="#">LZ_UNCOMPRESS_EXTRACT Procedure</a> on page 221-14	Extracts a piece of uncompressed data
<a href="#">LZ_UNCOMPRESS_OPEN Function</a> on page 221-15	Initializes a piecewise context that maintains the uncompress state and data
<a href="#">LZ_UNCOMPRESS_CLOSE Procedure</a> on page 221-16	Closes and finishes the piecewise uncompress

## ISOPEN Function

This function checks to see if the handle to a piecewise (un)compress context is open or closed.

### Syntax

```
UTL_COMPRESS.ISOPEN(  
    handle in binary_integer)  
RETURN BOOLEAN;
```

### Parameters

**Table 221–3 ISOPEN Function Parameters**

Parameter	Description
handle	The handle to a piecewise uncompress context.

### Return Values

TRUE if the given piecewise handle is opened, otherwise FALSE.

### Examples

```
IF (UTL_COMPRESS.ISOPEN(myhandle) = TRUE) then  
    UTL_COMPRESS.LZ_COMPRESS_CLOSE(myhandle, lob_1);  
END IF;
```

Alternatively:

```
IF (UTL_COMPRESS.ISOPEN(myhandle) = TRUE) THEN  
    UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE(myhandle);  
END IF;
```

## LZ\_COMPRESS Functions and Procedures

These functions and procedures compress data using Lempel-Ziv compression algorithm.

### Syntax

This function accept a RAW as input, compress it and return the compressed RAW result and metadata:

```
UTL_COMPRESS.LZ_COMPRESS (
  src      IN      RAW,
  quality  IN      BINARY_INTEGER DEFAULT 6)
RETURN RAW;
```

This function accept a BLOB as input, compress it and returns a temporary BLOB for the compressed data:

```
UTL_COMPRESS.LZ_COMPRESS (
  src      IN      BLOB,
  quality  IN      BINARY_INTEGER DEFAULT 6)
RETURN BLOB;
```

This procedure returns the compressed data into the existing BLOB(dst) which is trimmed to the compressed data size:

```
UTL_COMPRESS.LZ_COMPRESS (
  src      IN      BLOB,
  dst      IN OUT NOCOPY BLOB,
  quality  IN      BINARY_INTEGER DEFAULT 6);
```

This function returns a temporary BLOB for the compressed data:

```
UTL_COMPRESS.LZ_COMPRESS (
  src      IN      BFILE,
  quality  IN      BINARY_INTEGER DEFAULT 6)
RETURN BLOB;
```

This procedure will return the compressed data into the existing BLOB(dst) which is trimmed to the compressed data size:

```
UTL_COMPRESS.LZ_COMPRESS (
  src      IN      BFILE,
  dst      IN OUT NOCOPY BLOB,
  quality  IN      BINARY_INTEGER DEFAULT 6);
```

### Parameters

**Table 221–4 LZ\_COMPRESS Function and Procedures Parameters**

Parameter	Description
src	Data (RAW, BLOB or BFILE) to be compressed.
dst	Destination for compressed data
quality	An integer in the range 1 to 9, 1=fast compression, 9=best compression, default=6



**Usage Notes**

- `quality` is an optional compression tuning value. It allows the `UTL_COMPRESS` user to choose between speed and compression quality, meaning the percentage of reduction in size. A faster compression speed will result in less compression of the data. A slower compression speed will result in more compression of the data. Valid values are [1..9], with 1=fastest and 9=slowest. The default 'quality' value is 6.

## LZ\_COMPRESS\_ADD Procedure

This procedure adds a piece of compressed data.

### Syntax

```
UTL_COMPRESS.LZ_COMPRESS_ADD (  
    handle IN          BINARY_INTEGER,  
    dst    IN OUT NOCOPY BLOB,  
    src    IN          RAW);
```

### Parameters

**Table 221-5 LZ\_COMPRESS\_ADD Procedure Parameters**

Parameter	Description
handle	The handle to a piecewise compress context.
dst	The opened LOB from LZ_COMPRESS_OPEN to store compressed data.
src	The input data to be compressed.

### Exceptions

- `invalid_handle` - out of range invalid or unopened handle.
- `invalid_argument` - NULL handle, src, dst, or invalid dst.

## LZ\_COMPRESS\_CLOSE

This procedure closes and finishes piecewise compress operation.

### Syntax

```
UTL_COMPRESS.LZ_COMPRESS_CLOSE (  
  handle IN          BINARY_INTEGER,  
  dst      IN OUT NOCOPY BLOB);
```

### Parameters

**Table 221–6** LZ\_COMPRESS\_CLOSE Procedure Parameters

Parameter	Description
handle	The handle to a piecewise compress context.
dst	The opened LOB from LZ_COMPRESS_OPEN to store compressed data.

### Exceptions

- `invalid_handle` - out of range invalid or uninitialized handle.
- `invalid_argument` - NULL handle, dst, or invalid dst.

## LZ\_COMPRESS\_OPEN

This function initializes a piecewise context that maintains the compress state and data.

### Syntax

```
UTL_COMPRESS.LZ_COMPRESS_OPEN (  
    dst          IN OUT NOCOPY BLOB,  
    quality      IN          BINARY_INTEGER DEFAULT 6)  
RETURN BINARY_INTEGER;
```

### Parameters

**Table 221–7 LZ\_COMPRESS\_OPEN Function Parameters**

Parameter	Description
dst	User supplied LOB to store compressed data.
quality	Speed versus efficiency of resulting compressed output. <ul style="list-style-type: none"><li>Valid values are the range 1..9, with a default value of 6.</li><li>1=fastest compression, 9=slowest compression and best compressed file size.</li></ul>

### Return Values

A handle to an initialized piecewise compress context.

### Exceptions

- `invalid_handle` - invalid handle, too many open handles.
- `invalid_argument` - NULL `dst` or invalid quality specified.

### Usage Notes

Close the opened handle with `LZ_COMPRESS_CLOSE`

- once the piecewise compress is completed
- in the event of an exception in the middle of process because lack of doing so will cause these handles to leak.

## LZ\_UNCOMPRESS Functions and Procedures

This procedure accepts as input a RAW, BLOB or BFILE compressed string, verifies it to be a valid compressed value, uncompresses it using Lempel-Ziv compression algorithm, and returns the uncompressed RAW or BLOB result.

### Syntax

This function returns uncompressed data as RAW:

```
UTL_COMPRESS.LZ_UNCOMPRESS (
    src IN RAW)
RETURN RAW;
```

This function returns uncompressed data as a temporary BLOB:

```
UTL_COMPRESS.LZ_UNCOMPRESS (
    src IN BLOB)
RETURN BLOB;
```

This procedure returns the uncompressed data into the existing BLOB(dst), which will be trimmed to the uncompressed data size:

```
UTL_COMPRESS.LZ_UNCOMPRESS (
    src IN BLOB,
    dst IN OUT NOCOPY BLOB);
```

This function returns a temporary BLOB for the uncompressed data:

```
UTL_COMPRESS.LZ_UNCOMPRESS (
    src IN BFILE)
RETURN BLOB;
```

This procedure returns the uncompressed data into the existing BLOB(dst). The original dst data will be overwritten.

```
UTL_COMPRESS.LZ_UNCOMPRESS (
    src IN BFILE,
    dst IN OUT NOCOPY BLOB);
```

### Parameters

**Table 221–8 LZ\_UNCOMPRESS Function and Procedures Parameters**

Parameter	Description
src	Compressed data.
dst	Destination for uncompressed data.

## LZ\_UNCOMPRESS\_EXTRACT Procedure

This procedure extracts a piece of uncompressed data.

### Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_EXTRACT(  
  handle IN          BINARY_INTEGER,  
  dst    OUT NOCOPY RAW);
```

### Parameters

**Table 221–9 LZ\_UNCOMPRESS\_EXTRACT Function Parameters**

Parameter	Description
handle	The handle to a piecewise uncompress context.
dst	The uncompressed data.

### Exceptions

- no\_data\_found - finished uncompress.
- invalid\_handle - out of range invalid or uninitialized handle.
- invalid\_argument - NULL handle.

## LZ\_UNCOMPRESS\_OPEN Function

This function initializes a piecewise context that maintains the uncompress state and data.

### Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_OPEN(  
    src IN BLOB)  
RETURN BINARY_INTEGER;
```

### Parameters

**Table 221–10 LZ\_UNCOMPRESS\_OPEN Function Parameters**

Parameter	Description
src	The input data to be uncompressed.

### Return Values

A handle to an initialized piecewise compress context.

### Exceptions

- `invalid_handle` - invalid handle, too many open handles.
- `invalid_argument` - NULL src.

### Usage Notes

Close the opened handle with `LZ_UNCOMPRESS_CLOSE`

- once the piecewise uncompress is completed
- in the event of an exception in the middle of process because lack of doing so will cause these handles to leak.

## LZ\_UNCOMPRESS\_CLOSE Procedure

This procedure closes and finishes the piecewise uncompress.

### Syntax

```
UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE(  
    handle IN BINARY_INTEGER);
```

### Parameters

**Table 221–11 LZ\_UNCOMPRESS\_CLOSE Procedure Parameters**

Parameter	Description
handle	The handle to a piecewise uncompress context.

### Exceptions

- `invalid_handle` - out of range invalid or uninitialized handle.
- `invalid_argument` - NULL handle.



The UTL\_ENCODE package provides functions that encode RAW data into a standard encoded format so that the data can be transported between hosts. You can use UTL\_ENCODE functions to encode the body of email text. The package also contains the decode counterpart functions of the encode functions. The functions follow published standards for encoding to accommodate non-Oracle utilities on the sending or receiving ends.

This chapter contains the following topic:

- [Summary of UTL\\_ENCODE Subprograms](#)

---

## Summary of UTL\_ENCODE Subprograms

**Table 222–1 UTL\_ENCODE Package Subprograms**

Subprogram	Description
<a href="#">BASE64_DECODE Function</a> on page 222-3	Reads the base 64-encoded RAW input string and decodes it to its original RAW value
<a href="#">BASE64_ENCODE Function</a> on page 222-4	Encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string
<a href="#">MIMEHEADER_DECODE Function</a> on page 222-5	Decodes a string from mime header format
<a href="#">MIMEHEADER_ENCODE Function</a> on page 222-6	Encodes a string into mime header format
<a href="#">QUOTED_PRINTABLE_DECODE Function</a> on page 222-7	Reads the varchar2 quoted printable format input string and decodes it to the corresponding RAW string
<a href="#">QUOTED_PRINTABLE_ENCODE Function</a> on page 222-8	Reads the RAW input string and encodes it to the corresponding quoted printable format string
<a href="#">TEXT_DECODE Function</a> on page 222-9	Decodes a character set sensitive text string
<a href="#">TEXT_ENCODE Function</a> on page 222-10	Encodes a character set sensitive text string
<a href="#">UUDECODE Function</a> on page 222-11	Reads the RAW uuencode format input string and decodes it to the corresponding RAW string
<a href="#">UUENCODE Function</a> on page 222-12	Reads the RAW input string and encodes it to the corresponding uuencode format string

## BASE64\_DECODE Function

This function reads the base 64-encoded RAW input string and decodes it to its original RAW value.

### Syntax

```
UTL_ENCODE.BASE64_DECODE (
    r IN RAW)
RETURN RAW;
```

### Pragmas

```
pragma RESTRICT_REFERENCES (base64_decode, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 222–2 BASE64\_DECODE Function Parameters**

Parameter	Description
r	The RAW string containing base 64-encoded data. There are no defaults or optional parameters.

### Return Values

**Table 222–3 BASE64\_DECODE Function Return Values**

Return	Description
RAW	Contains the decoded string

## BASE64\_ENCODE Function

This function encodes the binary representation of the RAW value into base 64 elements and returns it in the form of a RAW string.

### Syntax

```
UTL_ENCODE.BASE64_ENCODE (  
    r IN RAW)  
RETURN RAW;
```

### Pragmas

```
pragma RESTRICT_REFERENCES (base64_encode, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 222-4** *BASE64\_ENCODE Function Parameters*

Parameter	Description
r	The RAW value to be encoded. There are no defaults or optional parameters.

### Return Values

**Table 222-5** *BASE64\_ENCODE Function Return Values*

Return	Description
RAW	Contains the encoded base 64 elements

## MIMEHEADER\_DECODE Function

This function accepts as input an "encoded word" of the form:

```
=?<charset>?<encoding>?<encoded text>?=
=?ISO-8859-1?Q?Here is some encoded text?=
```

The <encoded text> is encapsulated in mime header tags which give the MIMEHEADER\_DECODE function information about how to decode the string. The mime header metadata tags are stripped from the input string and the <encoded text> is converted to the base database character set as follows:

- If this is a UTF16 platform, convert the encoded text from UTF16 to ASCII
- If this is an EBCDIC platform, convert the encoded text from EBCDIC to ASCII
- If this is an ASCII or UTF8 platform, no conversion needed

The string is decoded using either quoted-printable or base64 decoding, as specified by the <encoding> metadata tag in the encoded word. The resulting converted and decoded text is returned to the caller as a VARCHAR2 string.

### Syntax

```
UTL_ENCODE.MIMEHEADER_DECODE (
  buf    IN  VARCHAR2 CHARACTER SET ANY_CS)
RETURN data VARCHAR2 CHARACTER SET buf%CHARSET;
```

### Parameters

**Table 222–6** *MIMEHEADER\_DECODE Function Parameters*

Parameter	Description
buf	The encoded text data with mime header format tags.

### Return Values

**Table 222–7** *MIMEHEADER\_DECODE Function Return Values*

Return	Description
data	The encoded text data with mime header format tags

### Examples

```
v2:=utl_encode.mimeheader_decode('=?ISO-8859-1?Q?Here is some encoded text?');
```

## MIMEHEADER\_ENCODE Function

This function returns as an output an "encoded word" of the form:

```
=?<charset>?<encoding>?<encoded text>?=
=?ISO-8859-1?Q?Here is some text?=-
```

The `buf` input parameter is the text to be encoded and becomes the `<encoded text>`.

The `<encoding>` value is either "Q" or "B" for quoted-printable encode or base64 encoding respectively. The `ENCODING` input parameter accepts as valid values `UTL_ENCODE.QUOTED_PRINTABLE` or `UTL_ENCODE.BASE64` or `NULL`. If `NULL`, quoted-printable encoding is selected as a default value.

The `<charset>` value is specified as the input parameter `encode_charset`. If `NULL`, the database character set is selected as a default value.

The mimeheader encoding process includes conversion of the `buf` input string to the character set specified by the `encode_charset` parameter. The converted string is encoded to either quoted-printable or base64 encoded format. The mime header tags are appended and prepended.

Finally, the string is converted to the base character set of the database:

- If this is a UTF16 platform, convert the encoded text to UTF16
- If this is an EBCDIC platform, convert the encoded text to EBCDIC
- If this is an ASCII or UTF8 platform, no conversion needed.

### Syntax

```
UTL_ENCODE.MIMEHEADER_ENCODE (
    buf          IN VARCHAR2 CHARACTER SET ANY_CS,
    encode_charset IN VARCHAR2 DEFAULT NULL,
    encoding     IN PLS_INTEGER DEFAULT NULL)
RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```

### Parameters

**Table 222–8** *MIMEHEADER\_ENCODE Function Parameters*

Parameter	Description
<code>buf</code>	The text data.
<code>encode_charset</code>	The target character set.
<code>encoding</code>	The encoding format. Valid values are <code>UTL_ENCODE.BASE64</code> , <code>UTL_ENCODE.QUOTED_PRINTABLE</code> and <code>NULL</code> .

### Return Values

**Table 222–9** *MIMEHEADER\_ENCODE Function Return Values*

Return	Description
<code>string</code>	A <code>VARCHAR2</code> encoded string with mime header format tags.

## QUOTED\_PRINTABLE\_DECODE Function

This function reads the `varchar2` quoted printable format input string and decodes it to the corresponding `RAW` string.

### Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_DECODE (
    r IN RAW)
RETURN RAW;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_decode, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 222–10 QUOTED\_PRINTABLE\_DECODE Function Parameters**

Parameters	Description
<code>r</code>	The <code>RAW</code> string containing a quoted printable data string. There are no defaults or optional parameters.

### Return Values

**Table 222–11 QUOTED\_PRINTABLE\_DECODE Function Return Values**

Return	Description
<code>RAW</code>	The decoded string

## QUOTED\_PRINTABLE\_ENCODE Function

This function reads the RAW input string and encodes it to the corresponding quoted printable format string.

### Syntax

```
UTL_ENCODE.QUOTED_PRINTABLE_ENCODE (  
    r IN RAW)  
RETURN RAW;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(quoted_printable_encode, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 222–12 QUOTED\_PRINTABLE\_ENCODE Function Parameters**

Parameter	Description
r	The RAW string. There are no defaults or optional parameters.

### Return Values

**Table 222–13 QUOTED\_PRINTABLE\_ENCODE Function Return Values**

Return	Description
RAW	Contains the quoted printable string



## TEXT\_DECODE Function

This function converts the input text to the target character set as specified by the `encode_charset` parameter, if not `NULL`. The encoded text is converted to the base character set of database, as follows:

- If this is a UTF16 platform, convert the encoded text from UTF16 to ASCII
- If this is an EBCDIC platform, convert the encoded text from EBCDIC to ASCII
- If this is an ASCII or UTF8 platform, no conversion needed

You can decode from either quoted-printable or base64 format, with regard to each encoding parameter. If `NULL`, quoted-printable is selected as a default decoding format. If `encode_charset` is not `NULL`, you convert the string from the specified character set to the database character set. The resulting decoded and converted text string is returned to the caller.

### Syntax

```
UTL_ENCODE.TEXT_DECODE(
  buf          IN VARCHAR2 CHARACTER SET ANY_CS,
  encode_charset IN VARCHAR2 DEFAULT NULL,
  encoding     IN PLS_INTEGER DEFAULT NULL)
RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```

### Parameters

**Table 222–14** TEXT\_DECODE Function Parameters

Parameter	Description
<code>buf</code>	The encoded text data.
<code>encode_charset</code>	The source character set.
<code>encoding</code>	The encoding format. Valid values are <code>UTL_ENCODE.BASE64</code> , <code>UTL_ENCODE.QUOTED_PRINTABLE</code> and <code>NULL</code> .

### Return Values

**Table 222–15** QUOTED\_PRINTABLE\_ENCODE Function Return Values

Return	Description
<code>string</code>	A <code>VARCHAR2</code> decoded text string.

### Examples

```
v2:=UTL_ENCODE.TEXT_DECODE(
  'Here is some text',
  WE8ISO8859P1,
  UTL_ENCODE.BASE64);
```

## TEXT\_ENCODE Function

This function converts the input text to the target character set as specified by the `encode_charset` parameter, if not NULL. The text is encoded to either base64 or quoted-printable format, as specified by the `encoding` parameter. Quoted-printable is selected as a default if `ENCODING` is NULL.

The encoded text is converted to the base character set of the database:

- If this is a UTF16 platform, convert the encoded text to UTF16
- If this is an EBCDIC platform, convert the encoded text to EBCDIC
- If this is an ASCII or UTF8 platform, no conversion needed

The resulting encoded and converted text string is returned to the caller.

### Syntax

```
UTL_ENCODE.TEXT_ENCODE (
    buf          IN VARCHAR2 CHARACTER SET ANY_CS,
    encode_charset IN VARCHAR2 DEFAULT NULL,
    encoding     IN PLS_INTEGER DEFAULT NULL)
RETURN string VARCHAR2 CHARACTER SET buf%CHARSET;
```

### Parameters

**Table 222–16 TEXT\_ENCODE Function Parameters**

Parameter	Description
<code>buf</code>	The text data.
<code>encode_charset</code>	The target character set.
<code>encoding</code>	The encoding format. Valid values are <code>UTL_ENCODE.BASE64</code> , <code>UTL_ENCODE.QUOTED_PRINTABLE</code> and <code>NULL</code>

### Return Values

**Table 222–17 TEXT\_ENCODE Function Return Values**

Return	Description
<code>string</code>	A <code>VARCHAR2</code> encoded string with mime header format tags.

### Examples

```
v2:=utl_encode.text_encode(
    'Here is some text',
    'WE8ISO8859P1',
    UTL_ENCODE.BASE64);
```

## UUDECODE Function

This function reads the RAW uuencode format input string and decodes it to the corresponding RAW string. See "[UUENCODE Function](#)" on page 222-12 for discussion of the cumulative nature of UUENCODE and UUDECODE for data streams.

### Syntax

```
UTL_ENCODE.UUDECODE (
    r IN RAW)
RETURN RAW;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(uudecode, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 222–18 UUDECODE Function Parameters**

Parameter	Description
r	The RAW string containing the uuencoded data string. There are no defaults or optional parameters.

### Return Values

**Table 222–19 UUDECODE Function Return Values**

Return	Description
RAW	The decoded RAW string

## UUENCODE Function

This function reads the RAW input string and encodes it to the corresponding uuencode format string. The output of this function is cumulative, in that it can be used to encode large data streams, by splitting the data stream into acceptably sized RAW values, encoded, and concatenated into a single encoded string.

### Syntax

```
UTL_ENCODE.UUENCODE (
  r          IN RAW,
  type       IN PLS_INTEGER DEFAULT 1,
  filename   IN VARCHAR2 DEFAULT NULL,
  permission IN VARCHAR2 DEFAULT NULL) RETURN RAW;
```

### Pragmas

```
pragma RESTRICT_REFERENCES(uuencode, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 222–20 UUENCODE Function Parameters**

Parameter	Description
r	RAW string
type	Optional number parameter containing the type of uuencoded output. Options: complete—a defined PL/SQL constant with a value of 1. (default) header_piece ...middle_piece ...end_piece
filename	Optional varchar2 parameter containing the uuencode filename; the default is uuencode.txt
permission	Optional varchar2 parameter containing the permission mode; the default is 0 (a text string zero).

### Return Values

**Table 222–21 UUENCODE Function Return Values**

Return	Description
RAW	Contains the uuencode format string

With the UTL\_FILE package, PL/SQL programs can read and write operating system text files. UTL\_FILE provides a restricted version of operating system stream file I/O.

This chapter contains the following topics:

- [Using UTL\\_FILE](#)
  - Security Model
  - Operational Notes
  - Rules and Limits
  - Exceptions
  - Examples
- [Data Structures](#)
- [Summary of UTL\\_FILE Subprograms](#)

## Using UTL\_FILE

- [Security Model](#)
- [Operational Notes](#)
- [Rules and Limits](#)
- [Exceptions](#)
- [Examples](#)

## Security Model

The set of files and directories that are accessible to the user through UTL\_FILE is controlled by a number of factors and database parameters. Foremost of these is the set of directory objects that have been granted to the user. The nature of directory objects is discussed in the *Oracle Database SQL Language Reference*.

Assuming the user has both READ and WRITE access to the directory object USER\_DIR, the user can open a file located in the operating system directory described by USER\_DIR, but not in subdirectories or parent directories of this directory.

Lastly, the client (text I/O) and server implementations are subject to operating system file permission checking.

UTL\_FILE provides file access both on the client side and on the server side. When run on the server, UTL\_FILE provides access to all operating system files that are accessible from the server. On the client side, as in the case for Forms applications, UTL\_FILE provides access to operating system files that are accessible from the client.

In the past, accessible directories for the UTL\_FILE functions were specified in the initialization file using the UTL\_FILE\_DIR parameter. However, UTL\_FILE\_DIR access is no longer recommended. Oracle recommends that you instead use the directory object feature, which replaces UTL\_FILE\_DIR. Directory objects offer more flexibility and granular control to the UTL\_FILE application administrator, can be maintained dynamically (that is, without shutting down the database), and are consistent with other Oracle tools. CREATE ANY DIRECTORY privilege is granted only to SYS and SYSTEM by default.

---

---

**Note:** Use the CREATE DIRECTORY feature instead of UTL\_FILE\_DIR for directory access verification.

---

---

Note that neither hard nor symbolic links are supported.

On UNIX systems, the owner of a file created by the FOPEN function is the owner of the shadow process running the instance. Normally, this owner is ORACLE. Files created using FOPEN are always writable and readable using the UTL\_FILE subprograms. However, non-privileged operating system users who need to read these files outside of PL/SQL may need access from a system administrator.

---

---

**Caution:** The privileges needed to access files in a directory object are operating system specific. UTL\_FILE directory object privileges give you read and write access to all files within the specified directory.

---

---

## Operational Notes

The file location and file name parameters are supplied to the `FOPEN` function as separate strings, so that the file location can be checked against the list of accessible directories as specified by the `ALL_DIRECTORIES` view of accessible directory objects. Together, the file location and name must represent a legal filename on the system, and the directory must be accessible. A subdirectory of an accessible directory is not necessarily also accessible; it too must be specified using a complete path name matching an `ALL_DIRECTORIES` object.

`UTL_FILE` implicitly interprets line terminators on read requests, thereby affecting the number of bytes returned on a `GET_LINE` call. For example, the `len` parameter of `UTL_FILE.GET_LINE` specifies the requested number of bytes of character data. The number of bytes actually returned to the user will be the lesser of:

- The `GET_LINE len` parameter, or
- The number of bytes until the next line terminator character, or
- The `max_linesize` parameter specified by `UTL_FILE.FOPEN`

The `FOPEN max_linesize` parameter must be a number in the range 1 and 32767. If unspecified, Oracle supplies a default value of 1024. The `GET_LINE len` parameter must be a number in the range 1 and 32767. If unspecified, Oracle supplies the default value of `max_linesize`. If `max_linesize` and `len` are defined to be different values, then the lesser value takes precedence.

`UTL_FILE.GET_RAW` ignores line terminators.

`UTL_FILE` expects that files opened by `UTL_FILE.FOPEN` in text mode are encoded in the database character set. It expects that files opened by `UTL_FILE.FOPEN_NCHAR` in text mode are encoded in the UTF8 character set. If an opened file is not encoded in the expected character set, the result of an attempt to read the file is indeterminate. When data encoded in one character set is read and Globalization Support is told (such as by means of `NLS_LANG`) that it is encoded in another character set, the result is indeterminate. If `NLS_LANG` is set, it should therefore be the same as the database character set.



## Rules and Limits

Operating system-specific parameters, such as C-shell environment variables under UNIX, cannot be used in the file location or file name parameters.

UTL\_FILE I/O capabilities are similar to standard operating system stream file I/O (OPEN, GET, PUT, CLOSE) capabilities, but with some limitations. For example, you call the FOPEN function to return a file handle, which you use in subsequent calls to GET\_LINE or PUT to perform stream I/O to a file. When file I/O is done, you call FCLOSE to complete any output and free resources associated with the file.

---

---

**Note:** The UTL\_FILE package is similar to the client-side TEXT\_IO package currently provided by Oracle Procedure Builder. Restrictions for a server implementation require some API differences between UTL\_FILE and TEXT\_IO. In PL/SQL file I/O, errors are returned using PL/SQL exceptions.

---

---

## Exceptions

**Table 223–1 UTL\_FILE Package Exceptions**

Exception Name	Description
INVALID_PATH	File location is invalid.
INVALID_MODE	The <code>open_mode</code> parameter in <code>FOPEN</code> is invalid.
INVALID_FILEHANDLE	File handle is invalid.
INVALID_OPERATION	File could not be opened or operated on as requested.
READ_ERROR	Destination buffer too small, or operating system error occurred during the read operation
WRITE_ERROR	Operating system error occurred during the write operation.
INTERNAL_ERROR	Unspecified PL/SQL error
CHARSETMISMATCH	A file is opened using <code>FOPEN_NCHAR</code> , but later I/O operations use nonchar functions such as <code>PUTF</code> or <code>GET_LINE</code> .
FILE_OPEN	The requested operation failed because the file is open.
INVALID_MAXLINESIZE	The <code>MAX_LINESIZE</code> value for <code>FOPEN()</code> is invalid; it should be within the range 1 to 32767.
INVALID_FILENAME	The filename parameter is invalid.
ACCESS_DENIED	Permission to access to the file location is denied.
INVALID_OFFSET	Causes of the <code>INVALID_OFFSET</code> exception: <ul style="list-style-type: none"> <li>■ <code>ABSOLUTE_OFFSET = NULL</code> and <code>RELATIVE_OFFSET = NULL</code>, or</li> <li>■ <code>ABSOLUTE_OFFSET &lt; 0</code>, or</li> <li>■ Either offset caused a seek past the end of the file</li> </ul>
DELETE_FAILED	The requested file delete operation failed.
RENAME_FAILED	The requested file rename operation failed.

Procedures in `UTL_FILE` can also raise predefined PL/SQL exceptions such as `NO_DATA_FOUND` or `VALUE_ERROR`.

## Examples

### Example 1

---



---

**Note:** The examples are UNIX-specific.

---



---

Given the following:

```
SQL> CREATE DIRECTORY log_dir AS '/appl/gl/log';
SQL> GRANT READ ON DIRECTORY log_dir TO DBA;
SQL> GRANT WRITE ON DIRECTORY log_dir TO DBA;
```

```
SQL> CREATE DIRECTORY USER_DIR AS '/appl/gl/user';
SQL> GRANT READ ON DIRECTORY USER_DIR TO PUBLIC;
SQL> GRANT WRITE ON DIRECTORY USER_DIR TO PUBLIC;
```

The following file locations and filenames are valid and accessible as follows:

File Location	Filename	READ and WRITE
/appl/gl/log	L12345.log	Users with DBA privilege
/appl/gl/user	u12345.tmp	All users

The following file locations and filenames are invalid:

File Location	Filename	Invalid Because
/appl/gl/log/backup	L12345.log	# subdirectories are not accessible
/APPL/gl/log	L12345.log	# directory strings must follow case sensitivity rules as required by the O/S
/appl/gl/log	backup/L1234.log	# filenames may not include portions of directory paths
/user/tmp	L12345.log	# no corresponding CREATE DIRECTORY command has been issued

### Example 2

```
DECLARE
  V1 VARCHAR2(32767);
  F1 UTL_FILE.FILE_TYPE;
BEGIN
  -- In this example MAX_LINESIZE is less than GET_LINE's length request
  -- so the number of bytes returned will be 256 or less if a line terminator is
  -- seen.
  F1 := UTL_FILE.FOPEN('USER_DIR', 'u12345.tmp', 'R', 256);
  UTL_FILE.GET_LINE(F1, V1, 32767);
  UTL_FILE.FCLOSE(F1);

  -- In this example, FOPEN's MAX_LINESIZE is NULL and defaults to 1024,
  -- so the number of bytes returned will be 1024 or less if a line terminator is
  -- seen.
  F1 := UTL_FILE.FOPEN('USER_DIR', 'u12345.tmp', 'R');
```

```
UTL_FILE.GET_LINE(F1,V1,32767);
UTL_FILE.FCLOSE(F1);
```

-- In this example, GET\_LINE doesn't specify a number of bytes, so it defaults to  
to  
-- the same value as FOPEN's MAX\_LINESIZE which is NULL in this case and  
defaults to 1024.  
-- So the number of bytes returned will be 1024 or less if a line terminator is  
seen.

```
F1 := UTL_FILE.FOPEN('USER_DIR','u12345.tmp','R');
UTL_FILE.GET_LINE(F1,V1);
UTL_FILE.FCLOSE(F1);
END;
```

## Data Structures

---

The `UTL_FILE` package defines a `RECORD` type.

### Record Types

- [FILETYPE Record Type](#)

## FILETYPE Record Type

The contents of `FILE_TYPE` are private to the `UTL_FILE` package. You should not reference or change components of this record.

```
TYPE file_type IS RECORD (  
    id          BINARY_INTEGER,  
    datatype    BINARY_INTEGER,  
    byte_mode   BOOLEAN);
```

### Fields

**Table 223–2** *FILE\_TYPE* Fields

Field	Description
<code>id</code>	A numeric value indicating the internal file handle number
<code>datatype</code>	Indicates whether the file is a CHAR file, Nchar file or other (binary)
<code>byte_mode</code>	Indicates whether the file was open as a binary file, or as a text file

---

---

**Caution:** Oracle does not guarantee the persistence of `FILE_TYPE` values between database sessions or within a single session. Attempts to clone file handles or use dummy file handles may have indeterminate outcomes.

---

---

---

## Summary of UTL\_FILE Subprograms

**Table 223–3 UTL\_FILE Subprograms**

Subprogram	Description
<a href="#">FCLOSE Procedure</a> on page 223-13	Closes a file
<a href="#">FCLOSE_ALL Procedure</a> on page 223-14	Closes all open file handles
<a href="#">FCOPY Procedure</a> on page 223-15	Copies a contiguous portion of a file to a newly created file
<a href="#">FFLUSH Procedure</a> on page 223-16	Physically writes all pending output to a file
<a href="#">FGETATTR Procedure</a> on page 223-17	Reads and returns the attributes of a disk file
<a href="#">FGETPOS Function</a> on page 223-18	Returns the current relative offset position within a file, in bytes
<a href="#">FOPEN Function</a> on page 223-19	Opens a file for input or output
<a href="#">FOPEN_NCHAR Function</a> on page 223-21	Opens a file in Unicode for input or output
<a href="#">FREMOVE Procedure</a> on page 223-22	Deletes a disk file, assuming that you have sufficient privileges
<a href="#">FRENAME Procedure</a> on page 223-23	Renames an existing file to a new name, similar to the UNIX mv function
<a href="#">FSEEK Procedure</a> on page 223-24	Adjusts the file pointer forward or backward within the file by the number of bytes specified
<a href="#">GET_LINE Procedure</a> on page 223-25	Reads text from an open file
<a href="#">GET_LINE_NCHAR Procedure</a> on page 223-26	Reads text in Unicode from an open file
<a href="#">GET_RAW Procedure</a> on page 223-27	Reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read
<a href="#">IS_OPEN Function</a> on page 223-28	Determines if a file handle refers to an open file
<a href="#">NEW_LINE Procedure</a> on page 223-29	Writes one or more operating system-specific line terminators to a file
<a href="#">PUT Procedure</a> on page 223-30	Writes a string to a file
<a href="#">PUT_LINE Procedure</a> on page 223-31	Writes a line to a file, and so appends an operating system-specific line terminator
<a href="#">PUT_LINE_NCHAR Procedure</a> on page 223-32	Writes a Unicode line to a file
<a href="#">PUT_NCHAR Procedure</a> on page 223-33	Writes a Unicode string to a file
<a href="#">PUTF Procedure</a> on page 223-34	A PUT procedure with formatting

**Table 223-3 (Cont.) UTL\_FILE Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">PUTF_NCHAR Procedure</a> on page 223-36	A <code>PUT_NCHAR</code> procedure with formatting, and writes a Unicode string to a file, with formatting
<a href="#">PUT_RAW Procedure</a> on page 223-37	Accepts as input a <code>RAW</code> data value and writes the value to the output buffer



## FCLOSE Procedure

This procedure closes an open file identified by a file handle.

### Syntax

```
UTL_FILE.FCLOSE (  
    file IN OUT FILE_TYPE);
```

### Parameters

**Table 223–4 FCLOSE Procedure Parameters**

Parameter	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call

### Usage Notes

If there is buffered data yet to be written when FCLOSE runs, then you may receive a WRITE\_ERROR exception when closing a file.

### Exceptions

```
WRITE_ERROR  
INVALID_FILEHANDLE
```

## FCLOSE\_ALL Procedure

This procedure closes all open file handles for the session. This should be used as an emergency cleanup procedure, for example, when a PL/SQL program exits on an exception.

### Syntax

```
UTL_FILE.FCLOSE_ALL;
```

### Usage Notes

---

---

**Note:** FCLOSE\_ALL does not alter the state of the open file handles held by the user. This means that an IS\_OPEN test on a file handle after an FCLOSE\_ALL call still returns TRUE, even though the file has been closed. No further read or write operations can be performed on a file that was open before an FCLOSE\_ALL.

---

---

### Exceptions

```
WRITE_ERROR
```

## FCOPY Procedure

This procedure copies a contiguous portion of a file to a newly created file. By default, the whole file is copied if the `start_line` and `end_line` parameters are omitted. The source file is opened in read mode. The destination file is opened in write mode. A starting and ending line number can optionally be specified to select a portion from the center of the source file for copying.

### Syntax

```
UTL_FILE.FCOPY (
  src_location      IN VARCHAR2,
  src_filename     IN VARCHAR2,
  dest_location    IN VARCHAR2,
  dest_filename    IN VARCHAR2,
  start_line       IN BINARY_INTEGER DEFAULT 1,
  end_line         IN BINARY_INTEGER DEFAULT NULL);
```

### Parameters

**Table 223–5** *FCOPY Procedure Parameters*

Parameters	Description
<code>src_location</code>	Directory location of the source file, a <code>DIRECTORY_NAME</code> from the <code>ALL_DIRECTORIES</code> view (case sensitive)
<code>src_filename</code>	Source file to be copied
<code>dest_location</code>	Destination directory where the destination file is created
<code>dest_filename</code>	Destination file created from the source file
<code>start_line</code>	Line number at which to begin copying. The default is 1 for the first line
<code>end_line</code>	Line number at which to stop copying. The default is <code>NULL</code> , signifying end of file

### Exceptions

```
INVALID_FILENAME
INVALID_PATH
INVALID_OPERATION
INVALID_OFFSET
READ_ERROR
WRITE_ERROR
```

## FFLUSH Procedure

FFLUSH physically writes pending data to the file identified by the file handle. Normally, data being written to a file is buffered. The FFLUSH procedure forces the buffered data to be written to the file. The data must be terminated with a newline character.

Flushing is useful when the file must be read while still open. For example, debugging messages can be flushed to the file so that they can be read immediately.

### Syntax

```
UTL_FILE.FFLUSH (  
    file IN FILE_TYPE);
```

### Parameters

**Table 223–6 FFLUSH Procedure Parameters**

Parameters	Description
file	Active file handle returned by an FOPEN or FOPEN_NCHAR call

### Exceptions

```
INVALID_FILENAME  
INVALID_MAXLINESIZE  
INVALID_OPERATION  
WRITE_ERROR
```

## FGETATTR Procedure

This procedure reads and returns the attributes of a disk file.

### Syntax

```
UTL_FILE.FGETATTR(
  location      IN VARCHAR2,
  filename      IN VARCHAR2,
  fexists       OUT BOOLEAN,
  file_length   OUT NUMBER,
  block_size    OUT BINARY_INTEGER);
```

### Parameters

**Table 223–7 FGETATTR Procedure Parameters**

Parameters	Description
location	Directory location of the source file, a DIRECTORY_NAME from the ALL_DIRECTORIES view (case sensitive)
filename	Name of the file to be examined
fexists	A BOOLEAN for whether or not the file exists
file_length	Length of the file in bytes. NULL if file does not exist.
block_size	File system block size in bytes. NULL if the file does not exist.

### Exceptions

```
INVALID_PATH
INVALID_FILENAME
INVALID_OPERATION
READ_ERROR
ACCESS_DENIED
```

## FGETPOS Function

This function returns the current relative offset position within a file, in bytes.

### Syntax

```
UTL_FILE.FGETPOS (  
    file IN FILE_TYPE)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 223–8 FGETPOS Parameters**

Parameters	Description
file	Directory location of the source file

### Return Values

FGETPOS returns the relative offset position for an open file, in bytes. It raises an exception if the file is not open. It returns 0 for the beginning of the file.

### Exceptions

INVALID\_FILEHANDLE  
INVALID\_OPERATION  
READ\_ERROR

### Usage Notes

If file is opened for byte mode operations, then the INVALID\_OPERATION exception is raised.

## FOPEN Function

This function opens a file. You can specify the maximum line size and have a maximum of 50 files open simultaneously. See also [FOPEN\\_NCHAR Function](#) on page 223-21.

### Syntax

```
UTL_FILE.FOPEN (
  location      IN VARCHAR2,
  filename      IN VARCHAR2,
  open_mode     IN VARCHAR2,
  max_linesize IN BINARY_INTEGER DEFAULT 1024)
RETURN FILE_TYPE;
```

### Parameters

**Table 223–9 FOPEN Function Parameters**

Parameter	Description
location	Directory location of file. This string is a directory object name and must be specified in upper case. Read privileges must be granted on this directory object for the UTL_FILE user to run FOPEN.
filename	File name, including extension (file type), without directory path. If a directory path is given as a part of the filename, it is ignored by FOPEN. On Unix, the filename cannot end with /.
open_mode	Specifies how the file is opened. Modes include: <ul style="list-style-type: none"> <li>▪ r -- read text</li> <li>▪ w -- write text</li> <li>▪ a -- append text</li> <li>▪ rb -- read byte mode</li> <li>▪ wb -- write byte mode</li> <li>▪ ab -- append byte mode</li> </ul> If you try to open a file specifying 'a' or 'ab' for open_mode but the file does not exist, the file is created in write mode.
max_linesize	Maximum number of characters for each line, including the newline character, for this file (minimum value 1, maximum value 32767). If unspecified, Oracle supplies a default value of 1024.

### Return Values

FOPEN returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL\_FILE package, and individual components should not be referenced or changed by the UTL\_FILE user.

**Table 223–10 FOPEN Function Return Values**

Return	Description
FILE_TYPE	Handle to open file

## Exceptions

INVALID\_MAXILINESIZE

INVALID\_MODE

INVALID\_OPERATION

INVALID\_PATH

## Usage Notes

The file location and file name parameters must be supplied to the FOPEN function as quoted strings so that the file location can be checked against the list of accessible directories as specified by the ALL\_DIRECTORIES view of accessible directory objects.



## FOPEN\_NCHAR Function

This function opens a file in national character set mode for input or output, with the maximum line size specified. You can have a maximum of 50 files open simultaneously. With this function, you can read or write a text file in Unicode instead of in the database character set.

Even though the contents of an NVARCHAR2 buffer may be AL16UTF16 or UTF8 (depending on the national character set of the database), the contents of the file are always read and written in UTF8. UTL\_FILE converts between UTF8 and AL16UTF16 as necessary.

See also [FOPEN Function](#) on page 223-19.

### Syntax

```
UTL_FILE.FOPEN_NCHAR (
    location      IN VARCHAR2,
    filename      IN VARCHAR2,
    open_mode     IN VARCHAR2,
    max_linesize  IN BINARY_INTEGER DEFAULT 1024)
RETURN FILE_TYPE;
```

### Parameters

**Table 223–11 FOPEN\_NCHAR Function Parameters**

Parameter	Description
location	Directory location of file
filename	File name (including extension)
open_mode	Open mode (r,w,a,rb,wb,ab)
max_linesize	Maximum number of characters for each line, including the newline character, for this file (minimum value 1, maximum value 32767)

### Return Values

FOPEN\_NCHAR returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL\_FILE package, and individual components should not be referenced or changed by the UTL\_FILE user.

**Table 223–12 FOPEN\_NCHAR Function Return Values**

Return	Description
FILE_TYPE	Handle to open file

### Exceptions

```
INVALID_MAXILINESIZE
INVALID_MODE
INVALID_OPERATION
INVALID_PATH
```

## FREMOVE Procedure

This procedure deletes a disk file, assuming that you have sufficient privileges.

### Syntax

```
UTL_FILE.FREMOVE (  
    location IN VARCHAR2,  
    filename IN VARCHAR2);
```

### Parameters

**Table 223–13** *FREMOVE Procedure Parameters*

Parameters	Description
location	Directory location of the file, a DIRECTORY_NAME from ALL_DIRECTORIES (case sensitive)
filename	Name of the file to be deleted

### Exceptions

ACCESS\_DENIED  
DELETE\_FAILED  
INVALID\_FILENAME  
INVALID\_OPERATION  
INVALID\_PATH

### Usage Notes

The FREMOVE procedure does not verify privileges before deleting a file. The O/S verifies file and directory permissions. An exception is returned on failure.

## FRENAME Procedure

This procedure renames an existing file to a new name, similar to the UNIX `mv` function.

### Syntax

```
UTL_FILE.FRENAME (
    src_location    IN  VARCHAR2,
    src_filename   IN  VARCHAR2,
    dest_location  IN  VARCHAR2,
    dest_filename  IN  VARCHAR2,
    overwrite      IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 223–14** *FRENAME Procedure Parameters*

Parameters	Description
<code>src_location</code>	Directory location of the source file, a <code>DIRECTORY_NAME</code> from the <code>ALL_DIRECTORIES</code> view (case sensitive)
<code>src_filename</code>	Source file to be renamed
<code>dest_location</code>	Destination directory of the destination file, a <code>DIRECTORY_NAME</code> from the <code>ALL_DIRECTORIES</code> view (case sensitive)
<code>dest_filename</code>	New name of the file
<code>overwrite</code>	Default is <code>FALSE</code> . Permission on both the source and destination directories must be granted. You can use the <code>overwrite</code> parameter to specify whether or not to overwrite a file if one exists in the destination directory. The default is <code>FALSE</code> for no overwrite.

### Exceptions

```
ACCESS_DENIED
INVALID_FILENAME
INVALID_PATH
RENAME_FAILED
```

## FSEEK Procedure

This procedure adjusts the file pointer forward or backward within the file by the number of bytes specified.

### Syntax

```
UTL_FILE.FSEEK (
    file           IN OUT  UTL_FILE.FILE_TYPE,
    absolute_offset IN     PL_INTEGER DEFAULT NULL,
    relative_offset IN     PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 223–15 FSEEK Procedure Parameters**

Parameters	Description
file	File handle
absolute_offset	Absolute location to which to seek; default = NULL
relative_offset	Number of bytes to seek forward or backward; positive = forward, negative integer = backward, zero = current position, default = NULL

### Exceptions

INVALID\_FILEHANDLE  
 INVALID\_OFFSET  
 INVALID\_OPERATION  
 READ\_ERROR

### Usage Notes

- Using FSEEK, you can read previous lines in the file without first closing and reopening the file. You must know the number of bytes by which you want to navigate.
- If *relative\_offset*, the procedure seeks forward. If *relative\_offset* > 0, or backward, if *relative\_offset* < 0, the procedure seeks through the file by the number of *relative\_offset* bytes specified.
- If the beginning of the file is reached before the number of bytes specified, then the file pointer is placed at the beginning of the file. If the end of the file is reached before the number of bytes specified, then an INVALID\_OFFSET error is raised.
- If *absolute\_offset*, the procedure seeks to an absolute location specified in bytes.
- If file is opened for byte mode operations, then the INVALID OPERATION exception is raised.

## GET\_LINE Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. Text is read up to, but not including, the line terminator, or up to the end of the file, or up to the end of the `len` parameter. It cannot exceed the `max_linesize` specified in `FOPEN`.

### Syntax

```
UTL_FILE.GET_LINE (
    file          IN  FILE_TYPE,
    buffer        OUT VARCHAR2,
    len           IN  PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 223–16** *GET\_LINE Procedure Parameters*

Parameters	Description
<code>file</code>	Active file handle returned by an <code>FOPEN</code> call. The file must be open for reading (mode <code>r</code> ); otherwise an <code>INVALID_OPERATION</code> exception is raised.
<code>buffer</code>	Data buffer to receive the line read from the file
<code>len</code>	The number of bytes read from the file. Default is <code>NULL</code> . If <code>NULL</code> , Oracle supplies the value of <code>max_linesize</code> .

### Exceptions

`INVALID_FILEHANDLE`

`INVALID_OPERATION`

`NO_DATA_FOUND`

`READ_ERROR`

### Usage Notes

If the line does not fit in the buffer, a `READ_ERROR` exception is raised. If no text was read due to end of file, the `NO_DATA_FOUND` exception is raised. If the file is opened for byte mode operations, the `INVALID_OPERATION` exception is raised.

Because the line terminator character is not read into the buffer, reading blank lines returns empty strings.

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`.

If unspecified, Oracle supplies a default value of 1024. See also "[GET\\_LINE\\_NCHAR Procedure](#)" on page 223-26.

## GET\_LINE\_NCHAR Procedure

This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. With this function, you can read a text file in Unicode instead of in the database character set.

The file must be opened in national character set mode, and must be encoded in the UTF8 character set. The expected buffer datatype is NVARCHAR2. If a variable of another datatype, such as NCHAR, NCLOB, or VARCHAR2 is specified, PL/SQL will perform standard implicit conversion from NVARCHAR2 after the text is read.

See also [GET\\_LINE Procedure](#) on page 223-25

### Syntax

```
UTL_FILE.GET_LINE_NCHAR (
    file          IN  FILE_TYPE,
    buffer        OUT NVARCHAR2,
    len           IN  PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 223–17** GET\_LINE\_NCHAR Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN_NCHAR call. The file must be open for reading (mode r). If the file is opened by FOPEN instead of FOPEN_NCHAR, a CHARSETMISMATCH exception is raised.
buffer	Data buffer to receive the line read from the file
len	The number of bytes read from the file. Default is NULL. If NULL, Oracle supplies the value of max_linesize.

### Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
NO_DATA_FOUND
READ_ERROR
```

## GET\_RAW Procedure

This procedure reads a RAW string value from a file and adjusts the file pointer ahead by the number of bytes read. UTL\_FILE.GET\_RAW ignores line terminators.

### Syntax

```
UTL_FILE.GET_RAW (
    file      IN          UTL_FILE.FILE_TYPE,
    buffer    OUT NOCOPY RAW,
    len       IN          PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 223–18** GET\_RAW Procedure Parameters

Parameters	Description
file	File handle
buffer	RAW data
len	The number of bytes read from the file. Default is NULL. If NULL, len is assumed to be the maximum length of RAW.

### Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
LENGTH_MISMATCH
NO_DATA_FOUND
READ_ERROR
```

### Usage Notes

The subprogram will raise `No_Data_Found` when it attempts to read past the end of the file. Your application should allow for this by catching the exception in its processing loop.

```
PROCEDURE Sys.p (n IN VARCHAR2) IS
    h      UTL_FILE.FILE_TYPE := UTL_FILE.FOPEN('D', n, 'r', 32767);
    Buf    RAW(32767);
    Amnt   CONSTANT PLS_INTEGER := 32767;
BEGIN
    LOOP
        BEGIN
            Utl_File.Get_Raw(h, Buf, Amnt);
            -- Do something with this chunk
        EXCEPTION WHEN No_Data_Found THEN EXIT; END;
    END LOOP;
    UTL_FILE.FCLOSE (h);
END;
```

## IS\_OPEN Function

This function tests a file handle to see if it identifies an open file. `IS_OPEN` reports only whether a file handle represents a file that has been opened, but not yet closed. It does not guarantee that there will be no operating system errors when you attempt to use the file handle.

### Syntax

```
UTL_FILE.IS_OPEN (  
    file IN FILE_TYPE)  
RETURN BOOLEAN;
```

### Parameters

**Table 223–19** *IS\_OPEN Function Parameters*

Parameter	Description
<code>file</code>	Active file handle returned by an <code>FOPEN</code> or <code>FOPEN_NCHAR</code> call

### Return Values

TRUE or FALSE

### Exceptions

`INVALID_FILEHANDLE`



## NEW\_LINE Procedure

This procedure writes one or more line terminators to the file identified by the input file handle. This procedure is separate from `PUT` because the line terminator is a platform-specific character or sequence of characters.

### Syntax

```
UTL_FILE.NEW_LINE (  
    file      IN FILE_TYPE,  
    lines     IN BINARY_INTEGER := 1);
```

### Parameters

**Table 223–20** *NEW\_LINE Procedure Parameters*

Parameters	Description
file	Active file handle returned by an <code>FOPEN</code> or <code>FOPEN_NCHAR</code> call
lines	Number of line terminators to be written to the file

### Exceptions

```
INVALID_FILEHANDLE  
INVALID_OPERATION  
WRITE_ERROR
```

## PUT Procedure

PUT writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. No line terminator is appended by PUT; use `NEW_LINE` to terminate the line or use `PUT_LINE` to write a complete line with a line terminator. See also "[PUT\\_NCHAR Procedure](#)" on page 223-33.

### Syntax

```
UTL_FILE.PUT (  
    file      IN FILE_TYPE,  
    buffer    IN VARCHAR2);
```

### Parameters

**Table 223–21** PUT Procedure Parameters

Parameters	Description
<code>file</code>	Active file handle returned by an <code>FOPEN_NCHAR</code> call. The file must be open for writing.
<code>buffer</code>	Buffer that contains the text to be written to the file. User must have opened the file using mode <code>w</code> or mode <code>a</code> ; otherwise, an <code>INVALID_OPERATION</code> exception is raised.

### Usage Notes

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential PUT calls cannot exceed 32767 without intermediate buffer flushes.

### Exceptions

```
INVALID_FILEHANDLE  
INVALID_OPERATION  
WRITE_ERROR
```

## PUT\_LINE Procedure

This procedure writes the text string stored in the `buffer` parameter to the open file identified by the file handle. The file must be open for write operations. `PUT_LINE` terminates the line with the platform-specific line terminator character or characters.

See also "[PUT\\_LINE\\_NCHAR Procedure](#)" on page 223-32.

### Syntax

```
UTL_FILE.PUT_LINE (
    file      IN FILE_TYPE,
    buffer    IN VARCHAR2,
    autoflush IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 223–22 PUT\_LINE Procedure Parameters**

Parameters	Description
<code>file</code>	Active file handle returned by an <code>FOPEN</code> call
<code>buffer</code>	Text buffer that contains the lines to be written to the file
<code>autoflush</code>	Flushes the buffer to disk after the <code>WRITE</code>

### Exceptions

`INVALID_FILEHANDLE`  
`INVALID_OPERATION`  
`WRITE_ERROR`

### Usage Notes

- The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.
- If file is opened for byte mode operations, then the `INVALID_OPERATION` exception is raised.

## PUT\_LINE\_NCHAR Procedure

This procedure writes the text string stored in the `buffer` parameter to the open file identified by the file handle. With this function, you can write a text file in Unicode instead of in the database character set. This procedure is equivalent to the [PUT\\_NCHAR Procedure](#), except that the line separator is appended to the written text. See also [PUT\\_LINE Procedure](#) on page 223-31.

### Syntax

```
UTL_FILE.PUT_LINE_NCHAR (
    file      IN FILE_TYPE,
    buffer    IN NVARCHAR2);
```

### Parameters

**Table 223–23 PUT\_LINE\_NCHAR Procedure Parameters**

Parameters	Description
<code>file</code>	Active file handle returned by an <code>FOPEN_NCHAR</code> call. The file must be open for writing.
<code>buffer</code>	Text buffer that contains the lines to be written to the file

### Exceptions

INVALID\_FILEHANDLE  
 INVALID\_OPERATION  
 WRITE\_ERROR

### Usage Notes

- The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.
- If file is opened for byte mode operations, then the `INVALID_OPERATION` exception is raised.

## PUT\_NCHAR Procedure

This procedure writes the text string stored in the `buffer` parameter to the open file identified by the file handle.

With this function, you can write a text file in Unicode instead of in the database character set. The file must be opened in the national character set mode. The text string will be written in the UTF8 character set. The expected buffer datatype is `NVARCHAR2`. If a variable of another datatype is specified, PL/SQL will perform implicit conversion to `NVARCHAR2` before writing the text.

See also [PUT Procedure](#) on page 223-30

### Syntax

```
UTL_FILE.PUT_NCHAR (
    file      IN FILE_TYPE,
    buffer    IN NVARCHAR2);
```

### Parameters

**Table 223–24 PUT\_NCHAR Procedure Parameters**

Parameters	Description
<code>file</code>	Active file handle returned by an <code>FOPEN_NCHAR</code> call. If the file is opened by <code>FOPEN</code> instead of <code>FOPEN_NCHAR</code> , a <code>CHARSETMISMATCH</code> exception is raised.
<code>buffer</code>	Buffer that contains the text to be written to the file. User must have opened the file using mode <code>w</code> or mode <code>a</code> ; otherwise, an <code>INVALID_OPERATION</code> exception is raised.

### Exceptions

`INVALID_FILEHANDLE`

`INVALID_OPERATION`

`WRITE_ERROR`

### Usage Notes

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.

## PUTF Procedure

This procedure is a formatted PUT procedure. It works like a limited `printf()`. See also [PUTF\\_NCHAR Procedure](#) on page 223-36.

### Syntax

```
UTL_FILE.PUTF (
    file      IN FILE_TYPE,
    format    IN VARCHAR2,
    [arg1     IN VARCHAR2  DEFAULT NULL,
    . . .
    arg5      IN VARCHAR2  DEFAULT NULL]);
```

### Parameters

**Table 223–25** *PUTF Procedure Parameters*

Parameters	Description
file	Active file handle returned by an FOPEN call
format	Format string that can contain text as well as the formatting characters <code>\n</code> and <code>%s</code>
arg1..arg5	From one to five operational argument strings. Argument strings are substituted, in order, for the <code>%s</code> formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each <code>%s</code> for which there is no argument.

### Usage Notes

- If file is opened for byte mode operations, then the `INVALID OPERATION` exception is raised.
- The format string can contain any text, but the character sequences `%s` and `\n` have special meaning.

Character Sequence	Meaning
<code>%s</code>	Substitute this sequence with the string value of the next argument in the argument list.
<code>\n</code>	Substitute with the appropriate platform-specific line terminator.

### Exceptions

`INVALID_FILEHANDLE`  
`INVALID_OPERATION`  
`WRITE_ERROR`

### Examples

The following example writes the lines:

```
Hello, world!
I come from Zork with greetings for all earthlings.
```

```
my_world varchar2(4) := 'Zork';  
...  
PUTF(my_handle, 'Hello, world!\nI come from %s with %s.\n',  
      my_world,  
      'greetings for all earthlings');
```

If there are more %s formatters in the format parameter than there are arguments, then an empty string is substituted for each %s for which there is no matching argument.

## PUTF\_NCHAR Procedure

This procedure is a formatted version of a [PUT\\_NCHAR Procedure](#). Using `PUTF_NCHAR`, you can write a text file in Unicode instead of in the database character set. It accepts a format string with formatting elements `\n` and `%s`, and up to five arguments to be substituted for consecutive instances of `%s` in the format string. The expected datatype of the format string and the arguments is `NVARCHAR2`.

If variables of another datatype are specified, PL/SQL will perform implicit conversion to `NVARCHAR2` before formatting the text. Formatted text is written in the UTF8 character set to the file identified by the file handle. The file must be opened in the national character set mode.

### Syntax

```
UTL_FILE.PUTF_NCHAR (
    file      IN FILE_TYPE,
    format    IN NVARCHAR2,
    [arg1     IN NVARCHAR2  DEFAULT NULL,
    . . .
    arg5      IN NVARCHAR2  DEFAULT NULL]);
```

### Parameters

**Table 223–26** *PUTF\_NCHAR Procedure Parameters*

Parameters	Description
file	Active file handle returned by an <code>FOPEN_NCHAR</code> call. The file must be open for reading (mode <code>r</code> ). If the file is opened by <code>FOPEN</code> instead of <code>FOPEN_NCHAR</code> , a <code>CHARSETMISMATCH</code> exception is raised.
format	Format string that can contain text as well as the formatting characters <code>\n</code> and <code>%s</code>
arg1..arg5	From one to five operational argument strings.  Argument strings are substituted, in order, for the <code>%s</code> formatters in the format string.  If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each <code>%s</code> for which there is no argument.

### Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

### Usage Notes

- The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.
- If file is opened for byte mode operations, then the `INVALID OPERATION` exception is raised.



## PUT\_RAW Procedure

This procedure accepts as input a RAW data value and writes the value to the output buffer.

### Syntax

```
UTL_FILE.PUT_RAW (
    file          IN    UTL_FILE.FILE_TYPE,
    buffer        IN    RAW,
    autoflush     IN    BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 223–27 PUT\_RAW Procedure Parameters**

Parameters	Description
file	File handle
buffer	The RAW data written to the buffer
autoflush	If TRUE, then performs a flush after writing the value to the output buffer; default is FALSE.

### Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

### Usage Notes

You can request an automatic flush of the buffer by setting the third argument to TRUE.

The maximum size of the `buffer` parameter is 32767 bytes unless you specify a smaller size in `FOPEN`. If unspecified, Oracle supplies a default value of 1024. The sum of all sequential `PUT` calls cannot exceed 32767 without intermediate buffer flushes.



The UTL\_HTTP package makes Hypertext Transfer Protocol (HTTP) callouts from SQL and PL/SQL. You can use it to access data on the Internet over HTTP.

When the package fetches data from a Web site using HTTPS, it requires Oracle Wallet Manager which can be created by either Oracle Wallet Manager or the orapki utility. Non-HTTPS fetches do not require an Oracle wallet.

**See Also:**

- [Chapter 238, "UTL\\_URL"](#)
- [Chapter 235, "UTL\\_SMTP"](#)
- *Oracle Database Advanced Security Administrator's Guide* for more information on Wallet Manager

This chapter contains the following topics:

- [Using UTL\\_HTTP](#)
  - Overview
  - Security Model
  - Constants
  - Datatypes
  - Operational Notes
  - Exceptions
  - Examples
- [Subprogram Groups](#)
  - Session Settings Subprograms
  - HTTP Requests Subprograms
  - HTTP Request Contexts Subprograms
  - HTTP Responses Subprograms
  - HTTP Cookies Subprograms
  - HTTP Persistent Connections Subprograms
  - Error Conditions Subprograms
- [Summary of UTL\\_HTTP Subprograms](#)

## Using UTL\_HTTP

This section contains topics which relate to using the UTL\_HTTP package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Datatypes](#)
- [Operational Notes](#)
- [Exceptions](#)
- [Examples](#)

## Overview

With the `UTL_HTTP` package, you can write PL/SQL programs that communicate with Web (HTTP) servers. And `UTL_HTTP` contains a function that can be used in SQL queries.

The package supports HTTP over the Secured Socket Layer protocol (SSL), also known as HTTPS. It also supports SSL client authentication by sending the client-certificate in a wallet to authenticate with the remote Web server.

Other Internet-related data-access protocols (such as the File Transfer Protocol (FTP) or the Gopher protocol) are also supported using an HTTP proxy server that supports those protocols.

## Security Model

This package is an invoker's rights package and the invoking user will need the `connect` privilege granted in the access control list assigned to the remote network host to which he wants to connect, as well as the `use-client-certificates` or the `use-passwords` privilege to authenticate himself with the remote Web server using the credentials stored in an Oracle wallet.

---

---

**Note:** For more information, see *Managing Fine-grained Access to External Network Services* in *Oracle Database Security Guide*

---

---

## Constants

The UTL\_HTTP package uses the constants shown in following tables.

- [UTL\\_HTTP Constants - HTTP Versions](#)
- [UTL\\_HTTP Constants - Default Ports](#)
- [UTL\\_HTTP Constants - HTTP 1.1 Status Codes](#)

**Table 224–1 UTL\_HTTP Constants - HTTP Versions**

Name	Type	Value	Description
HTTP_VERSION_1_0	VARCHAR2(10)	'HTTP /1.0'	Denotes HTTP version 1.0 that can be used in the function BEGIN_REQUEST.
HTTP_VERSION_1_1	VARCHAR2(10)	'HTTP /1.1'	Denotes HTTP version 1.1 that can be used in the function BEGIN_REQUEST.

**Table 224–2 UTL\_HTTP Constants - Default Ports**

Name	Type	Value	Description
DEFAULT_HTTP_PORT	PLS_INTEGER	80	The default TCP/IP port (80) at which a Web server or proxy server listens
DEFAULT_HTTPS_PORT	PLS_INTEGER	443	The default TCP/IP port (443) at which an HTTPS Web server listens

**Table 224–3 UTL\_HTTP Constants - HTTP 1.1 Status Codes**

Name	Type	Value	Description
HTTP_CONTINUE	PLS_INTEGER	100	The client should continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server.
HTTP_SWITCHING_PROTOCOLS	PLS_INTEGER	101	The server understands and is willing to comply with the client's request, through the Upgrade message header field, for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.
HTTP_OK	PLS_INTEGER	200	The request has succeeded. The information returned with the response is dependent on the method used in the request
HTTP_CREATED CONSTANT	PLS_INTEGER	201	The request has been fulfilled and resulted in a new resource being created.
HTTP_ACCEPTED	PLS_INTEGER	202	The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.

**Table 224-3 (Cont.) UTL\_HTTP Constants - HTTP 1.1 Status Codes**

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
HTTP_NON_AUTHORITATIVE_INFO	PLS_INTEGER	203	The returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy.
HTTP_NO_CONTENT	PLS_INTEGER	204	The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation.
HTTP_RESET_CONTENT	PLS_INTEGER	205	The server has fulfilled the request and the user agent should reset the document view which caused the request to be sent. The response must not include an entity.
HTTP_PARTIAL_CONTENT	PLS_INTEGER	206	The server has fulfilled the partial GET request for the resource.
HTTP_MULTIPLE_CHOICES	PLS_INTEGER	300	The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent-driven negotiation information is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.
HTTP_MOVED_PERMANENTLY	PLS_INTEGER	301	The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs.
HTTP_FOUND_CONSTANT	PLS_INTEGER	302	The requested resource resides temporarily under a different URI.
HTTP_SEE_OTHER	PLS_INTEGER	303	The response to the request can be found under a different URI and should be retrieved using a GET method on that resource.
HTTP_NOT_MODIFIED	PLS_INTEGER	304	If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server responds with this status code.
HTTP_USE_PROXY	PLS_INTEGER	305	The requested resource must be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy.
HTTP_TEMPORARY_REDIRECT	PLS_INTEGER	307	The requested resource resides temporarily under a different URI.
HTTP_BAD_REQUEST	PLS_INTEGER	400	The request could not be understood by the server due to malformed syntax.
HTTP_UNAUTHORIZED	PLS_INTEGER	401	The request requires user authentication. The client may repeat the request with a suitable Authorization header field. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.
HTTP_PAYMENT_REQUIRED	PLS_INTEGER	402	This code is reserved for future use.



**Table 224–3 (Cont.) UTL\_HTTP Constants - HTTP 1.1 Status Codes**

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
HTTP_FORBIDDEN	PLS_INTEGER	403	The server understood the request, but is refusing to fulfill it.
HTTP_NOT_FOUND	PLS_INTEGER	404	The server has not found anything matching the Request-URI.
HTTP_NOT_ACCEPTABLE	PLS_INTEGER	406	The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.
HTTP_PROXY_AUTH_REQUIRED	PLS_INTEGER	407	This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.
HTTP_REQUEST_TIMEOUT	PLS_INTEGER	408	The client did not produce a request within the time that the server was prepared to wait.
HTTP_CONFLICT	PLS_INTEGER	409	The request could not be completed due to a conflict with the current state of the resource.
HTTP_GONE	PLS_INTEGER	410	The requested resource is no longer available at the server and no forwarding address is known.
HTTP_LENGTH_REQUIRED	PLS_INTEGER	411	The server refuses to accept the request without a defined Content-Length.
HTTP_PRECONDITION_FAILED	PLS_INTEGER	412	The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.
HTTP_REQUEST_ENTITY_TOO_LARGE_CONSTANT	PLS_INTEGER	413	The server is refusing to process a request because the request entity is larger than the server is willing or able to process.
HTTP_REQUEST_URI_TOO_LARGE	PLS_INTEGER	414	The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.
HTTP_UNSUPPORTED_MEDIA_TYPE	PLS_INTEGER	415	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
HTTP_REQ_RANGE_NOT_SATISFIABLE	PLS_INTEGER	416	A server returns a response with this status code if a request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field.
HTTP_EXPECTATION_FAILED	PLS_INTEGER	417	The expectation given in an Expect request-header field could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

**Table 224–3 (Cont.) UTL\_HTTP Constants - HTTP 1.1 Status Codes**

<b>Name</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
HTTP_NOT_IMPLEMENTED	PLS_INTEGER	501	The server does not support the functionality required to fulfill the request.
HTTP_BAD_GATEWAY	PLS_INTEGER	502	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request
HTTP_SERVICE_UNAVAILABLE	PLS_INTEGER	503	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
HTTP_GATEWAY_TIME_OUT	PLS_INTEGER	504	The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (for example, HTTP, FTP, LDAP) or some other auxiliary server (for example, DNS) it needed to access in attempting to complete the request.
HTTP_VERSION_NOT_SUPPORTED	PLS_INTEGER	505	The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

## Datatypes

- [REQ Type](#)
- [RESP Type](#)
- [COOKIE and COOKIE\\_TABLE Types](#)
- [CONNECTION Type](#)
- [REQUEST\\_CONTEXT\\_KEY Type](#)

### REQ Type

Use this PL/SQL record type to represent an HTTP request.

#### Syntax

```
TYPE req IS RECORD (
  url          VARCHAR2(32767),
  method      VARCHAR2(64),
  http_version VARCHAR2(64));
```

#### Parameters

**Table 224–4 REQ Type Parameters**

Parameter	Description
url	The URL of the HTTP request. It is set after the request is created by <code>BEGIN_REQUEST</code> .
method	The method to be performed on the resource identified by the URL. It is set after the request is created by <code>BEGIN_REQUEST</code> .
http_version	The HTTP protocol version used to send the request. It is set after the request is created by <code>BEGIN_REQUEST</code> .

#### Usage Notes

The information returned in `REQ` from the interface `begin_request` is for read-only. Changing the field values in the record has no effect on the request.

There are other fields in `REQ` record type whose names begin with the prefix `private_`. The fields are private and are intended for use by implementation of the `UTL_HTTP` package. You should not modify the fields.

### REQUEST\_CONTEXT\_KEY Type

This type is used to represent the key to a request context. A request context is a context that holds a private wallet and cookie table to make a HTTP request. This private wallet and cookie table, unlike the session-wide ones maintained in the package, will not be shared with other HTTP requests within the database session.

#### Syntax

```
SUBTYPE request_context_key IS PLS_INTEGER;
```

#### Usage Notes

To provide enhanced security, `UTL_HTTP` allows PL/SQL programs to create request contexts. A request context is a private context that holds a wallet and a cookie table that will not be shared with other programs in the same database session when

making HTTP requests and receiving HTTP responses. PL/SQL programs should use request contexts when they need to use wallets or cookies that contain sensitive information such as authentication credentials.

## RESP Type

This PL/SQL record type is used to represent an HTTP response.

### Syntax

```
TYPE resp IS RECORD (
    status_code    PLS_INTEGER,
    reason_phrase  VARCHAR2(256),
    http_version   VARCHAR2(64));
```

### Parameters

**Table 224–5** *RESP Type Parameters*

Parameter	Description
status_code	The status code returned by the Web server. It is a 3-digit integer that indicates the results of the HTTP request as handled by the Web server. It is set after the response is processed by GET_RESPONSE.
reason_phrase	The short textual message returned by the Web server that describe the status code. It gives a brief description of the results of the HTTP request as handled by the Web server. It is set after the response is processed by GET_RESPONSE.
http_version	The HTTP protocol version used in the HTTP response. It is set after the response is processed by GET_RESPONSE.

### Usage Notes

The information returned in RESP from the interface GET\_RESPONSE is read-only. There are other fields in the RESP record type whose names begin with the prefix `private_`. The fields are private and are intended for use by implementation of the UTL\_HTTP package. You should not modify the fields.

## COOKIE and COOKIE\_TABLE Types

The COOKIE type is the PL/SQL record type that represents an HTTP cookie. The COOKIE\_TABLE type is a PL/SQL index-by-table type that represents a collection of HTTP cookies.

### Syntax

```
TYPE cookie IS RECORD (
    name  VARCHAR2(256),
    value VARCHAR2(1024),
    domain VARCHAR2(256),
    expire TIMESTAMP WITH TIME ZONE,
    path  VARCHAR2(1024),
    secure BOOLEAN,
    version PLS_INTEGER,
    comment VARCHAR2(1024));

TYPE cookie_table IS TABLE OF cookie INDEX BY binary_integer;
```

## Fields of COOKIE Record Type

Table 224–6 shows the fields for the `COOKIE` and `COOKIE_TABLE` record types.

**Table 224–6 Fields of `COOKIE` and `COOKIE_TABLE` Type**

Field	Description
<code>name</code>	The name of the HTTP cookie
<code>value</code>	The value of the cookie
<code>domain</code>	The domain for which the cookie is valid
<code>expire</code>	The time by which the cookie will expire
<code>path</code>	The subset of URLs to which the cookie applies
<code>secure</code>	Should the cookie be returned to the Web server using secured means only.
<code>version</code>	The version of the HTTP cookie specification the cookie conforms. This field is <code>NULL</code> for Netscape cookies.
<code>comment</code>	The comment that describes the intended use of the cookie. This field is <code>NULL</code> for Netscape cookies.

## Usage Notes

PL/SQL programs do not usually examine or change the cookie information stored in the `UTL_HTTP` package. The cookies are maintained by the package transparently. They are maintained inside the `UTL_HTTP` package, and they last for the duration of the database session only. PL/SQL applications that require cookies to be maintained beyond the lifetime of a database session can read the cookies using `GET_COOKIES`, store them persistently in a database table, and re-store the cookies back in the package using `ADD_COOKIES` in the next database session. All the fields in the `cookie` record, except for the comment field, must be stored. Do not alter the cookie information, which can result in an application error in the Web server or compromise the security of the PL/SQL and the Web server applications. See "[Retrieving and Restoring Cookies](#)" on page 224-22.

## CONNECTION Type

Use the PL/SQL record type to represent the remote hosts and TCP/IP ports of a network connection that is kept persistent after an HTTP request is completed, according to the HTTP 1.1 protocol specification. The persistent network connection may be reused by a subsequent HTTP request to the same host and port. The subsequent HTTP request may be completed faster because the network connection latency is avoided. `connection_table` is a PL/SQL table of `connection`.

For a direct HTTP persistent connection to a Web server, the `host` and `port` fields contain the host name and TCP/IP port number of the Web server. The `proxy_host` and `proxy_port` fields are not set. For an HTTP persistent connection that was previously used to connect to a Web server using a proxy, the `proxy_host` and `proxy_port` fields contain the host name and TCP/IP port number of the proxy server. The `host` and `port` fields are not set, which indicates that the persistent connection, while connected to a proxy server, is not bound to any particular target Web server. An HTTP persistent connection to a proxy server can be used to access any target Web server that is using a proxy.

The `SSL` field indicates if Secured Socket Layer (SSL) is being used in an HTTP persistent connection. An HTTPS request is an HTTP request made over SSL. For an HTTPS (SSL) persistent connection connected using a proxy, the `host` and `port` fields contain the host name and TCP/IP port number of the target HTTPS Web server and

the fields will always be set. An HTTPS persistent connection to an HTTPS Web server using a proxy server can only be reused to make another request to the same target Web server.

**Syntax**

```
TYPE connection IS RECORD (  
    host VARCHAR2(256),  
    port PLS_INTEGER,  
    proxy_host VARCHAR2(256),  
    proxy_port PLS_INTEGER,  
    ssl BOOLEAN);
```

```
TYPE connection_table IS TABLE OF connection INDEX BY BINARY_INTEGER;
```

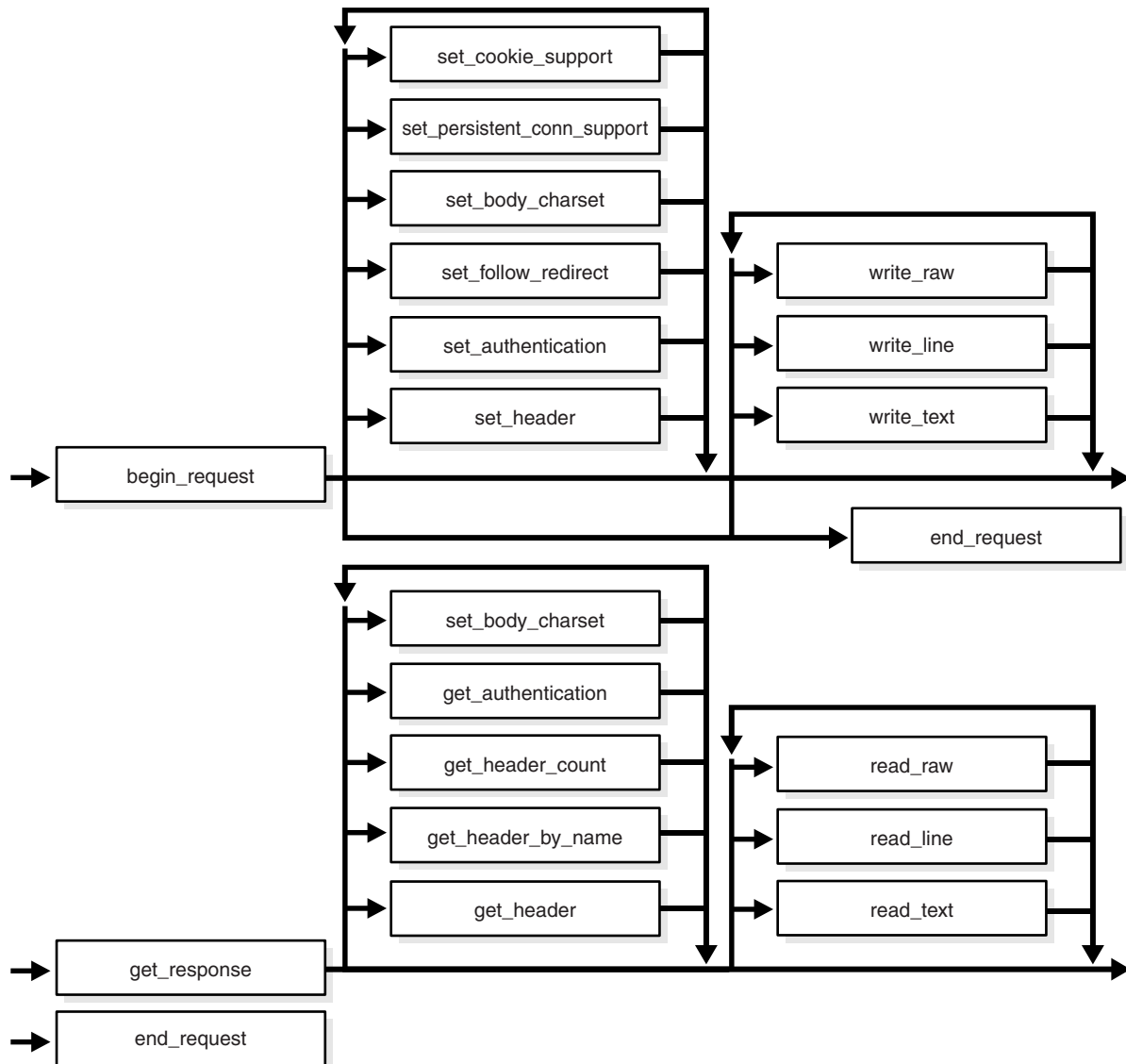
## Operational Notes

- [Operational Flow](#)
- [Simple HTTP Fetches](#)
- [HTTP Requests](#)
- [HTTP Responses](#)
- [HTTP Persistent Connections](#)
- [Error Conditions](#)
- [Session Settings](#)
- [Request Context](#)
- [External Password Store](#)

## Operational Flow

The UTL\_HTTP package provides access to the HTTP protocol. The interfaces must be called in the order shown in [Figure 224-1](#), or an exception will be raised.

Figure 224-1 Flow of the Core UTL\_HTTP Package



This illustration describes execution flow in UTL\_HTTP that has two phases: (1) beginning with "begin\_request" and ending with "end\_request", (2) beginning with "get\_response" and ending with "end\_response". Key to the symbols used:

- ... -> denotes "continue to next step", for example, "begin\_request" -> "get\_response"
- ... material enclosed in double quotation marks denotes an action, for example "PARSE"
- ... material enclosed in single quotation marks denotes a choice point, for example, 'perform pre-response set operations': Yes/No
- ... material unenclosed denotes a choice of direction, for example, Yes

-----  
PHASE #1.



- "begin\_request" -> 'perform pre-response set operations': Yes/No
  - 'perform set operations': Yes -> "set\_cookie\_support" (optional with option to repeat multiple times) -> "set\_persistent\_conn\_support" (optional with option to repeat multiple times)-> "set\_body\_charset" (optional with option to repeat multiple times) -> "set\_follow\_redirect" (optional with option to repeat multiple times) ->"set\_authentication" (optional with option to repeat multiple times) -> "set\_header" (optional with option to repeat multiple times) -> 'perform pre-response write operations':Yes/No
  - 'perform set operations': No -> 'perform write operations':Yes/No
- 'perform pre-response write operations':Yes -> "write\_raw" (optional with option to repeat multiple times) -> "write\_line" (optional with option to repeat multiple times) -> "write\_text" (optional with option to repeat multiple times) -> "end\_request"
- 'perform pre-response write operations': No -> "end\_request"

#### PHASE #2.

- "get\_response" -> 'perform get operations': Yes/No
  - 'perform get operations': Yes -> "set\_body\_charset" (optional with option to repeat multiple times) -> "get\_authentication" (optional with option to repeat multiple times) -> "get\_header\_count" (optional with option to repeat multiple times) ->"get\_header\_by\_name" (optional with option to repeat multiple times) -> 'perform read operations':Yes/No
  - 'perform get operations': No -> 'perform read operations':Yes/No
- 'perform read operations':Yes -> "read\_raw" (optional with option to repeat multiple times) ->"read\_line" (optional with option to repeat multiple times) -> "read\_text" (optional with option to repeat multiple times) -> end\_response
- 'perform read operations':No -> end\_response

\*\*\*\*\*

The following can be called at any time:

- Non-protocol interfaces that manipulate cookies
  - GET\_COOKIE\_COUNT
  - GET\_COOKIES
  - ADD\_COOKIES
  - CLEAR\_COOKIES
- Persistent connections
  - GET\_PERSISTENT\_CONN\_COUNT
  - GET\_PERSISTENT\_CONNS
  - CLOSE\_PERSISTENT\_CONN
  - CLOSE\_PERSISTENT\_CONNS
- Interfaces that manipulate attributes and configurations of the UTL\_HTTP package in the current session
  - SET\_PROXY
  - GET\_PROXY

- SET\_COOKIE\_SUPPORT
- GET\_COOKIE\_SUPPORT
- SET\_FOLLOW\_REDIRECT
- GET\_FOLLOW\_REDIRECT
- SET\_BODY\_CHARSET
- GET\_BODY\_CHARSET
- SET\_PERSISTENT\_CONN\_SUPPORT
- GET\_PERSISTENT\_CONN\_SUPPORT
- SET\_DETAILED\_EXCP\_SUPPORT
- GET\_DETAILED\_EXCP\_SUPPORT
- SET\_WALLET
- SET\_TRANSFER\_TIMEOUT
- GET\_TRANSFER\_TIMEOUT
- Interfaces that retrieve the last detailed exception code and message UTL\_HTTP package in the current session
  - GET\_DETAILED\_SQLCODE
  - GET\_DETAILED\_SQLERRM

---

---

**NOTE:** Some of the request and response interfaces bear the same name as the interface that manipulates the attributes and configurations of the package in the current session. They are overloaded versions of the interface that manipulate a request or a response.

---

---

## Simple HTTP Fetches

REQUEST and REQUEST\_PIECES take a string uniform resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

You should not expect REQUEST or REQUEST\_PIECES to succeed in contacting a URL unless you can contact that URL by using a browser on the same machine (and with the same privileges, environment variables, and so on.)

If REQUEST or REQUEST\_PIECES fails (for example, if it raises an exception, or if it returns an HTML-formatted error message, but you believe that the URL argument is correct), then try contacting that same URL with a browser to verify network availability from your machine. You may have a proxy server set in your browser that needs to be set with each REQUEST or REQUEST\_PIECES call using the optional proxy parameter.

---

---

**Note:** UTL\_HTTP can also use environment variables to specify its proxy behavior. For example, on UNIX, setting the environment variable `http_proxy` to a URL uses that service as the proxy server for HTTP requests. Setting the environment variable `no_proxy` to a domain name does not use the HTTP proxy server for URLs in that domain. When the UTL\_HTTP package is executed in the Oracle database server, the environment variables are the ones that are set when the database instance is started.

---

---

**See Also:** [Simple HTTP Fetches in a Single Call Subprograms](#) on page 224-26

## HTTP Requests

The HTTP Requests group of subprograms begin an HTTP request, manipulate attributes, and send the request information to the Web server. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout of the current session. The settings can be changed by calling the request interface.

**See Also:** [HTTP Requests Subprograms](#) on page 224-28

## HTTP Responses

The HTTP Responses group of subprograms manipulate an HTTP response obtained from GET\_RESPONSE and receive response information from the Web server. When a response is created for a request, it inherits settings of the HTTP cookie support, follow-redirect, body character set, persistent-connection support, and transfer timeout from the request. Only the body character set can be changed by calling the response interface.

**See Also:** [HTTP Responses Subprograms](#) on page 224-30

## HTTP Cookies

The UTL\_HTTP package provides subprograms to manipulate HTTP cookies.

**See Also:** [HTTP Cookies Subprograms](#) on page 224-31

## HTTP Persistent Connections

The UTL\_HTTP package provides subprograms to manipulate persistent connections.

**See Also:** [HTTP Persistent Connections Subprograms](#) on page 224-32

## Error Conditions

The UTL\_HTTP package provides subprograms to retrieve error information.

**See Also:** [Error Conditions Subprograms](#) on page 224-33

## Session Settings

Session settings manipulate the configuration and default behavior of UTL\_HTTP when HTTP requests are executed within a database user session. When a request is created, it inherits the default settings of the HTTP cookie support, follow-redirect,

body character set, persistent-connection support, and transfer timeout of the current session. Those settings can be changed later by calling the request interface. When a response is created for a request, it inherits those settings from the request. Only the body character set can be changed later by calling the response interface.

**See Also:** [Session Settings Subprograms](#) on page 224-27

## Request Context

The `UTL_HTTP` package maintains a common wallet and cookie table within the database session that all HTTP requests and responses share. This makes it easy for users to share the wallet or to maintain application state in the cookies within the session. However, if an application stores private information in the wallet or in the cookies that it does not want to share with other applications in the same database session, it may define a request context to hold its own wallet and cookie table and use this request context to make HTTP requests.

**See Also:** [HTTP Requests Subprograms](#) on page 224-28

## External Password Store

The `UTL_HTTP` package allows HTTP password credentials to be stored in an Oracle wallet's external password store. The external password store provides an easy but secure storage for passwords and frees the application developers from the need to maintain their own storage.

**See Also:** [SET\\_AUTHENTICATION\\_FROM\\_WALLET Procedure](#) on page 224-79

## Exceptions

Table 224–7 lists the exceptions that the UTL\_HTTP package interface can raise. By default, UTL\_HTTP raises the exception `request_failed` when a request fails to execute. If the package is set to raise a detailed exception by `set_detailed_excpt_support`, the rest of the exceptions will be raised directly (except for the exception `end_of_body`, which will be raised by `READ_TEXT`, `READ_LINE`, and `READ_RAW` regardless of the setting).

**Table 224–7 UTL\_HTTP Exceptions**

Exception	Error Code	Reason	Where Raised
BAD_ARGUMENT	29261	The argument passed to the interface is bad	Any HTTP request or response interface when <code>detailed_exception</code> is enabled
BAD_URL	29262	The requested URL is badly formed	BEGIN_REQUEST, when <code>detailed_exception</code> is enabled
END_OF_BODY	29266	The end of HTTP response body is reached	READ_RAW, READ_TEXT, and READ_LINE, when <code>detailed_exception</code> is enabled
HEADER_NOT_FOUND	29265	The header is not found	GET_HEADER, GET_HEADER_BY_NAME, when <code>detailed_exception</code> is enabled
HTTP_CLIENT_ERROR	29268	From GET_RESPONSE, the response status code indicates that a client error has occurred (status code in 4xx range). Or from begin_request, the HTTP proxy returns a status code in the 4xx range when making an HTTPS request through the proxy.	GET_RESPONSE, BEGIN_REQUEST when <code>detailed_exception</code> is enabled
HTTP_SERVER_ERROR	29269	From GET_RESPONSE, the response status code indicates that a client error has occurred (status code in 5xx range). Or from begin_request, the HTTP proxy returns a status code in the 5xx range when making an HTTPS request through the proxy.	GET_RESPONSE, BEGIN_REQUEST when <code>detailed_exception</code> is enabled
NETWORK_ACCESS_DENIED	24247	Access to the remote network host or credentials in an Oracle wallet is denied	BEGIN_REQUEST and SET_AUTHENTICATION_FROM_WALLET when <code>detailed_exception</code> is enabled
ILLEGAL_CALL	29267	The call to UTL_HTTP is illegal at the current state of the HTTP request	SET_HEADER, SET_AUTHENTICATION, and SET_PERSISTENT_CONN_SUPPORT, when <code>detailed_exception</code> is enabled
PARTIAL_MULTIBYTE_EXCEPTION	29275	No complete character is read and a partial multibyte character is found at the end of the response body	READ_TEXT and READ_LINE, when <code>detailed_exception</code> is enabled
PROTOCOL_ERROR	29263	An HTTP protocol error occurs when communicating with the Web server	SET_HEADER, GET_RESPONSE, READ_RAW, READ_TEXT, and READ_LINE, when <code>detailed_exception</code> is enabled

**Table 224–7 (Cont.) UTL\_HTTP Exceptions**

Exception	Error Code	Reason	Where Raised
REQUEST_FAILED	29273	The request fails to executes	Any HTTP request or response interface when <code>detailed_exception</code> is disabled
TOO_MANY_REQUESTS	29270	Too many requests or responses are open	BEGIN_REQUEST, when <code>detailed_exception</code> is enabled
TRANSFER_TIMEOUT	29276	No data is read and a read timeout occurred	READ_TEXT and READ_LINE, when <code>detailed_exception</code> is enabled
UNKNOWN_SCHEME	29264	The scheme of the requested URL is unknown	BEGIN_REQUEST and GET_RESPONSE, when <code>detailed_exception</code> is enabled

---

**NOTE:** The `partial_multibyte_char` and `transfer_timeout` exceptions are duplicates of the same exceptions defined in `UTL_TCP`. They are defined in this package so that the use of this package does not require the knowledge of the `UTL_TCP`. As those exceptions are duplicates, an exception handle that catches the `partial_multibyte_char` and `transfer_timeout` exceptions in this package also catch the exceptions in the `UTL_TCP`.

---

For `REQUEST` and `REQUEST_PIECES`, the `request_failed` exception is raised when any exception occurs and `detailed_exception` is disabled.

## Examples

The following examples demonstrate how to use UTL\_HTTP.

- [General Usage](#)
- [Retrieving HTTP Response Headers](#)
- [Handling HTTP Authentication](#)
- [Retrieving and Restoring Cookies](#)
- [Making HTTP Request with Private Wallet and Cookie Table](#)

### General Usage

```
SET SERVEROUTPUT ON SIZE 40000

DECLARE
    req    UTL_HTTP.REQ;
    resp   UTL_HTTP.RESP;
    value  VARCHAR2(1024);
BEGIN
    UTL_HTTP.SET_PROXY('proxy.my-company.com', 'corp.my-company.com');
    req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
    UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
    resp := UTL_HTTP.GET_RESPONSE(req);
    LOOP
        UTL_HTTP.READ_LINE(resp, value, TRUE);
        DBMS_OUTPUT.PUT_LINE(value);
    END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
EXCEPTION
    WHEN UTL_HTTP.END_OF_BODY THEN
        UTL_HTTP.END_RESPONSE(resp);
END;
```

### Retrieving HTTP Response Headers

```
SET SERVEROUTPUT ON SIZE 40000

DECLARE
    req    UTL_HTTP.REQ;
    resp   UTL_HTTP.RESP;
    name   VARCHAR2(256);
    value  VARCHAR2(1024);
BEGIN
    UTL_HTTP.SET_PROXY('proxy.my-company.com', 'corp.my-company.com');
    req := UTL_HTTP.BEGIN_REQUEST('http://www-hr.corp.my-company.com');
    UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
    resp := UTL_HTTP.GET_RESPONSE(req);
    DBMS_OUTPUT.PUT_LINE('HTTP response status code: ' || resp.status_code);
    DBMS_OUTPUT.PUT_LINE('HTTP response reason phrase: ' || resp.reason_phrase);
    FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
        UTL_HTTP.GET_HEADER(resp, i, name, value);
        DBMS_OUTPUT.PUT_LINE(name || ': ' || value);
    END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
END;
```

## Handling HTTP Authentication

```

SET serveroutput ON SIZE 40000

CREATE OR REPLACE PROCEDURE get_page (url          IN VARCHAR2,
                                     username IN VARCHAR2 DEFAULT NULL,
                                     password IN VARCHAR2 DEFAULT NULL,
                                     realm     IN VARCHAR2 DEFAULT NULL) AS

    req          UTL_HTTP.REQ;
    resp         UTL_HTTP.RESP;
    my_scheme    VARCHAR2(256);
    my_realm     VARCHAR2(256);
    my_proxy     BOOLEAN;
BEGIN
    -- Turn off checking of status code. We will check it by ourselves.
    UTL_HTTP.HTTP_RESPONSE_ERROR_CHECK(FALSE);
    req := UTL_HTTP.BEGIN_REQUEST(url);
    IF (username IS NOT NULL) THEN
        UTL_HTTP.SET_AUTHENTICATION(req, username, password); -- Use HTTP Basic
Authen. Scheme
    END IF;
    resp := UTL_HTTP.GET_RESPONSE(req);
    IF (resp.status_code = UTL_HTTP.HTTP_UNAUTHORIZED) THEN
        UTL_HTTP.GET_AUTHENTICATION(resp, my_scheme, my_realm, my_proxy);
        IF (my_proxy) THEN
            DBMS_OUTPUT.PUT_LINE('Web proxy server is protected. ');
            DBMS_OUTPUT.PUT('Please supplied the required ' || my_scheme || '
authentication username/password for realm ' || my_realm || ' for the proxy
server. ');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Web page ' || url || ' is protected. ');
            DBMS_OUTPUT.PUT('Please supplied the required ' || my_scheme || '
authentication username/password for realm ' || my_realm || ' for the Web page. ');
        END IF;
        UTL_HTTP.END_RESPONSE(resp);
        RETURN;
    END IF;
    FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
        UTL_HTTP.GET_HEADER(resp, i, name, value);
        DBMS_OUTPUT.PUT_LINE(name || ': ' || value);
    END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
END;

```

## Retrieving and Restoring Cookies

```

CREATE TABLE my_cookies (
    session_id INTEGER,
    name       VARCHAR2(256),
    value      VARCHAR2(1024),
    domain     VARCHAR2(256),
    expire     DATE,
    path       VARCHAR2(1024),
    secure     VARCHAR2(1),
    version    INTEGER);

CREATE SEQUENCE session_id;
SET SERVEROUTPUT ON SIZE 40000

REM Retrieve cookies from UTL_HTTP
CREATE OR REPLACE FUNCTION save_cookies RETURN PLS_INTEGER AS

```



```

cookies          UTL_HTTP.COOKIE_TABLE;
my_session_id    PLS_INTEGER;
secure           VARCHAR2(1);
BEGIN
  /* assume that some cookies have been set in previous HTTP requests. */
  UTL_HTTP.GET_COOKIES(cookies);
  SELECT session_id.nextval INTO my_session_id FROM DUAL;
  FOR i in 1..cookies.count LOOP
    IF (cookies(i).secure) THEN
      secure := 'Y';
    ELSE
      secure := 'N';
    END IF;
    INSERT INTO my_cookies
    VALUES (my_session_id, cookies(i).name, cookies(i).value,
            cookies(i).domain,
            cookies(i).expire, cookies(i).path, secure, cookies(i).version);
  END LOOP;
  RETURN my_session_id;
END;
/

REM Retrieve cookies from UTL_HTTP
CREATE OR REPLACE PROCEDURE restore_cookies (this_session_id IN PLS_INTEGER)
AS
  cookies          UTL_HTTP.COOKIE_TABLE;
  cookie           UTL_HTTP.COOKIE;
  i                PLS_INTEGER := 0;
  CURSOR c (c_session_id PLS_INTEGER) IS
    SELECT * FROM my_cookies WHERE session_id = c_session_id;
BEGIN
  FOR r IN c(this_session_id) LOOP
    i := i + 1;
    cookie.name     := r.name;
    cookie.value    := r.value;
    cookie.domain   := r.domain;
    cookie.expire   := r.expire;
    cookie.path     := r.path;
    IF (r.secure = 'Y') THEN
      cookie.secure := TRUE;
    ELSE
      cookie.secure := FALSE;
    END IF;
    cookie.version := r.version;
    cookies(i) := cookie;
  END LOOP;
  UTL_HTTP.CLEAR_COOKIES;
  UTL_HTTP.ADD_COOKIES(cookies);
END;
/

```

## Making HTTP Request with Private Wallet and Cookie Table

```

SET SERVEROUTPUT ON SIZE 40000

CREATE OR REPLACE PROCEDURE DISPLAY_PAGE(url IN VARCHAR2) AS
  request_context UTL_HTTP.REQUEST_CONTEXT_KEY;
  req             UTL_HTTP.REQ;
  resp            UTL_HTTP.RESP;
  data            VARCHAR2(1024);

```

```
BEGIN

-- Create a request context with its wallet and cookie table
request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(
    wallet_path      => 'file:/oracle/wallets/test/wallet',
    wallet_password  => '*****',
    enable_cookies   => TRUE,
    max_cookies      => 300,
    max_cookies_per_site => 20);

-- Make a HTTP request using the private wallet and cookie
-- table in the request context
req := UTL_HTTP.BEGIN_REQUEST(
    url              => url,
    request_context => request_context);
resp := UTL_HTTP.GET_RESPONSE(req);

BEGIN
    LOOP
        UTL_HTTP.READ_TEXT(resp, data);
        DBMS_OUTPUT.PUT(data);
    END LOOP;
EXCEPTION
    WHEN UTL_HTTP.END_OF_BODY THEN
        UTL_HTTP.END_RESPONSE(resp);
END;

-- Destroy the request context
UTL_HTTP.DESTROY_REQUEST_CONTEXT(request_context);

END;

BEGIN
    DISPLAY_PAGE('https://www.example.com/');
END;
/
```

## Subprogram Groups

UTL\_HTTP subprograms are grouped by function:

- [Simple HTTP Fetches in a Single Call Subprograms](#)
- [Session Settings Subprograms](#)
- [HTTP Requests Subprograms](#)
- [HTTP Request Contexts Subprograms](#)
- [HTTP Responses Subprograms](#)
- [HTTP Cookies Subprograms](#)
- [HTTP Persistent Connections Subprograms](#)
- [Error Conditions Subprograms](#)

## Simple HTTP Fetches in a Single Call Subprograms

REQUEST and REQUEST\_PIECES take a string uniform resource locator (URL), contact that site, and return the data (typically HTML) obtained from that site.

**Table 224–8** UTL\_HTTP Subprograms—Simple HTTP Fetches in a Single Call

Subprogram	Description
<a href="#">REQUEST Function</a> on page 224-73	Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.
<a href="#">REQUEST_PIECES Function</a> on page 224-75	Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL

## Session Settings Subprograms

**Table 224–9 UTL\_HTTP Subprograms—Session Settings**

Subprogram	Description
<a href="#">GET_BODY_CHARSET Procedure</a> on page 224-51	Retrieves the default character set of the body of all future HTTP requests
<a href="#">GET_COOKIE_SUPPORT Procedure</a> on page 224-53	Retrieves the current cookie support settings
<a href="#">GET_DETAILED_EXCP_SUPPORT Procedure</a> on page 224-55	Checks if the UTL_HTTP package will raise a detailed exception or not
<a href="#">GET_FOLLOW_REDIRECT Procedure</a> on page 224-58	Retrieves the follow-redirect setting in the current session
<a href="#">GET_PERSISTENT_CONN_SUPPORT Procedure</a> on page 224-63	Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session
<a href="#">GET_PROXY Procedure</a> on page 224-65	Retrieves the current proxy settings
<a href="#">GET_RESPONSE_ERROR_CHECK Procedure</a> on page 224-67	Checks if the response error check is set or not
<a href="#">GET_TRANSFER_TIMEOUT Procedure</a> on page 224-68	Retrieves the current network transfer timeout value
<a href="#">SET_TRANSFER_TIMEOUT Procedure</a> on page 224-92	Sets the default character set of the body of all future HTTP requests when the media type is <code>text</code> and the character set is not specified in the <code>Content-Type</code> header
<a href="#">SET_COOKIE_SUPPORT Procedures</a> on page 224-83	Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session
<a href="#">SET_DETAILED_EXCP_SUPPORT Procedure</a> on page 224-85	Sets the UTL_HTTP package to raise a detailed exception
<a href="#">SET_FOLLOW_REDIRECT Procedures</a> on page 224-86	Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP responses to future requests in the <code>GET_RESPONSE</code> function
<a href="#">SET_PERSISTENT_CONN_SUPPORT Procedure</a> on page 224-88	Sets whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session
<a href="#">SET_PROXY Procedure</a> on page 224-90	Sets the proxy to be used for requests of HTTP or other protocols
<a href="#">SET_RESPONSE_ERROR_CHECK Procedure</a> on page 224-91	Sets whether or not <code>GET_RESPONSE</code> raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges
<a href="#">SET_TRANSFER_TIMEOUT Procedure</a> on page 224-92	Sets the timeout value for UTL_HTTP to read the HTTP response from the Web server or proxy server
<a href="#">SET_WALLET Procedure</a> on page 224-93	Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS

## HTTP Requests Subprograms

**Table 224–10 UTL\_HTTP Subprograms—HTTP Requests**

Subprogram	Description
<a href="#">BEGIN_REQUEST Function</a> on page 224-39	Begins a new HTTP request. <code>UTL_HTTP</code> establishes the network connection to the target Web server or the proxy server and sends the HTTP request line.
<a href="#">SET_HEADER Procedure</a> on page 224-87	Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.
<a href="#">SET_AUTHENTICATION Procedure</a> on page 224-78	Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.
<a href="#">SET_AUTHENTICATION_FROM_WALLET Procedure</a> on page 224-79	Sets the HTTP authentication information in the HTTP request header needed for the request to be authorized by the Web server using the username and password credential stored in the Oracle wallet.
<a href="#">SET_BODY_CHARSET Procedures</a> on page 224-81	Sets the character set of the request body when the media type is text but the character set is not specified in the <code>Content-Type</code> header
<a href="#">SET_COOKIE_SUPPORT Procedures</a> on page 224-83	Enables or disables support for the HTTP cookies in the request
<a href="#">SET_FOLLOW_REDIRECT Procedures</a> on page 224-86	Sets the maximum number of times <code>UTL_HTTP</code> follows the HTTP redirect instruction in the HTTP response to this request in the <a href="#">GET_RESPONSE Function</a> on page 224-66
<a href="#">SET_PERSISTENT_CONN_SUPPORT Procedure</a> on page 224-88	Enables or disables support for the HTTP 1.1 persistent-connection in the request
<a href="#">SET_PROXY Procedure</a> on page 224-90	Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in <code>UTL_TCP</code> )
<a href="#">WRITE_RAW Procedure</a> on page 224-96	Writes some binary data in the HTTP request body
<a href="#">WRITE_TEXT Procedure</a> on page 224-97	Writes some text data in the HTTP request body

---

## HTTP Request Contexts Subprograms

**Table 224–11** *UTL\_HTTP Subprograms—HTTP Request Contexts*

<b>Subprogram</b>	<b>Description</b>
<a href="#">CREATE_REQUEST_CONTEXT Function</a> on page 224-45	Creates a request context in UTL_HTTP for a wallet and a cookie table
<a href="#">DESTROY_REQUEST_CONTEXT Procedure</a> on page 224-47	Destroys a request context in UTL_HTTP

---

## HTTP Responses Subprograms

**Table 224–12** *UTL\_HTTP Subprograms—HTTP Responses*

Subprogram	Description
<a href="#">END_RESPONSE Procedure</a> on page 224-49	Ends the HTTP response. It completes the HTTP request and response.
<a href="#">GET_AUTHENTICATION Procedure</a> on page 224-50	Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header
<a href="#">GET_HEADER Procedure</a> on page 224-59	Returns the n <sup>th</sup> HTTP response header name and value returned in the response
<a href="#">GET_HEADER_BY_NAME Procedure</a> on page 224-60	Returns the HTTP response header value returned in the response given the name of the header
<a href="#">GET_HEADER_COUNT Function</a> on page 224-61	Returns the number of HTTP response headers returned in the response
<a href="#">GET_RESPONSE Function</a> on page 224-66	Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed.
<a href="#">READ_LINE Procedure</a> on page 224-69	Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer
<a href="#">READ_RAW Procedure</a> on page 224-70	Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer
<a href="#">READ_TEXT Procedure</a> on page 224-71	Reads the HTTP response body in text form and returns the output in the caller-supplied buffer
<a href="#">SET_BODY_CHARSET Procedures</a> on page 224-81	Sets the character set of the response body when the media type is "text" but the character set is not specified in the <code>Content-Type</code> header



## HTTP Cookies Subprograms

**Table 224–13** *UTL\_HTTP Subprograms—HTTP Cookies*

<b>Subprogram</b>	<b>Description</b>
<a href="#">ADD_COOKIES Procedure</a> on page 224-38	Add the cookies either to a request context or to the UTL_HTTP package's session state
<a href="#">CLEAR_COOKIES Procedure</a> on page 224-41	Clears all the cookies maintained either in a request context or in the UTL_HTTP package's session state
<a href="#">GET_COOKIE_COUNT Function</a> on page 224-52	Returns the number of cookies maintained either in a request context or in the UTL_HTTP package's session states
<a href="#">GET_COOKIES Function</a> on page 224-54	Returns all the cookies maintained either in a request context or in the UTL_HTTP package's session state.

## HTTP Persistent Connections Subprograms

**Table 224–14** *UTL\_HTTP Subprograms—HTTP Persistent Connections*

<b>Subprogram</b>	<b>Description</b>
<a href="#">CLOSE_PERSISTENT_CONN Procedure</a> on page 224-42	Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session
<a href="#">CLOSE_PERSISTENT_CONNS Procedure</a> on page 224-43	Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session
<a href="#">GET_PERSISTENT_CONN_COUNT Function</a> on page 224-62	Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers
<a href="#">GET_PERSISTENT_CONNS Procedure</a> on page 224-64	Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers

---

## Error Conditions Subprograms

**Table 224–15** *UTL\_HTTP Subprograms—Error Conditions*

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_DETAILED_SQLCODE Function</a> on page 224-56	Retrieves the detailed SQLCODE of the last exception raised
<a href="#">GET_DETAILED_SQLERRM Function</a> on page 224-57	Retrieves the detailed SQLERRM of the last exception raised

---

## Summary of UTL\_HTTP Subprograms

**Table 224–16 UTL\_HTTP Package Subprograms**

Subprogram	Description	Group
<a href="#">ADD_COOKIES Procedure</a> on page 224-38	Add the cookies either to a request context or to the UTL_HTTP package's session state	<a href="#">HTTP Cookies Subprograms</a> on page 224-31
<a href="#">BEGIN_REQUEST Function</a> on page 224-39	Begins a new HTTP request. UTL_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">CLEAR_COOKIES Procedure</a> on page 224-41	Clears all the cookies maintained either in a request context or in the UTL_HTTP package's session state	<a href="#">HTTP Cookies Subprograms</a> on page 224-31
<a href="#">CLOSE_PERSISTENT_CONN Procedure</a> on page 224-42	Closes an HTTP persistent connection maintained by the UTL_HTTP package in the current database session	<a href="#">HTTP Persistent Connections Subprograms</a> on page 224-32
<a href="#">CLOSE_PERSISTENT_CONNS Procedure</a> on page 224-43	Closes a group of HTTP persistent connections maintained by the UTL_HTTP package in the current database session	<a href="#">HTTP Persistent Connections Subprograms</a> on page 224-32
<a href="#">CREATE_REQUEST_CONTEXT Function</a> on page 224-45	Creates a request context in UTL_HTTP for a wallet and a cookie table	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">DESTROY_REQUEST_CONTEXT Procedure</a> on page 224-47	Destroys a request context in UTL_HTTP for a wallet and a cookie table	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">END_REQUEST Procedure</a> on page 224-48	Ends the HTTP request	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">END_RESPONSE Procedure</a> on page 224-49	Ends the HTTP response. It completes the HTTP request and response	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">GET_AUTHENTICATION Procedure</a> on page 224-50	Retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">GET_BODY_CHARSET Procedure</a> on page 224-51	Retrieves the default character set of the body of all future HTTP requests	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">GET_COOKIE_COUNT Function</a> on page 224-52	Returns the number of cookies currently maintained by the UTL_HTTP package set by all Web servers	<a href="#">HTTP Cookies Subprograms</a> on page 224-31
<a href="#">GET_COOKIE_SUPPORT Procedure</a> on page 224-53	Retrieves the current cookie support settings	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">GET_COOKIES Function</a> on page 224-54	Returns all the cookies currently maintained by the UTL_HTTP package set by all Web servers	<a href="#">HTTP Cookies Subprograms</a> on page 224-31

**Table 224–16 (Cont.) UTL\_HTTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">GET_DETAILED_EXCP_SUPPORT Procedure</a> on page 224-55	Checks if the UTL_HTTP package will raise a detailed exception or not	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">GET_DETAILED_SQLCODE Function</a> on page 224-56	Retrieves the detailed <code>SQLCODE</code> of the last exception raised	<a href="#">Error Conditions Subprograms</a> on page 224-33
<a href="#">GET_DETAILED_SQLERRM Function</a> on page 224-57	Retrieves the detailed <code>SQLERRM</code> of the last exception raised	<a href="#">Error Conditions Subprograms</a> on page 224-33
<a href="#">GET_FOLLOW_REDIRECT Procedure</a> on page 224-58	Retrieves the follow-redirect setting in the current session	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">GET_HEADER Procedure</a> on page 224-59	Returns the $n^{\text{th}}$ HTTP response header name and value returned in the response	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">GET_HEADER_BY_NAME Procedure</a> on page 224-60	Returns the HTTP response header value returned in the response given the name of the header	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">GET_HEADER_COUNT Function</a> on page 224-61	Returns the number of HTTP response headers returned in the response	<a href="#">HTTP Responses</a> on page 224-17 and <a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">GET_PERSISTENT_CONN_COUNT Function</a> on page 224-62	Returns the number of network connections currently kept persistent by the UTL_HTTP package to the Web servers	<a href="#">HTTP Persistent Connections Subprograms</a> on page 224-32
<a href="#">GET_HEADER_COUNT Function</a> on page 224-61	Sees whether or not future HTTP requests will support the HTTP 1.1 persistent connection; sets the maximum number of persistent connections maintained in the current database user session	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">GET_PERSISTENT_CONN_SUPPORT Procedure</a> on page 224-63	Checks if the persistent connection support is enabled and gets the maximum number of persistent connections in the current session (see <a href="#">Session Settings Subprograms</a> on page 224-27)	<a href="#">HTTP Persistent Connections Subprograms</a> on page 224-32
<a href="#">GET_PERSISTENT_CONNS Procedure</a> on page 224-64	Returns all the network connections currently kept persistent by the UTL_HTTP package to the Web servers	<a href="#">HTTP Persistent Connections Subprograms</a> on page 224-32
<a href="#">GET_PROXY Procedure</a> on page 224-65	Retrieves the current proxy settings	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">GET_RESPONSE Function</a> on page 224-66	Reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">GET_RESPONSE_ERROR_CHECK Procedure</a> on page 224-67	Checks if the response error check is set or no	<a href="#">Session Settings Subprograms</a> on page 224-27

**Table 224–16 (Cont.) UTL\_HTTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">GET_TRANSFER_TIMEOUT Procedure</a> on page 224-68	Retrieves the current network transfer timeout value	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">READ_LINE Procedure</a> on page 224-69	Reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">READ_RAW Procedure</a> on page 224-70	Reads the HTTP response body in binary form and returns the output in the caller-supplied buffer	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">READ_TEXT Procedure</a> on page 224-71	Reads the HTTP response body in text form and returns the output in the caller-supplied buffer	<a href="#">HTTP Responses Subprograms</a> on page 224-30
<a href="#">REQUEST Function</a> on page 224-73	Returns up to the first 2000 bytes of the data retrieved from the given URL. This function can be used directly in SQL queries.	<a href="#">Simple HTTP Fetches in a Single Call Subprograms</a> on page 224-26
<a href="#">REQUEST_PIECES Function</a> on page 224-75	Returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL	<a href="#">Simple HTTP Fetches in a Single Call Subprograms</a> on page 224-26
<a href="#">SET_AUTHENTICATION Procedure</a> on page 224-78	Sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_AUTHENTICATION_FROM_WALLET Procedure</a> on page 224-79	Sets the HTTP authentication information in the HTTP request header needed for the request to be authorized by the Web server using the username and password credential stored in the Oracle wallet.	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_BODY_CHARSET Procedures</a> on page 224-81	Sets the default character set of the body of all future HTTP requests when the media type is <code>text</code> and the character set is not specified in the <code>Content-Type</code> header	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_BODY_CHARSET Procedures</a> on page 224-81	Sets the character set of the request body when the media type is <code>text</code> but the character set is not specified in the <code>Content-Type</code> header	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_BODY_CHARSET Procedures</a> on page 224-81	Sets the character set of the response body when the media type is "text" but the character set is not specified in the <code>Content-Type</code> header	<a href="#">HTTP Responses Subprograms</a> and <a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_COOKIE_SUPPORT Procedures</a> on page 224-83	Enables or disables support for the HTTP cookies in the request	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_DETAILED_EXCP_SUPPORT Procedure</a> on page 224-85	Sets whether or not future HTTP requests will support HTTP cookies; sets the maximum number of cookies maintained in the current database user session	<a href="#">Session Settings Subprograms</a> on page 224-27

**Table 224–16 (Cont.) UTL\_HTTP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">SET_DETAILED_EXCP_SUPPORT Procedure</a> on page 224-85	Sets the UTL_HTTP package to raise a detailed exception	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_FOLLOW_REDIRECT Procedures</a> on page 224-86	Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP response to this request in the GET_RESPONSE function	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_HEADER Procedure</a> on page 224-87	Sets the maximum number of times UTL_HTTP follows the HTTP redirect instruction in the HTTP responses to future requests in the GET_RESPONSE function	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_HEADER Procedure</a> on page 224-87	Sets an HTTP request header. The request header is sent to the Web server as soon as it is set.	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_PERSISTENT_CONN_SUPPORT Procedure</a> on page 224-88	Enables or disables support for the HTTP 1.1 persistent-connection in the request	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">SET_PROXY Procedure</a> on page 224-90	Sets the proxy to be used for requests of HTTP or other protocols	<a href="#">Session Settings</a> on page 224-17 and <a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_RESPONSE_ERROR_CHECK Procedure</a> on page 224-91	Sets whether or not GET_RESPONSE raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_TRANSFER_TIMEOUT Procedure</a> on page 224-92	Sets the timeout value for UTL_HTTP to read the HTTP response from the Web server or proxy server	<a href="#">Session Settings</a> on page 224-17 and <a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">SET_WALLET Procedure</a> on page 224-93	Sets the Oracle Wallet used for all HTTP requests over Secured Socket Layer (SSL), that is, HTTPS	<a href="#">Session Settings Subprograms</a> on page 224-27
<a href="#">WRITE_LINE Procedure</a> on page 224-95	Writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in UTL_TCP	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">WRITE_RAW Procedure</a> on page 224-96	Writes some binary data in the HTTP request body	<a href="#">HTTP Requests Subprograms</a> on page 224-28
<a href="#">WRITE_TEXT Procedure</a> on page 224-97	Writes some text data in the HTTP request body	<a href="#">HTTP Requests Subprograms</a> on page 224-28

## ADD\_COOKIES Procedure

This procedure adds the cookies either to a request context or to the UTL\_HTTP package's session state.

**See Also:** [HTTP Cookies](#) on page 224-17 and [HTTP Cookies Subprograms](#) on page 224-31

### Syntax

```
UTL_HTTP.ADD_COOKIES (  
    cookies          IN  cookie_table,  
    request_context IN  request_context_key DEFAULT NULL);
```

### Parameters

**Table 224–17 ADD\_COOKIES Procedure Parameters**

Parameter	Description
cookies	The cookies to be added
request_context	Request context to add the cookies. If NULL, the cookies will be added to the UTL_HTTP package's session state instead.

### Usage Notes

The cookies that the package currently maintains are not cleared before new cookies are added.



## BEGIN\_REQUEST Function

This function begins a new HTTP request. UTL\_HTTP establishes the network connection to the target Web server or the proxy server and sends the HTTP request line. The PL/SQL program continues the request by calling some other interface to complete the request. The URL may contain the username and password needed to authenticate the request to the server. The format is

```
scheme://[user[:password]@]host[:port]/[...]
```

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.BEGIN_REQUEST (
    url            IN  VARCHAR2,
    method         IN  VARCHAR2 DEFAULT 'GET',
    http_version   IN  VARCHAR2 DEFAULT NULL,
    request_context IN  request_context_key DEFAULT NULL)
RETURN req;
```

### Parameters

**Table 224–18** *BEGIN\_REQUEST Function Parameters*

Parameter	Description
url	The URL of the HTTP request
method	The method performed on the resource identified by the URL
http_version	The HTTP protocol version that sends the request. The format of the protocol version is HTTP/major-version.minor-version, where major-version and minor-version are positive numbers. If this parameter is set to NULL, UTL_HTTP uses the latest HTTP protocol version that it supports to send the request. The latest version that the package supports is 1.1 and it can be upgraded to a later version. The default is NULL.
request_context	Request context that holds the private wallet and the cookie table to use in this HTTP request. If this parameter is NULL, the wallet and cookie table shared in the current database session will be used instead.

### Usage Notes

- The URL passed as an argument to this function is not examined for illegal characters, such as spaces, according to URL specification RFC 2396. You should escape those characters with the UTL\_URL package to return illegal and reserved characters. URLs should consist of US-ASCII characters only. See [Chapter 238, "UTL\\_URL"](#) for a list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.
- BEGIN\_REQUEST can send a URL whose length is up to 32767 bytes. However, different Web servers impose different limits on the length of the URL they can accept. This limit is often about 4000 bytes. If this limit is exceeded, the outcome will depend on the Web server. For example, a Web server might simply drop the HTTP connection without returning a response of any kind. If this happens, a

subsequent invocation of the [GET\\_RESPONSE Function](#) will raise the `PROTOCOL_ERROR` exception.

A URL will be long when its `QUERY_STRING` (that is, the information that follows the question mark (?)) is long. In general, it is better to send this parameterization in the body of the request using the `POST` method.

```
req := UTL_HTTP.BEGIN_REQUEST (url=>the_url, method=>'POST');
UTL_HTTP.SET_HEADER (r      => req,
                    name   => 'Content-Type',
                    value  => 'application/x-www-form-urlencoded');
UTL_HTTP.SET_HEADER (r      => req,
                    name   => 'Content-Length',
                    value  => '<length of data posted in bytes>');
UTL_HTTP.WRITE_TEXT (r      => req,
                    data   => 'p1 = value1&p2=value2...');
resp := UTL_HTTP.GET_RESPONSE
      (r      => req);
...
```

The programmer must determine whether a particular Web server may, or may not, accept data provided in this way.

- An Oracle wallet must be set before accessing Web servers over HTTPS. See the [SET\\_WALLET Procedure](#) on how to set up an Oracle wallet. To use SSL client authentication, the client certificate should be stored in the wallet and the caller must have the `use-client-certificates` privilege on the wallet. See "Managing Fine-grained Access to External Network Services" in the *Oracle Database Security Guide* to grant the privilege.
- To connect to the remote Web server directly, or indirectly through a HTTP proxy, the `UTL_HTTP` must have the `connect` ACL privilege to the remote Web server host or the proxy host respectively.

## CLEAR\_COOKIES Procedure

This procedure clears all the cookies maintained either in a request context or in the UTL\_HTTP package's session state.

**See Also:** [HTTP Cookies](#) on page 224-17 and [HTTP Cookies Subprograms](#) on page 224-31

### Syntax

```
UTL_HTTP.CLEAR_COOKIES (  
    request_context IN request_context_key DEFAULT NULL);
```

### Parameters

**Table 224–19** CLEAR\_COOKIES Procedure Parameters

Parameter	Description
request_context	Request context to clear the cookies. If NULL, the cookies maintained in the UTL_HTTP package's session state will be cleared instead.

## CLOSE\_PERSISTENT\_CONN Procedure

This procedure closes an HTTP persistent connection maintained by the UTL\_HTTP package in the current database session.

**See Also:** [HTTP Persistent Connections](#) on page 224-17 and [HTTP Persistent Connections Subprograms](#) on page 224-32

### Syntax

```
UTL_HTTP.CLOSE_PERSISTENT_CONN (  
    conn IN connection);
```

### Parameters

**Table 224–20** *CLOSE\_PERSISTENT\_CONN Procedure Parameters*

Parameter	Description
conn	The HTTP persistent connection to close

## CLOSE\_PERSISTENT\_CONNS Procedure

This procedure closes a group of HTTP persistent connections maintained by the UTL\_HTTP package in the current database session. This procedure uses a pattern-match approach to decide which persistent connections to close.

To close a group of HTTP persistent connection that share a common property (for example, all connections to a particular host, or all SSL connections), set the particular parameters and leave the rest of the parameters NULL. If a particular parameter is set to NULL when this procedure is called, that parameter will not be used to decide which connections to close.

For example, the following call to the procedure closes all persistent connections to foobar:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS(host => 'foobar');
```

And the following call to the procedure closes all persistent connections through the proxy www-proxy at TCP/IP port 80:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS(proxy_host => 'foobar',
                                proxy_port => 80);
```

And the following call to the procedure closes all persistent connections:

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS;
```

**See Also:** [HTTP Persistent Connections](#) on page 224-17 and [HTTP Persistent Connections Subprograms](#) on page 224-32

### Syntax

```
UTL_HTTP.CLOSE_PERSISTENT_CONNS (
  host      IN VARCHAR2 DEFAULT NULL,
  port      IN PLS_INTEGER DEFAULT NULL,
  proxy_host IN VARCHAR2 DEFAULT NULL,
  proxy_port IN PLS_INTEGER DEFAULT NULL,
  ssl       IN BOOLEAN DEFAULT NULL);
```

### Parameters

**Table 224–21** *CLOSE\_PERSISTENT\_CONNS Procedure Parameters*

Parameter	Description
host	The host for which persistent connections are to be closed
port	The port number for which persistent connections are to be closed
proxy_host	The proxy host for which persistent connections are to be closed
proxy_port	The proxy port for which persistent connections are to be closed
ssl	Close persistent SSL connection

### Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL

of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

Note that the use of a `NULL` value in a parameter when this procedure is called means that the caller does not care about its value when the package decides which persistent connection to close. If you want a `NULL` value in a parameter to match only a `NULL` value of the parameter of a persistent connection (which is when you want to close a specific persistent connection), you should use the `CLOSE_PERSISTENT_CONN` procedure that closes a specific persistent connection.

## CREATE\_REQUEST\_CONTEXT Function

This function creates a request context. A request context is a context that holds a wallet and a cookie for private use in making a HTTP request. This allows the HTTP request to use a wallet and a cookie table that will not be shared with other applications making HTTP requests in the same database session.

**See Also:** [Request Context](#) on page 224-18 and [HTTP Request Contexts Subprograms](#) on page 224-29

### Syntax

```
UTL_HTTP.CREATE_REQUEST_CONTEXT (
  wallet_path          IN VARCHAR2 DEFAULT NULL,
  wallet_password     IN VARCHAR2 DEFAULT NULL,
  enable_cookies      IN BOOLEAN  DEFAULT TRUE,
  max_cookies         IN PLS_INTEGER DEFAULT 300,
  max_cookies_per_site IN PLS_INTEGER DEFAULT 20)
RETURN request_context_key;
```

### Parameters

**Table 224–22 CREATE\_REQUEST\_CONTEXT Function Parameters**

Parameter	Description
wallet_path	Directory path that contains the Oracle wallet. The format is <code>file:directory-path</code>
wallet_password	The password needed to open the wallet. If the wallet is auto-login enabled, the password may be omitted and should be set to NULL. See the <i>Oracle Database Advanced Security Administrator's Guide</i> for detailed information about wallets.
enable_cookies	Sets whether HTTP requests using this request context should support HTTP cookies or not: TRUE to enable the support, FALSE to disable it.
max_cookies	Sets the maximum total number of cookies that will be maintained in this request context
max_cookies_per_site	Sets the maximum number of cookies per each Web site that will be maintained in this request context

### Return Values

The request context created.

### Examples

```
DECLARE
  request_context UTL_HTTP.REQUEST_CONTEXT_KEY;
  req            utl_http.req;
BEGIN
  request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(
    wallet_path          => 'file:/oracle/wallets/test_wallets',
    wallet_password     => NULL,
    enable_cookies      => TRUE,
    max_cookies         => 300,
    max_cookies_per_site => 20);
  req := UTL_HTTP.BEGIN_REQUEST(
    url                  => 'http://www.example.com/',
```

```
        request_context => request_context);  
END;
```



## DESTROY\_REQUEST\_CONTEXT Procedure

This procedure destroys a request context in UTL\_HTTP. A request context cannot be destroyed when it is in use by a HTTP request or response.

**See Also:** [Request Context](#) on page 224-18 and [HTTP Request Contexts Subprograms](#) on page 224-29

### Syntax

```
UTL_HTTP.DESTROY_REQUEST_CONTEXT (
    request_context    request_context_key);
```

### Parameters

**Table 224–23 DESTROY\_REQUEST\_CONTEXT Procedure Parameters**

Parameter	Description
request_context	Request context to destroy

### Examples

```
DECLARE
    request_context    UTL_HTTP.REQUEST_CONTEXT_KEY;
BEGIN
    request_context := UTL_HTTP.CREATE_REQUEST_CONTEXT(...);
    ...
    UTL_HTTP.DESTROY_REQUEST_CONTEXT(request_context);
END;
```

## END\_REQUEST Procedure

This procedure ends the HTTP request. To terminate the HTTP request without completing the request and waiting for the response, the program can call this procedure. Otherwise, the program should go through the normal sequence of beginning a request, getting the response, and closing the response. The network connection will always be closed and will not be reused.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.END_REQUEST (  
    r IN OUT NOCOPY req);
```

### Parameters

**Table 224–24** *END\_REQUEST Procedure Parameters*

Parameter	Description
r	The HTTP request

## END\_RESPONSE Procedure

This procedure ends the HTTP response. It completes the HTTP request and response. Unless HTTP 1.1 persistent connection is used in this request, the network connection is also closed.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.END_RESPONSE (  
    r IN OUT NOCOPY resp);
```

### Parameters

**Table 224–25** *END\_RESPONSE Procedure Parameters*

Parameter	Description
r	The HTTP response

## GET\_AUTHENTICATION Procedure

This procedure retrieves the HTTP authentication information needed for the request to be accepted by the Web server as indicated in the HTTP response header.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.GET_AUTHENTICATION(
  r          IN OUT NOCOPY resp,
  scheme     OUT VARCHAR2,
  realm      OUT VARCHAR2,
  for_proxy  IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 224–26** GET\_AUTHENTICATION Procedure Parameters

Parameter	Description
r	The HTTP response
scheme	The scheme for the required HTTP authentication
realm	The realm for the required HTTP authentication
for_proxy	Returns the HTTP authentication information required for the access to the HTTP proxy server instead of the Web server? Default is FALSE

### Usage Notes

When a Web client is unaware that a document is protected, at least two HTTP requests are required for the document to be retrieved. In the first HTTP request, the Web client makes the request without supplying required authentication information; so the request is denied. The Web client can determine the authentication information required for the request to be authorized by calling GET\_AUTHENTICATION. The Web client makes the second request and supplies the required authentication information with SET\_AUTHORIZATION. If the authentication information can be verified by the Web server, the request will succeed and the requested document is returned. Before making the request, if the Web client knows that authentication information is required, it can supply the required authentication information in the first request, thus saving an extra request.

## GET\_BODY\_CHARSET Procedure

This procedure retrieves the default character set of the body of all future HTTP requests.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_BODY_CHARSET (  
    charset OUT NOCOPY VARCHAR2);
```

### Parameters

**Table 224–27** *GET\_BODY\_CHARSET Procedure Parameters*

Parameter	Description
charset	The default character set of the body of all future HTTP requests

## GET\_COOKIE\_COUNT Function

This function returns the number of cookies maintained either in a request context or in the UTL\_HTTP package's session state.

**See Also:** [HTTP Cookies](#) on page 224-17 and [HTTP Cookies Subprograms](#) on page 224-31

### Syntax

```
UTL_HTTP.GET_COOKIE_COUNT (  
    request_context IN request_context_key DEFAULT NULL)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 224–28** GET\_COOKIE\_COUNT Function Parameters

Parameter	Description
request_context	Request context to return the cookie count for. If NULL, the cookie count maintained in the UTL_HTTP package's session state will be returned instead.

## GET\_COOKIE\_SUPPORT Procedure

This procedure retrieves the current cookie support settings.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_COOKIE_SUPPORT (
    enable           OUT BOOLEAN,
    max_cookies      OUT PLS_INTEGER,
    max_cookies_per_site OUT PLS_INTEGER);
```

### Parameters

**Table 224–29** *GET\_COOKIE\_SUPPORT Procedure Parameters*

Parameter	Description
enable	Indicates whether future HTTP requests should support HTTP cookies (TRUE) or not (FALSE)
max_cookies	Indicates the maximum total number of cookies maintained in the current session
max_cookies_per_site	Indicates the maximum number of cookies maintained in the current session for each Web site

## GET\_COOKIES Function

This function returns all the cookies maintained either in a request context or in the UTL\_HTTP package's session state.

**See Also:** [HTTP Cookies](#) on page 224-17 and [HTTP Cookies Subprograms](#) on page 224-31

### Syntax

```
UTL_HTTP.GET_COOKIES (  
    cookies          IN OUT NOCOPY cookie_table,  
    request_context IN          request_context_key DEFAULT NULL);
```

### Parameters

**Table 224–30** GET\_COOKIES Procedure Parameters

Parameter	Description
cookies	The cookies returned
request_context	Request context to return the cookies for. If NULL, the cookies maintained in the UTL_HTTP package's session state will be returned instead.



## GET\_DETAILED\_EXCP\_SUPPORT Procedure

This procedure checks if the UTL\_HTTP package will raise a detailed exception or not.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_DETAILED_EXCP_SUPPORT (  
    enable OUT BOOLEAN);
```

### Parameters

**Table 224–31** GET\_DETAILED\_EXCP\_SUPPORT Procedure Parameters

Parameter	Description
enable	TRUE if UTL_HTTP raises a detailed exception; otherwise FALSE

## GET\_DETAILED\_SQLCODE Function

This function retrieves the detailed SQLCODE of the last exception raised.

**See Also:** [Error Conditions](#) on page 224-17 and [Error Conditions Subprograms](#) on page 224-33

### Syntax

```
UTL_HTTP.GET_DETAILED_SQLCODE  
RETURN PLS_INTEGER;
```

## GET\_DETAILED\_SQLERRM Function

This function retrieves the detailed `SQLERRM` of the last exception raised.

**See Also:** [Error Conditions](#) on page 224-17 and [Error Conditions Subprograms](#) on page 224-33

### Syntax

```
UTL_HTTP.GET_DETAILED_SQLERRM  
RETURN VARCHAR2;
```

## GET\_FOLLOW\_REDIRECT Procedure

This procedure retrieves the follow-redirect setting in the current session

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_FOLLOW_REDIRECT (  
    max_redirects OUT PLS_INTEGER);
```

### Parameters

**Table 224–32 GET\_FOLLOW\_REDIRECT Procedure Parameters**

Parameter	Description
max_redirects	The maximum number of redirections for all future HTTP requests

## GET\_HEADER Procedure

This procedure returns the  $n^{\text{th}}$  HTTP response header name and value returned in the response.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.GET_HEADER (
  r      IN OUT NOCOPY resp,
  n      IN PLS_INTEGER,
  name   OUT NOCOPY VARCHAR2,
  value  OUT NOCOPY VARCHAR2);
```

### Parameters

**Table 224–33** GET\_HEADER Procedure Parameters

Parameter	Description
r	The HTTP response
n	The $n^{\text{th}}$ header to return
name	The name of the HTTP response header
value	The value of the HTTP response header

### Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

## GET\_HEADER\_BY\_NAME Procedure

This procedure returns the HTTP response header value returned in the response given the name of the header.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.GET_HEADER_BY_NAME(  
    r      IN OUT NOCOPY resp,  
    name   IN VARCHAR2,  
    value  OUT NOCOPY VARCHAR2,  
    n      IN PLS_INTEGER DEFAULT 1);
```

### Parameters

**Table 224–34** *GET\_HEADER\_BY\_NAME Procedure Parameters*

Parameter	Description
r	The HTTP response
name	The name of the HTTP response header for which the value is to return
value	The value of the HTTP response header
n	The n <sup>th</sup> occurrence of an HTTP response header by the specified name to return. The default is 1.

### Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

## GET\_HEADER\_COUNT Function

This function returns the number of HTTP response headers returned in the response.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.GET_HEADER_COUNT (  
    r IN OUT NOCOPY resp)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 224–35** *GET\_HEADER\_COUNT Function Parameters*

Parameter	Description
r	The HTTP response

### Usage Notes

If the response body returned by the remote Web server is encoded in chunked transfer encoding format, the trailer headers that are returned at the end of the response body will be added to the response, and the response header count will be updated. You can retrieve the additional headers after the end of the response body is reached and before you end the response.

## GET\_PERSISTENT\_CONN\_COUNT Function

This function returns the number of network connections currently kept persistent by the UTL\_HTTP package to the Web servers.

**See Also:** [HTTP Persistent Connections](#) on page 224-17 and [HTTP Persistent Connections Subprograms](#) on page 224-32

### Syntax

```
UTL_HTTP.GET_PERSISTENT_CONN_COUNT  
RETURN PLS_INTEGER;
```

### Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.



## GET\_PERSISTENT\_CONN\_SUPPORT Procedure

This procedure checks:

- If the persistent connection support is enabled
- Gets the maximum number of persistent connections in the current session

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_PERSISTENT_CONN_SUPPORT (  
    enable      OUT BOOLEAN,  
    max_conns  OUT PLS_INTEGER);
```

### Parameters

**Table 224–36** *GET\_PERSISTENT\_CONN\_SUPPORT Procedure Parameters*

Parameter	Description
enable	TRUE if persistent connection support is enabled; otherwise FALSE
max_conns	the maximum number of persistent connections maintained in the current session

## GET\_PERSISTENT\_CONNS Procedure

This procedure returns all the network connections currently kept persistent by the UTL\_HTTP package to the Web servers.

**See Also:** [HTTP Persistent Connections](#) on page 224-17 and [HTTP Persistent Connections Subprograms](#) on page 224-32

### Syntax

```
UTL_HTTP.get_persistent_conns (  
    connections IN OUT NOCOPY connection_table);
```

### Parameters

**Table 224–37 GET\_PERSISTENT\_CONNS Procedure Parameters**

Parameter	Description
connections	The network connections kept persistent

### Usage Notes

Connections to the same Web server at different TCP/IP ports are counted individually. The host names of the Web servers are identified as specified in the URL of the original HTTP requests. Therefore, fully qualified host names with domain names will be counted differently from the host names without domain names.

## GET\_PROXY Procedure

This procedure retrieves the current proxy settings.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_PROXY (  
    proxy          OUT NOCOPY VARCHAR2,  
    no_proxy_domains OUT NOCOPY VARCHAR2);
```

### Parameters

**Table 224–38** *GET\_PROXY Procedure Parameters*

Parameter	Description
proxy	The proxy (host and an optional port number) currently used by the UTL_HTTP package
no_proxy_domains	The list of hosts and domains for which no proxy is used for all requests

## GET\_RESPONSE Function

This function reads the HTTP response. When the function returns, the status line and the HTTP response headers have been read and processed. The status code, reason phrase, and the HTTP protocol version are stored in the response record. This function completes the HTTP headers section.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.GET_RESPONSE (  
    r IN OUT NOCOPY req)  
RETURN resp;
```

### Parameters

**Table 224–39** *GET\_RESPONSE Procedure Parameters*

Parameter	Description
r	The HTTP response

## GET\_RESPONSE\_ERROR\_CHECK Procedure

This procedure checks if the response error check is set or not.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_RESPONSE_ERROR_CHECK (  
    enable OUT BOOLEAN);
```

### Parameters

**Table 224–40** *GET\_RESPONSE\_ERROR\_CHECK Procedure Parameters*

Parameter	Description
enable	TRUE if the response error check is set; otherwise FALSE

## GET\_TRANSFER\_TIMEOUT Procedure

This procedure retrieves the default timeout value for all future HTTP requests.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.GET_TRANSFER_TIMEOUT (  
    timeout OUT PLS_INTEGER);
```

### Parameters

**Table 224–41 GET\_TRANSFER\_TIMEOUT Procedure Parameters**

Parameter	Description
timeout	The network transfer timeout value in seconds

## READ\_LINE Procedure

This procedure reads the HTTP response body in text form until the end of line is reached and returns the output in the caller-supplied buffer. The end of line is as defined in the function `read_line` of `UTL_TCP`. The `end_of_body` exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.READ_LINE(
  r          IN OUT NOCOPY resp,
  data       OUT NOCOPY  VARCHAR2 CHARACTER SET ANY_CS,
  remove_crlf IN  BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 224-42 READ\_LINE Procedure Parameters**

Parameter	Description
<code>r</code>	The HTTP response
<code>data</code>	The HTTP response body in text form
<code>remove_crlf</code>	Removes the newline characters if set to TRUE

### Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, `read_line` waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, `read_line` stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `read_raw` procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

When the `Content-Type` response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482: unsupported character set" exception is raised if you try to read the response body as text. You can either read the response body as binary using the `READ_RAW` procedure, or set the character set of the response body explicitly using the `SET_BODY_CHARSET` procedure and read the response body as text again.

## READ\_RAW Procedure

This procedure reads the HTTP response body in binary form and returns the output in the caller-supplied buffer. The `end_of_body` exception will be raised if the end of the HTTP response body is reached.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.READ_RAW(
  r      IN OUT NOCOPY resp,
  data   OUT NOCOPY RAW,
  len    IN PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 224–43** READ\_RAW Procedure Parameters

Parameter	Description
r	The HTTP response
data	The HTTP response body in binary form
len	The number of bytes of data to read. If len is NULL, this procedure will read as much input as possible to fill the buffer allocated in data. The actual amount of data returned may be less than that specified if not much data is available before the end of the HTTP response body is reached or the <code>transfer_timeout</code> amount of time has elapsed. The default is NULL

### Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If transfer timeout is set in the request of this response, `read_raw` waits for each data packet to be ready to read until timeout occurs. If it occurs, `read_raw` stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.



## READ\_TEXT Procedure

This procedure reads the HTTP response body in text form and returns the output in the caller-supplied buffer. The `end_of_body` exception will be raised if the end of the HTTP response body is reached. Text data is automatically converted from the response body character set to the database character set.

**See Also:** [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30

### Syntax

```
UTL_HTTP.READ_TEXT(
  r      IN OUT NOCOPY resp,
  data   OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
  len    IN PLS_INTEGER DEFAULT NULL);
```

### Parameters

**Table 224-44** *READ\_TEXT Procedure Parameters*

Parameter	Description
<code>r</code>	The HTTP response
<code>data</code>	The HTTP response body in text form
<code>len</code>	The maximum number of characters of data to read. If <code>len</code> is NULL, this procedure will read as much input as possible to fill the buffer allocated in <code>data</code> . The actual amount of data returned may be less than that specified if little data is available before the end of the HTTP response body is reached or the <code>transfer_timeout</code> amount of time has elapsed. The default is NULL.

### Usage Notes

The `UTL_HTTP` package supports HTTP 1.1 chunked transfer-encoding. When the response body is returned in chunked transfer-encoding format as indicated in the response header, the package automatically decodes the chunks and returns the response body in de-chunked format.

If `transfer_timeout` is set in the request of this response, `read_text` waits for each data packet to be ready to read until timeout occurs. If it occurs, this procedure stops reading and returns all the data read successfully. If no data is read successfully, the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of the response body, `read_text` stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `read_raw` procedure. If a partial multibyte character is seen in the middle of the response body because the remaining bytes of the character have not arrived and read timeout occurs, the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

When the `Content-Type` response header specifies the character set of the response body and the character set is unknown or unsupported by Oracle, the "ORA-01482 :

`unsupported character set` exception is raised if you try to read the response body as text. You can either read the response body as binary using the `READ_RAW` procedure, or set the character set of the response body explicitly using the `SET_BODY_CHARSET` procedure and read the response body as text again.

## REQUEST Function

This function returns up to the first 2000 bytes of data retrieved from the given URL. This function can be used directly in SQL queries. The URL may contain the username and password needed to authenticate the request to the server. The format is

```
scheme://[user[:password]@]host[:port]/[...]
```

You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port][/]
```

**See Also:** [Simple HTTP Fetches](#) on page 224-16 and [Simple HTTP Fetches in a Single Call Subprograms](#) on page 224-26

### Syntax

```
UTL_HTTP.REQUEST (
    url          IN VARCHAR2,
    proxy        IN VARCHAR2 DEFAULT NULL,
    wallet_path  IN VARCHAR2 DEFAULT NULL,
    wallet_password IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Pragmas

```
pragma restrict_references (request, wnds, rnds, wnps, rnps);
```

### Parameters

**Table 224–45** *REQUEST Function Parameters*

Parameter	Description
url	Uniform resource locator
proxy	(Optional) Specifies a proxy server to use when making the HTTP request. See SET_PROXY for the full format of the proxy setting.
wallet_path	(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request. The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\username\WALLETS, and in Unix is, for example, file:/home/username/wallets  When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server. See SET_WALLET for a description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.
wallet_password	(Optional) Specifies the password required to open the wallet

### Return Values

The return type is a string of length 2000 or less, which contains up to the first 2000 bytes of the HTML result returned from the HTTP request to the argument URL.

## Exceptions

```
INIT_FAILED
REQUEST_FAILED
```

## Usage Notes

The URL passed as an argument to this function is not examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL\_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Please see the documentation of the function SET\_WALLET on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

```
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>
<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HTTPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

## Examples

```
SQL> SELECT UTL_HTTP.REQUEST('http://www.my-company.com/') FROM DUAL;
UTL_HTTP.REQUEST('HTTP://WWW.MY-COMPANY.COM/')
<html>
<head><title>My Company Home Page</title>
<!--changed Jan. 16, 19
1 row selected.
```

If you are behind a firewall, include the proxy parameter. For example, from within the Oracle firewall, where there might be a proxy server named www-proxy.my-company.com:

```
SQLPLUS> SELECT
UTL_HTTP.REQUEST('http://www.my-company.com', 'www-proxy.us.my-company.com') FROM
DUAL;
```

## REQUEST\_PIECES Function

This function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL. You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port] [/]
```

**See Also:** [Simple HTTP Fetches](#) on page 224-16 and [Simple HTTP Fetches in a Single Call Subprograms](#) on page 224-26

### Syntax

```
TYPE html_pieces IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;

UTL_HTTP.REQUEST_PIECES (
    url            IN VARCHAR2,
    max_pieces    IN NATURAL DEFAULT 32767,
    proxy         IN VARCHAR2 DEFAULT NULL,
    wallet_path   IN VARCHAR2 DEFAULT NULL,
    wallet_password IN VARCHAR2 DEFAULT NULL)
RETURN html_pieces;
```

### Pragmas

```
PRAGMA RESTRICT_REFERENCES (request_pieces, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 224–46** *REQUEST\_PIECES Function Parameters*

Parameter	Description
url	Uniform resource locator
max_pieces	(Optional) The maximum number of pieces (each 2000 characters in length, except for the last, which may be shorter), that REQUEST_PIECES should return. If provided, then that argument should be a positive integer.
proxy	(Optional) Specifies a proxy server to use when making the HTTP request. See SET_PROXY for the full format of the proxy setting.
wallet_path	(Optional) Specifies a client-side wallet. The client-side wallet contains the list of trusted certificate authorities required for HTTPS request.  The format of wallet_path on a PC is, for example, file:c:\WINNT\Profiles\username\WALLETS, and in Unix is, for example, file:/home/username/wallets. When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server.  See SET_WALLET for the description on how to set up an Oracle wallet. Non-HTTPS requests do not require an Oracle wallet.
wallet_password	(Optional) Specifies the password required to open the wallet

## Return Values

REQUEST\_PIECES returns a PL/SQL table of type UTL\_HTTP.HTML\_PIECES. Each element of that PL/SQL table is a string of maximum length 2000. The elements of the PL/SQL table returned by REQUEST\_PIECES are successive pieces of the data obtained from the HTTP request to that URL.

## Exceptions

INIT\_FAILED  
REQUEST\_FAILED

## Usage Notes

The URL passed as an argument to this function will not be examined for illegal characters, for example, spaces, according to URL specification RFC 2396. The caller should escape those characters with the UTL\_URL package. See the comments of the package for the list of legal characters in URLs. Note that URLs should consist of US-ASCII characters only. The use of non-US-ASCII characters in a URL is generally unsafe.

Each entry of the PL/SQL table (the "pieces") returned by this function may not be filled to their fullest capacity. The function may start filling the data in the next piece before the previous "piece" is totally full.

Please see the documentation of the function SET\_WALLET on the use of an Oracle wallet, which is required for accessing HTTPS Web servers.

Unless response error check is turned on, this function does not raise an exception when a 4xx or 5xx response is received from the Web server. Instead, it returns the formatted error message from the Web server:

```
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>
<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HITPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

## Examples

```
SET SERVEROUTPUT ON

DECLARE
  x  UTL_HTTP.HTML_PIECES;
  len PLS_INTEGER;
BEGIN
  x := UTL_HTTP.REQUEST_PIECES('http://www.oracle.com/', 100);
  DBMS_OUTPUT.PUT_LINE(x.count || ' pieces were retrieved. ');
  DBMS_OUTPUT.PUT_LINE('with total length ');
  IF x.count < 1 THEN
    DBMS_OUTPUT.PUT_LINE('0');
```

```
ELSE
  len := 0;
  FOR i in 1..x.count LOOP
    len := len + length(x(i));
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(i);
END IF;
END;
/
-- Output
Statement processed.
4 pieces were retrieved.
with total length
7687
```

## SET\_AUTHENTICATION Procedure

This procedure sets HTTP authentication information in the HTTP request header. The Web server needs this information to authorize the request.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.SET_AUTHENTICATION(
    r           IN OUT NOCOPY req,
    username   IN VARCHAR2,
    password   IN VARCHAR2,
    scheme     IN VARCHAR2 DEFAULT 'Basic',
    for_proxy  IN BOOLEAN  DEFAULT FALSE);
```

### Parameters

**Table 224–47** SET\_AUTHENTICATION Procedure Parameters

Parameter	Description
r	HTTP request
username	Username for the HTTP authentication
password	Password for the HTTP authentication
scheme	HTTP authentication scheme. Either <code>Basic</code> for the HTTP basic or <code>AWS</code> for Amazon S3 authentication scheme. Default is <code>basic</code> .
for_proxy	Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is <code>FALSE</code> .

### Usage Notes

The supported authentication schemes are HTTP basic and Amazon S3 authentication.



## SET\_AUTHENTICATION\_FROM\_WALLET Procedure

This procedure sets the HTTP authentication information in the HTTP request header needed for the request to be authorized by the Web server using the username and password credential stored in the Oracle wallet.

**See Also:** [External Password Store](#) on page 224-18, and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(
  r          IN OUT NOCOPY req,
  alias      IN VARCHAR2,
  scheme     IN VARCHAR2 DEFAULT 'Basic',
  for_proxy  IN BOOLEAN   DEFAULT FALSE);
```

### Parameters

**Table 224–48 SET\_AUTHENTICATION\_FROM\_WALLET Procedure Parameters**

Parameter	Description
r	The HTTP request
alias	Alias to identify and retrieve the username and password credential stored in the Oracle wallet
scheme	HTTP authentication scheme. Either <code>Basic</code> for the HTTP basic or <code>AWS</code> for Amazon S3 authentication scheme. Default is <code>basic</code> .
for_proxy	Identifies if the HTTP authentication information is for access to the HTTP proxy server instead of the Web server. Default is <code>FALSE</code> .

### Usage Notes

- To use the password credentials in a wallet, the `UTL_HTTP` user must have the `use-passwords` privilege on the wallet.
- The supported authentication schemes are HTTP basic and Amazon S3 authentication schemes.

### Examples

#### Creating a wallet and entering username and password in the wallet

```
> mkstore -wrl /oracle/wallets/test_wallet -create
Enter password: *****
Enter password again: *****
> mkstore -wrl /oracle/wallets/test_wallet -createCredential hr-access jsmith
Your secret/Password is missing in the command line
Enter your secret/Password: ****
Re-enter your secret/Password: ****
Enter wallet password: *****
```

#### Granting the use-passwords privilege on the wallet to a user by the database administrator

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
```

```
    acl          => 'wallet-acl.xml',
    description => 'Wallet ACL',
    principal    => 'SCOTT',
    is_grant     => TRUE,
    privilege    => 'use-passwords');
DBMS_NETWORK_ACL_ADMIN.ASSIGN_WALLET_acl(
    acl          => 'wallet-acl.xml',
    wallet_path  => 'file: /oracle/wallets/test_wallet');
END;
```

### Using username and password from the wallet

```
DECLARE
    req UTL_HTTP.req;
BEGIN
    UTL_HTTP.SET_WALLET(path => 'file:/oracle/wallets/test_wallet');
    req := UTL_HTTP.BEGIN_REQUEST(...);
    UTL_HTTP.SET_AUTHENTICATION_FROM_WALLET(req, 'hr-access');
    ...
END;
```

## SET\_BODY\_CHARSET Procedures

This procedure is overloaded. The description of different functionality is located alongside the syntax declarations.

### See Also:

- [HTTP Responses](#) on page 224-17 and [HTTP Responses Subprograms](#) on page 224-30
- [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

Sets the default character set of the body of all future HTTP requests when the media type is `text` and the character set is not specified in the `Content-Type` header. Following the HTTP protocol standard specification, if the media type of a request or a response is `text`, but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to ISO-8859-1. A response created for a request inherits the default body character set of the request instead of the body character set of the current session. The default body character set is ISO-8859-1 in a database user session. The default body character set setting affects only future requests and has no effect on existing requests. After a request is created, the body character set can be changed by using the other `SET_BODY_CHARSET` procedure that operates on a request:

```
UTL_HTTP.SET_BODY_CHARSET (
    charset IN VARCHAR2 DEFAULT NULL);
```

Sets the character set of the request body when the media type is `text` but the character set is not specified in the `Content-Type` header. According to the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to "ISO-8859-1". Use this procedure to change the default body character set a request inherits from the session default setting:

```
UTL_HTTP.SET_BODY_CHARSET(
    r          IN OUT NOCOPY req,
    charset IN VARCHAR2 DEFAULT NULL);
```

Sets the character set of the response body when the media type is "text" but the character set is not specified in the `Content-Type` header. For each the HTTP protocol standard specification, if the media type of a request or a response is "text" but the character set information is missing in the `Content-Type` header, the character set of the request or response body should default to "ISO-8859-1". Use this procedure to change the default body character set a response inherits from the request:

```
UTL_HTTP.SET_BODY_CHARSET(
    r          IN OUT NOCOPY resp,
    charset IN VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 224–49** *SET\_BODY\_CHARSET Procedure Parameters*

<b>Parameter</b>	<b>Description</b>
<code>r</code>	The HTTP response.
<code>charset</code>	The default character set of the response body. The character set can be in Oracle or Internet Assigned Numbers Authority (IANA) naming convention. If <code>charset</code> is <code>NULL</code> , the database character set is assumed.

## SET\_COOKIE\_SUPPORT Procedures

This procedure is overloaded. The description of different functionality is located alongside the syntax declarations.

This procedure

### See Also:

- [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28
- [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

## Syntax

Enables or disables support for the HTTP cookies in the request. Use this procedure to change the cookie support setting a request inherits from the session default setting:

```
UTL_HTTP.SET_COOKIE_SUPPORT(
  r          IN OUT NOCOPY REQ,
  enable    IN BOOLEAN DEFAULT TRUE);
```

Sets whether or not future HTTP requests will support HTTP cookies, and the maximum number of cookies maintained in the current database user session:

```
UTL_HTTP.SET_COOKIE_SUPPORT (
  enable      IN BOOLEAN,
  max_cookies IN PLS_INTEGER DEFAULT 300,
  max_cookies_per_site IN PLS_INTEGER DEFAULT 20);
```

## Parameters

**Table 224–50 SET\_COOKIE\_SUPPORT Procedure Parameters**

Parameter	Description
r	The HTTP request
enable	Set enable to TRUE to enable HTTP cookie support; FALSE to disable
max_cookies	Sets the maximum total number of cookies maintained in the current session
max_cookies_per_site	Sets the maximum number of cookies maintained in the current session for each Web site

## Usage Notes

If cookie support is enabled for an HTTP request, all cookies saved in the current session and applicable to the request are returned to the Web server in the request in accordance with HTTP cookie specification standards. Cookies set in the response to the request are saved in the current session for return to the Web server in the subsequent requests if cookie support is enabled for those requests. If the cookie support is disabled for an HTTP request, no cookies are returned to the Web server in the request and the cookies set in the response to the request are not saved in the current session, although the Set-Cookie HTTP headers can still be retrieved from the response.

Cookie support is enabled by default for all HTTP requests in a database user session. The default setting of the cookie support (enabled versus disabled) affects only the future requests and has no effect on the existing ones. After your request is created, the cookie support setting may be changed by using the other `SET_COOKIE_SUPPORT` procedure that operates on a request.

The default maximum number of cookies saved in the current session is 20 for each site and 300 total.

If you lower the maximum total number of cookies or the maximum number of cookies for each Web site, the oldest cookies will be purged first to reduce the number of cookies to the lowered maximum. HTTP cookies saved in the current session last for the duration of the database session only; there is no persistent storage for the cookies. Cookies saved in the current session are not cleared if you disable cookie support.

See ["Examples"](#) on page 224-21 for how to use `GET_COOKIES` and `ADD_COOKIES` to retrieve, save, and restore cookies.

## SET\_DETAILED\_EXCP\_SUPPORT Procedure

This procedure sets the UTL\_HTTP package to raise a detailed exception. By default, UTL\_HTTP raises the `request_failed` exception when an HTTP request fails. Use `GET_DETAILED_SQLCODE` and `GET_DETAILED_SQLERRM` for more detailed information about the error.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.SET_DETAILED_EXCP_SUPPORT (  
    enable IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 224–51 SET\_DETAILED\_EXCP\_SUPPORT Procedure Parameters**

Parameter	Description
<code>enable</code>	Asks UTL_HTTP to raise a detailed exception directly if set to TRUE; otherwise FALSE

## SET\_FOLLOW\_REDIRECT Procedures

This procedure sets the maximum number of times UTL\_HTTP follows the HTTP redirect instruction in the HTTP response to this request, or future requests, in the GET\_RESPONSE function.

### See Also:

- [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28
- [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

## Syntax

Use this procedure to set the maximum number of redirections:

```
UTL_HTTP.SET_FOLLOW_REDIRECT (
    max_redirects IN PLS_INTEGER DEFAULT 3);
```

Use this procedure to change the maximum number of redirections a request inherits from the session default setting:

```
UTL_HTTP.SET_FOLLOW_REDIRECT (
    r              IN OUT NOCOPY req,
    max_redirects IN PLS_INTEGER DEFAULT 3);
```

## Parameters

**Table 224–52 SET\_FOLLOW\_REDIRECT Procedure Parameters**

Parameter	Description
r	The HTTP request
max_redirects	The maximum number of redirects. Set to zero to disable redirects.

## Usage Notes

If `max_redirects` is set to a positive number, the [GET\\_RESPONSE Function](#) will automatically follow the redirected URL for the HTTP response status code 301, 302, and 307 for the HTTP HEAD and GET methods, and 303 for all HTTP methods, and retry the HTTP request (the request method will be changed to HTTP GET for the status code 303) at the new location. It follows the redirection until the final, non-redirect location is reached, or an error occurs, or the maximum number of redirections has been reached (to prevent an infinite loop). The URL and method fields in the REQ record will be updated to the last redirected URL and the method used to access the URL. Set the maximum number of redirects to zero to disable automatic redirection.

While it is set not to follow redirect automatically in the current session, it is possible to specify individual HTTP requests to follow redirect instructions the function FOLLOW\_REDIRECT and vice versa.

The default maximum number of redirections in a database user session is 3. The default value affects only future requests and has no effect on existing requests.

The SET\_FOLLOW\_REDIRECT procedure must be called before GET\_RESPONSE for any redirection to take effect.



## SET\_HEADER Procedure

This procedure sets an HTTP request header. The request header is sent to the Web server as soon as it is set.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.SET_HEADER (
    r          IN OUT NOCOPY req,
    name      IN VARCHAR2,
    value     IN VARCHAR2);
```

### Parameters

**Table 224–53 SET\_HEADER Procedure Parameters**

Parameter	Description
r	The HTTP request
name	The name of the HTTP request header
value	The value of the HTTP request header

### Usage Notes

Multiple HTTP headers with the same name are allowed in the HTTP protocol standard. Therefore, setting a header does not replace a prior header with the same name.

If the request is made using HTTP 1.1, UTL\_HTTP sets the Host header automatically for you.

When you set the Content-Type header with this procedure, UTL\_HTTP looks for the character set information in the header value. If the character set information is present, it is set as the character set of the request body. It can be overridden later by using the SET\_BODY\_CHARSET procedure.

When you set the Transfer-Encoding header with the value chunked, UTL\_HTTP automatically encodes the request body written by the WRITE\_TEXT, WRITE\_LINE and WRITE\_RAW procedures. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format.

## SET\_PERSISTENT\_CONN\_SUPPORT Procedure

This procedure enables or disables support for the HTTP 1.1 persistent-connection in the request.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(
    r          IN OUT NOCOPY req,
    enable    IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 224–54 SET\_PERSISTENT\_CONN\_SUPPORT Procedure Parameters**

Parameter	Description
r	The HTTP request
enable	TRUE to keep the network connection persistent. FALSE otherwise.

### Usage Notes

If the persistent-connection support is enabled for an HTTP request, the package will keep the network connections to a Web server or the proxy server open in the package after the request is completed properly for a subsequent request to the same server to reuse for each HTTP 1.1 protocol specification. With the persistent connection support, subsequent HTTP requests may be completed faster because the network connection latency is avoided. If the persistent-connection support is disabled for a request, the package will always send the HTTP header "Connection: close" automatically in the HTTP request and close the network connection when the request is completed. This setting has no effect on HTTP requests that follows HTTP 1.0 protocol, for which the network connections will always be closed after the requests are completed.

When a request is being made, the package attempts to reuse an existing persistent connection to the target Web server (or proxy server) if one is available. If none is available, a new network connection will be initiated. The persistent-connection support setting for a request affects only whether the network connection should be closed after a request completes.

Use this procedure to change the persistent-connection support setting a request inherits from the session default setting.

Users should note that while the use of persistent connections in UTL\_HTTP may reduce the time it takes to fetch multiple Web pages from the same server, it consumes precious system resources (network connections) in the database server. Also, excessive use of persistent connections may reduce the scalability of the database server when too many network connections are kept open in the database server. Network connections should be kept open only if they will be used immediately by subsequent requests and should be closed immediately when they are no longer needed. Set the default persistent connection support as disabled in the session, and enable persistent connection in individual HTTP requests as shown in ["Examples"](#) on page 224-89.

The default value of the maximum number of persistent connections in a database session is zero. To truly enable persistent connections, you must also set the maximum number of persistent connections to a positive value or no connections will be kept persistent.

## Examples

### Using SET\_PERSISTENT\_CONN\_SUPPORT in HTTP Requests

```

DECLARE
    TYPE vc2_table IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
    paths VC2_TABLE;

    UTL_HTTP.fetch_pages(paths IN vc2_table) AS
        url_prefix  VARCHAR2(256) := 'http://www.my-company.com/';
        req         UTL_HTTP.REQ;
        resp        UTL_HTTP.RESP;
        data        VARCHAR2(1024);
    BEGIN
        FOR i IN 1..paths.count LOOP
            req := UTL_HTTP.BEGIN_REQUEST(url_prefix || paths(i));
            -- Use persistent connection except for the last request
            IF (i < paths.count) THEN
                UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(req, TRUE);
            END IF;
            resp := UTL_HTTP.GET_RESPONSE(req);
            BEGIN
                LOOP
                    UTL_HTTP.READ_TEXT(resp, data);
                    -- do something with the data
                END LOOP;
            EXCEPTION
                WHEN UTL_HTTP.END_OF_BODY THEN
                    NULL;
            END;
            UTL_HTTP.END_RESPONSE(resp);
        END LOOP;
    END;

    BEGIN
        UTL_HTTP.SET_PERSISTENT_CONN_SUPPORT(FALSE, 1);
        paths(1) := '...';
        paths(2) := '...';
        ...
        fetch_pages(paths);
    END;

```

## SET\_PROXY Procedure

This procedure sets the proxy to be used for requests of the HTTP or other protocols, excluding those for hosts that belong to the domain specified in `no_proxy_domains`. `no_proxy_domains` is a comma-, semi-colon-, or space-separated list of domains or hosts for which HTTP requests should be sent directly to the destination HTTP server instead of going through a proxy server.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.SET_PROXY (
    proxy          IN VARCHAR2,
    no_proxy_domains IN VARCHAR2);
```

### Parameters

**Table 224–55 SET\_PROXY Procedure Parameters**

Parameter	Description
<code>proxy</code>	The proxy (host and an optional port number) to be used by the UTL_HTTP package
<code>no_proxy_domains</code>	The list of hosts and domains for which no proxy should be used for all requests

### Usage Notes

The proxy may include an optional TCP/IP port number at which the proxy server listens. The syntax is `[http://]host[:port][/]`, for example, `www-proxy.my-company.com:80`. If the port is not specified for the proxy, port 80 is assumed.

Optionally, a port number can be specified for each domain or host. If the port number is specified, the no-proxy restriction is only applied to the request at the port of the particular domain or host, for example, `corp.my-company.com`, `eng.my-company.com:80`. When `no_proxy_domains` is NULL and the proxy is set, all requests go through the proxy. When the proxy is not set, UTL\_HTTP sends requests to the target Web servers directly.

You can define a username/password for the proxy to be specified in the proxy string. The format is

```
[http://][user[:password]@]host[:port][/]
```

If proxy settings are set when the database server instance is started, the proxy settings in the environment variables `http_proxy` and `no_proxy` are assumed. Proxy settings set by this procedure override the initial settings.

## SET\_RESPONSE\_ERROR\_CHECK Procedure

This procedure sets whether or not `GET_RESPONSE` raises an exception when the Web server returns a status code that indicates an error—a status code in the 4xx or 5xx ranges. For example, when the requested URL is not found in the destination Web server, a 404 (document not found) response status code is returned.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.SET_RESPONSE_ERROR_CHECK (
    enable IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 224–56** *SET\_RESPONSE\_ERROR\_CHECK Procedure Parameters*

Parameter	Description
<code>enable</code>	TRUE to check for response errors; otherwise FALSE

### Usage Notes

If the status code indicates an error—a 4xx or 5xx code—and this procedure is enabled, `GET_RESPONSE` will raise the `HTTP_CLIENT_ERROR` or `HTTP_SERVER_ERROR` exception. If `SET_RESPONSE_ERROR_CHECK` is set to `FALSE`, `GET_RESPONSE` will not raise an exception when the status code indicates an error.

Response error check is turned off by default.

The `GET_RESPONSE` function can raise other exceptions when `SET_RESPONSE_ERROR_CHECK` is set to `FALSE`.

## SET\_TRANSFER\_TIMEOUT Procedure

This procedure sets the default time out value for all future HTTP requests that the UTL\_HTTP package should attempt while reading the HTTP response from the Web server or proxy server. This time out value may be used to avoid the PL/SQL programs from being blocked by busy Web servers or heavy network traffic while retrieving Web pages from the Web servers.

**See Also:** [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27

### Syntax

```
UTL_HTTP.SET_TRANSFER_TIMEOUT (  
    timeout IN PLS_INTEGER DEFAULT 60);
```

### Parameters

**Table 224–57 SET\_TRANSFER\_TIMEOUT Procedure Parameters**

Parameter	Description
timeout	The network transfer timeout value in seconds.

### Usage Notes

The default value of the time out is 60 seconds.

## SET\_WALLET Procedure

This procedure sets the Oracle wallet used for all HTTP requests over Secured Socket Layer (SSL), namely HTTPS. When the UTL\_HTTP package communicates with an HTTP server over SSL, the HTTP server presents its digital certificate, which is signed by a certificate authority, to the UTL\_HTTP package for identification purpose. The Oracle wallet contains the list of certificate authorities that are trusted by the user of the UTL\_HTTP package. An Oracle wallet is required to make an HTTPS request.

### See Also:

- [Session Settings](#) on page 224-17 and [Session Settings Subprograms](#) on page 224-27
- "Managing Fine-Grained Access in PL/SQL Network Utility Packages" in the *Oracle Database Security Guide*

## Syntax

```
UTL_HTTP.SET_WALLET (
    path          IN VARCHAR2,
    password      IN VARCHAR2 DEFAULT NULL);
```

## Parameters

**Table 224–58 SET\_WALLET Procedure Parameters**

Parameter	Description
path	The directory path that contains the Oracle wallet. The format is <code>file:directory-path</code> .  The format of <code>wallet_path</code> on a PC is, for example, <code>file:c:\WINNT\Profiles\username\WALLETS</code> , and in Unix is, for example, <code>file:/home/username/wallets</code> . When the UTL_HTTP package is executed in the Oracle database server, the wallet is accessed from the database server. Therefore, the wallet path must be accessible from the database server.
password	The password needed to open the wallet. If the wallet is auto-login enabled, the password may be omitted and should be set to NULL. See "Using Wallets with Automatic Login Enabled" in the <i>Oracle Database Advanced Security Administrator's Guide</i> for information about using Oracle Wallet Manager and the ORAPKI utility to create an auto-login wallet

## Usage Notes

To set up an Oracle wallet, use the Oracle Wallet Manager to create a wallet. In order for the HTTPS request to succeed, the certificate authority that signs the certificate of the remote HTTPS Web server must be a trust point set in the wallet.

When a wallet is created, it is populated with a set of well-known certificate authorities as trust points. If the certificate authority that signs the certificate of the remote HTTPS Web server is not among the trust points, or the certificate authority has new root certificates, you should obtain the root certificate of that certificate authority and install it as a trust point in the wallet using Oracle Wallet Manager

**See Also:** *Oracle Database Advanced Security Administrator's Guide*  
for more information on Wallet Manager



## WRITE\_LINE Procedure

This procedure writes a text line in the HTTP request body and ends the line with new-line characters (CRLF as defined in `UTL_TCP`). As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.WRITE_LINE(
  r      IN OUT NOCOPY req,
  data  IN VARCHAR2 CHARACTER SET ANY_CS);
```

### Parameters

**Table 224–59** *WRITE\_LINE Procedure Parameters*

Parameter	Description
<code>r</code>	The HTTP request
<code>data</code>	The text line to send in the HTTP request body

### Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the `Content-Length` header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. The `UTL_HTTP` package performs chunked transfer-encoding on the request body transparently when the `Transfer-Encoding: chunked` header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the `SET_HEADER` procedure for details.

If you send the `Content-Length` header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the `Content-Length` header, and the results using the `WRITE_RAW` procedure to avoid the automatic character set conversion. Or, if the remote Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where `UTL_HTTP` handles the length of the chunks transparently.

## WRITE\_RAW Procedure

This procedure writes some binary data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.WRITE_RAW(
  r      IN OUT NOCOPY REQ,
  data  IN          RAW);
```

### Parameters

**Table 224–60** WRITE\_RAW Procedure Parameters

Parameter	Description
r	The HTTP request
data	The binary data to send in the HTTP request body

### Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the `Content-Length` header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. `UTL_HTTP` performs chunked transfer-encoding on the request body transparently when the `Transfer-Encoding: chunked` header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the `SET_HEADER` procedure for details.

## WRITE\_TEXT Procedure

This procedure writes some text data in the HTTP request body. As soon as some data is sent as the HTTP request body, the HTTP request headers section is completed. Text data is automatically converted from the database character set to the request body character set.

**See Also:** [HTTP Requests](#) on page 224-17 and [HTTP Requests Subprograms](#) on page 224-28

### Syntax

```
UTL_HTTP.WRITE_TEXT(
  r      IN OUT NOCOPY REQ,
  data  IN          VARCHAR2 CHARACTER SET ANY_CS);
```

### Parameters

**Table 224–61** *WRITE\_TEXT Procedure Parameters*

Parameter	Description
r	The HTTP request
data	The text data to send in the HTTP request body

### Usage Notes

An HTTP client must always let the remote Web server know the length of the request body it is sending. If the amount of data is known beforehand, you can set the `Content-Length` header in the request, where the length of the content is measured in bytes instead of characters. If the length of the request body is not known beforehand, you can send the request body using the HTTP 1.1 chunked transfer-encoding format. The request body is sent in chunks, where the length of each chunk is sent before the chunk is sent. `UTL_HTTP` performs chunked transfer-encoding on the request body transparently when the `Transfer-Encoding: chunked` header is set. Note that some HTTP-1.1-based Web servers or CGI programs do not support or accept the request body encoding in the HTTP 1.1 chunked transfer-encoding format. See the `SET_HEADER` procedure for details.

If you send the `Content-Length` header, you should note that the length specified in the header should be the byte-length of the textual request body after it is converted from the database character set to the request body character set. When either one of the two character sets is a multibyte character set, the precise byte-length of the request body in the request body character set cannot be known beforehand. In this case, you can perform the character set conversion explicitly, determine the byte-length of the results, send the `Content-Length` header, and the results using the `WRITE_RAW` procedure to avoid the automatic character set conversion. Or, if the remote Web server or CGI programs allow, you can send the request body using the HTTP 1.1 chunked transfer-encoding format, where `UTL_HTTP` handles the length of the chunks transparently.



UTL\_I18N is a set of services that provides additional globalization functionality for applications written in PL/SQL.

**See Also:** *Oracle Database Globalization Support Guide*

The chapter contains the following topics:

- [Using UTL\\_I18N](#)
  - Overview
  - Security Model
  - Constants
- [Summary of UTL\\_I18N Subprograms](#)

## Using UTL\_I18N

This section contains topics which relate to using the UTL\_I18N package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)

## Overview

The UTL\_I18N PL/SQL package consists of the following categories of services:

- String conversion functions for various datatypes.
- Functions that convert a text string to character references and vice versa.
- Functions that map between Oracle, Java, and ISO languages and territories.
- Functions that map between Oracle, Internet Assigned Numbers Authority (IANA), and e-mail safe character sets.
- A function that returns the Oracle character set name from an Oracle language name.
- A function that performs script transliteration.
- Functions that return the ISO currency code, local time zones, and local languages supported for a given territory.
- Functions that return the most appropriate linguistic sort, a listing of all the applicable linguistic sorts, and the local territories supported for a given language.
- Functions that map between the Oracle full and short language names.
- A function that returns the language translation of a given language and territory name.
- A function that returns a listing of the most commonly used time zones.

## Security Model

The functions of the `UTL_I18N` package neither read database contents nor modify them. The functions operate on their arguments only and/or they retrieve static internationalization information from NLS Data files. The execution privilege for the package is granted to `PUBLIC` by default



## Constants

The UTL\_I18N package uses the constants shown in [Table 225–1](#).

**Table 225–1 UTL\_I18N Constants**

Constant	Type	Value	Description
GENERIC_CONTEXT	PLS_INTEGER	0	Returns the default character set for general cases.
MAIL_GENERIC	PLS_INTEGER	0	Map from an Oracle character set name to an email safe character set name on a non-Windows platform.
ORACLE_TO_IANA	PLS_INTEGER	0	Map from an Oracle character set name to an IANA character set name.
SHIFT_IN	PLS_INTEGER	0	Used with <code>shift_status</code> . Must be set the first time it is called in piecewise conversion.
IANA_TO_ORACLE	PLS_INTEGER	1	Map from an IANA character set name to an Oracle character set name.
MAIL_CONTEXT	PLS_INTEGER	1	The mapping is between an Oracle character set name and an email safe character set name.
MAIL_WINDOWS	PLS_INTEGER	1	Map from an Oracle character set name to an email safe character set name on a Windows platform.
SHIFT_OUT	PLS_INTEGER	1	
FWKATAKANA_HIRAGANA	VARCHAR2 (30)	'fwkatakana_hiragana'	Converts only fullwidth Katakana characters to fullwidth Hiragana characters.
FWKATAKANA_HWKATAKANA	VARCHAR2 (30)	'fwkatakana_hwkatakana'	Converts only fullwidth Katakana characters to halfwidth Katakana characters.
HIRAGANA_FWKATAKANA	VARCHAR2 (30)	'hiragana_fwkatakana'	Converts only fullwidth Hiragana characters to fullwidth Katakana characters.
HIRAGANA_HWKATAKANA	VARCHAR2 (30)	'hiragana_hwkatakana'	Converts only fullwidth Hiragana characters to halfwidth Katakana characters.
HWKATAKANA_FWKATAKANA	VARCHAR2 (30)	'hwkatakana_fwkatakana'	Converts only halfwidth Katakana characters to fullwidth Katakana characters.
HWKATAKANA_HIRAGANA	VARCHAR2 (30)	'hwkatakana_hiragana'	Converts only halfwidth Katakana characters to fullwidth Hiragana characters.
KANA_FWKATAKANA	VARCHAR2 (30)	'kana_fwkatakana'	Converts any type of Kana character to a fullwidth Katakana character.
KANA_HIRAGANA	VARCHAR2 (30)	'kana_hiragana'	Converts any type of Kana character to a fullwidth Hiragana character.

**Table 225–1 (Cont.) UTL\_I18N Constants**

<b>Constant</b>	<b>Type</b>	<b>Value</b>	<b>Description</b>
KANA_ HWKATAKANA	VARCHAR2 (30)	'kana_ hwkatakana '	Converts any type of Kana character to a halfwidth Katakana character.

## Summary of UTL\_I18N Subprograms

**Table 225–2 UTL\_I18N Package Subprograms**

Procedure	Description
<a href="#">ESCAPE_REFERENCE Function</a> on page 225-9	Converts a given text string to its character reference counterparts, for characters that fall outside the document character set.
<a href="#">GET_COMMON_TIME_ZONES Function</a> on page 225-10	Returns the list of common time zone IDs that are independent of the locales.
<a href="#">GET_DEFAULT_CHARSET Function</a> on page 225-11	Returns the default Oracle character set name or the default e-mail safe character set name from an Oracle language name.
<a href="#">GET_DEFAULT_ISO_CURRENCY Function</a> on page 225-12	Returns the default ISO 4217 currency code for the specified territory.
<a href="#">GET_DEFAULT_LINGUISTIC_SORT Function</a> on page 225-13	Returns the default linguistic sort name for the specified language.
<a href="#">GET_LOCAL_LANGUAGES Function</a> on page 225-14	Returns the local language names for the specified territory.
<a href="#">GET_LOCAL_LINGUISTIC_SORTS Function</a> on page 225-15	Returns the local linguistic sort names for the specified language.
<a href="#">GET_LOCAL_TERRITORIES Function</a> on page 225-16	Returns the local territory names for the specified language.
<a href="#">GET_LOCAL_TIME_ZONES Function</a> on page 225-17	Returns the local time zone IDs for the specified territory.
<a href="#">GET_TRANSLATION Function</a> on page 225-19	Returns the translation of the language and territory name in the specified translation language.
<a href="#">MAP_CHARSET Function</a> on page 225-20	<ul style="list-style-type: none"> <li>■ Maps an Oracle character set name to an IANA character set name.</li> <li>■ Maps an IANA character set name to an Oracle character set name.</li> <li>■ Maps an Oracle character set name to an e-mail safe character set name.</li> </ul>
<a href="#">MAP_FROM_SHORT_LANGUAGE Function</a> on page 225-22	Maps an Oracle short language name to an Oracle language name.
<a href="#">MAP_LANGUAGE_FROM_ISO Function</a> on page 225-23	Returns an Oracle language name from an ISO locale name.
<a href="#">MAP_LOCALE_TO_ISO Function</a> on page 225-24	Returns an ISO locale name from the Oracle language and territory name.
<a href="#">MAP_TERRITORY_FROM_ISO Function</a> on page 225-25	Returns an Oracle territory name from an ISO locale name.
<a href="#">MAP_TO_SHORT_LANGUAGE Function</a> on page 225-26	Maps an Oracle language name to an Oracle short language name.
<a href="#">RAW_TO_CHAR Functions</a> on page 225-27	Converts RAW data that is not encoded in the database character set into a VARCHAR2 string
<a href="#">RAW_TO_NCHAR Functions</a> on page 225-29	Converts RAW data that is not encoded in the national character set into an NVARCHAR2 string
<a href="#">STRING_TO_RAW Function</a> on page 225-31	Converts a VARCHAR2 or NVARCHAR2 string to another character set. The result is returned as a RAW datatype.

**Table 225–2 (Cont.) UTL\_I18N Package Subprograms**

<b>Procedure</b>	<b>Description</b>
<a href="#">TRANSLITERATE Function</a> on page 225-32	Transliterates between Japanese hiragana and katakana.
<a href="#">UNESCAPE_REFERENCE Function</a> on page 225-34	Converts an input string that contains character references to a text string.

## ESCAPE\_REFERENCE Function

This function converts a text string to its character reference counterparts for characters that fall outside the character set used by the current document. Character references are mainly used in HTML and XML documents to represent characters independently of the encoding of the document.

Character references may appear in two forms, numeric character references and character entity references. Numeric character references specify the Unicode code point value of a character, while character entity references use symbolic names to refer to the same character. For example, `&#xe5;` is the numeric character reference for the small letter "a" with a ring above, whereas `&aring;` is the character entity reference for the same character. Character entity references are also used to escape special characters, as an example, `&lt;` represents the < (less than) sign. This is to avoid possible confusion with the beginning of a tag in Markup languages.

### Syntax

```
UTL_I18N.ESCAPE_REFERENCE(
    str          IN VARCHAR2 CHARACTER SET ANY_CS,
    page_cs_name IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2 CHARACTER SET str%CHARSET;
```

### Parameters

**Table 225-3 ESCAPE\_REFERENCE Function Parameters**

Parameter	Description
<code>str</code>	Specifies the input string
<code>page_cs_name</code>	Specifies the character set of the document. If <code>page_cs_name</code> is NULL, then the database character set is used for CHAR data and the national character set is used for NCHAR data.

### Usage Notes

If the user specifies an invalid character set or a NULL string, then the function returns a NULL string.

### Examples

```
UTL_I18N.ESCAPE_REFERENCE('hello < '||chr(229),'us7ascii')
```

This returns 'hello &lt; &#xe5;'.

## GET\_COMMON\_TIME\_ZONES Function

This function returns a listing of the most commonly used time zones. This list contains a subset of the time zones that are supported in the database.

### Syntax

```
UTL_I18N.GET_COMMON_TIME_ZONES  
RETURN STRING_ARRAY;
```

### Examples

Returns the list of the most commonly used time zones.

```
DECLARE  
    retval UTL_I18N.STRING_ARRAY;  
BEGIN  
    retval := UTL_I18N.GET_COMMON_TIME_ZONES;  
END;  
/
```

## GET\_DEFAULT\_CHARSET Function

This function returns the default Oracle character set name or the default e-mail safe character set name from an Oracle language name.

**See Also:** "[MAP\\_CHARSET Function](#)" on page 225-20 for an explanation of an e-mail safe character set

### Syntax

```
UTL_I18N.GET_DEFAULT_CHARSET(
    language IN VARCHAR2,
    context  IN PLS_INTEGER DEFAULT GENERIC_CONTEXT,
    iswindows IN BOOLEAN DEFAULT FALSE)
RETURN VARCHAR2;
```

### Parameters

**Table 225-4** GET\_DEFAULT\_CHARSET Function Parameters

Parameter	Description
language	Specifies a valid Oracle language
context	GENERIC_CONTEXT   MAIL_CONTEXT GENERIC_CONTEXT: Returns the default character set for general cases MAIL_CONTEXT: Returns the default e-mail safe character set name
iswindows	If context is set as MAIL_CONTEXT, then iswindows should be set to TRUE if the platform is Windows and FALSE if the platform is not Windows. The default is FALSE.  iswindows has no effect if context is set as GENERIC_CONTEXT.

### Usage Notes

If the user specifies an invalid language name or an invalid flag, then the function returns a NULL string.

### Examples

#### GENERIC\_CONTEXT, iswindows=FALSE

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.GENERIC_CONTEXT, FALSE)
```

This returns 'WE8ISO8859P1'.

#### MAIL\_CONTEXT, iswindows=TRUE

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.MAIL_CONTEXT, TRUE)
```

This returns 'WE8MSWIN1252'.

#### MAIL\_CONTEXT, iswindows=FALSE

```
UTL_I18N.GET_DEFAULT_CHARSET('French', UTL_I18N.MAIL_CONTEXT, FALSE)
```

This returns 'WE8ISO8859P1'.

## GET\_DEFAULT\_ISO\_CURRENCY Function

This function returns the default ISO 4217 currency code for the specified territory.

### Syntax

```
UTL_I18N.GET_DEFAULT_ISO_CURRENCY (  
    territory    IN VARCHAR2 CHARACTER SET ANY_CS)  
RETURN VARCHAR2;
```

### Parameters

**Table 225–5** GET\_DEFAULT\_ISO\_CURRENCY Function Parameters

Parameter	Description
territory	Specifies a valid Oracle territory. It is case-insensitive.

### Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

### Examples

Displays the default ISO currency code for China.

```
DECLARE  
    retval VARCHAR2(50);  
BEGIN  
    retval := UTL_I18N.GET_DEFAULT_ISO_CURRENCY('CHINA');  
    DBMS_OUTPUT.PUT_LINE(retval);  
END;  
/
```



## GET\_DEFAULT\_LINGUISTIC\_SORT Function

This function returns the most commonly used Oracle linguistic sort for the specified language.

### Syntax

```
UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT (
    language IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

### Parameters

**Table 225–6** *GET\_DEFAULT\_LINGUISTIC\_SORT Function Parameters*

Parameter	Description
language	Specifies a valid Oracle language. It is case-insensitive.

### Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

### Examples

Displays the name of the most appropriate linguistic sort name for the language used in the current SQL session.

```
DECLARE
    retval VARCHAR2(50);
BEGIN
    SELECT value INTO retval FROM nls_session_parameters
    WHERE parameter = 'NLS_LANGUAGE';
    retval := UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT(retval);
    DBMS_OUTPUT.PUT_LINE(retval);
END;
/
```

## GET\_LOCAL\_LANGUAGES Function

This function returns the local language names for the specified territory.

### Syntax

```
UTL_I18N.GET_LOCAL_LANGUAGES (
    territory    IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN STRING_ARRAY;
```

### Parameters

**Table 225–7 GET\_LOCAL\_LANGUAGES Function Parameters**

Parameter	Description
territory	Specifies a valid Oracle territory. It is case-insensitive.

### Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

### Examples

Returns the list of local languages used in Belgium.

```
DECLARE
    retval UTL_I18N.STRING_ARRAY;
    cnt    INTEGER;
BEGIN
    retval := UTL_I18N.GET_LOCAL_LANGUAGES('BELGIUM');
    DBMS_OUTPUT.PUT('Count = ');
    DBMS_OUTPUT.PUT_LINE(retval.LAST);
    cnt := retval.FIRST;
    WHILE cnt IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(retval(cnt));
        cnt := retval.NEXT(cnt);
    END LOOP;
END;
/
...
Count = 2
DUTCH
FRENCH
```

## GET\_LOCAL\_LINGUISTIC\_SORTS Function

This function returns a list of the Oracle linguistic sort names that are appropriate for the specified language. A BINARY sort is included for all languages.

### Syntax

```
UTL_I18N.GET_LOCAL_LINGUISTIC_SORTS (
    language IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN STRING_ARRAY;
```

### Parameters

**Table 225–8 GET\_LOCAL\_LINGUISTIC\_SORTS Function Parameters**

Parameter	Description
language	Specifies a valid Oracle language. It is case-insensitive.

### Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

### Examples

Displays the local linguistic sort names for JAPANESE.

```
DECLARE
    retval UTL_I18N.STRING_ARRAY;
    cnt INTEGER;
BEGIN
    retval := UTL_I18N.GET_LOCAL_LINGUISTIC_SORTS('Japanese');
    DBMS_OUTPUT.PUT('Count = ');
    DBMS_OUTPUT.PUT_LINE(retval.COUNT);
    cnt := retval.FIRST;
    WHILE cnt IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(retval(cnt));
        cnt := retval.NEXT(cnt);
    END LOOP;
END;
/

...
Count = 2
JAPANESE_M
BINARY
```

## GET\_LOCAL\_TERRITORIES Function

This function returns the local territory names for the specified language.

### Syntax

```
UTL_I18N.GET_LOCAL_TERRITORIES (
    language IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN STRING_ARRAY;
```

### Parameters

**Table 225–9 GET\_LOCAL\_TERRITORIES Function Parameters**

Parameter	Description
language	Specifies a valid Oracle language. It is case-insensitive.

### Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

### Examples

Returns the list of Oracle territories that use German as one of their local languages.

```
DECLARE
    retval UTL_I18N.STRING_ARRAY;
    cnt    INTEGER;
BEGIN
    retval := UTL_I18N.GET_LOCAL_TERRITORIES('GERMAN');
    DBMS_OUTPUT.PUT('Count = ');
    DBMS_OUTPUT.PUT_LINE(retval.LAST);
    cnt := retval.FIRST;
    WHILE cnt IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(retval(cnt));
        cnt := retval.NEXT(cnt);
    END LOOP;
END;
/
...
Count = 4
GERMANY
AUSTRIA
LUXEMBOURG
SWITZERLAND
```

## GET\_LOCAL\_TIME\_ZONES Function

This function returns the local time zone IDs for the specified territory.

### Syntax

```
UTL_I18N.GET_LOCAL_TIME_ZONES (
    territory      IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL)
RETURN STRING_ARRAY;
```

### Parameters

**Table 225–10 GET\_LOCAL\_TIME\_ZONES Function Parameters**

Parameter	Description
territory	Specifies a valid Oracle territory. It is case-insensitive.

### Usage Notes

If the user specifies an invalid territory name, then the function returns a NULL string.

### Examples

Creates a function that returns the list of time zones locally used in the territory AZERBAIJAN followed by the general common time zones. This is useful for when the user's territory is known and the application still allows the user to choose other time zones as a user's preference.

```
CREATE OR REPLACE FUNCTION get_time_zones
(territory IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN utl_i18n.string_array
IS
    retval  utl_i18n.string_array;
    retval2 utl_i18n.string_array;
    stpos   INTEGER;
BEGIN
    retval := utl_i18n.get_local_time_zones(
        territory);
    retval2 := utl_i18n.get_common_time_zones;
    stpos := retval.LAST + 1;
    retval(stpos) := '-----'; -- a separator
    FOR i IN retval2.FIRST..retval2.LAST LOOP
        stpos := stpos + 1;
        retval(stpos) := retval2(i);
    END LOOP;
    RETURN retval;
END;
```

Returns the list of local time zones for AZERBAIJAN followed by the common time zones with a separator string of five dashes (-----).

```
DECLARE
    retval UTL_I18N.STRING_ARRAY;
    cnt   INTEGER;
BEGIN
    DBMS_OUTPUT.ENABLE(100000);
    retval UTL_I18N.GET_TIME_ZONES('AZERBAIJAN');
    cnt := retval.FIRST;
```

```
    WHILE cnt IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE(retval(cnt));
        cnt := retval.NEXT(cnt);
    END LOOP;
END;
/
```

Asia/Baku

-----

```
Pacific/Pago_Pago
Pacific/Honolulu
America/Anchorage
America/Vancouver
America/Los_Angeles
America/Tijuana
America/Edmonton
America/Denver
America/Phoenix
America/Mazatlan
America/Winnipeg
America/Regina
America/Chicago
America/Mexico_City
America/Guatemala
America/El_Salvador
America/Managua
America/Costa_Rica
America/Montreal
...
```

## GET\_TRANSLATION Function

This function returns the translation of the language and territory name in the specified translation language.

### Syntax

```
UTL_I18N.GET_TRANSLATION (
    parameter          IN VARCHAR2 CHARACTER SET ANY_CS,
    trans_language     IN VARCHAR2 'AMERICAN',
    flag               IN PLS_INTEGER DEFAULT LANGUAGE_TRANS)
RETURN VARCHAR2 CHARACTER SET parameter%CHARSET;
```

### Parameters

**Table 225–11** GET\_TRANSLATION Function Parameters

Parameter	Description
parameter	Specifies a valid language name, territory name, or a combined string in the form of <i>language_territory</i> . It is case-insensitive.
trans_language	Specifies a translation language name. For example, ITALIAN is for the Italian language. The default is AMERICAN, which indicates American English.
flag	Specifies the translation type: <ul style="list-style-type: none"> <li>▪ LANGUAGE_TRANS: The function returns the language translation.</li> <li>▪ TERRITORY_TRANS: The function returns the territory translation.</li> <li>▪ LANGUAGE_TERRITORY_TRANS: The function returns the language and territory translation.</li> </ul> <p>The default translation type is LANGUAGE_TRANS.</p>

### Usage Notes

If VARCHAR2 is used as a parameter type, the returned translation text can be corrupted due to the conversion to the database character set. Using NVARCHAR2 as the parameter type will preserve the translation text because Unicode can encode all translated languages.

If the specified translation language is not available or an invalid name is provided, the default "American English" translations are returned. For example, Oracle does not provide GUJARATI translations, so the returned translation would be in American English.

### Examples

The following returns the names of all the Oracle-supported languages in Italian.

```
DECLARE
    CURSOR c1 IS
        SELECT value FROM V$NLS_VALID_VALUES
        WHERE parameter = 'LANGUAGE'
        ORDER BY value;
    retval NVARCHAR2(100);
BEGIN
    FOR item IN c1 LOOP
        retval := UTL_I18N.GET_TRANSLATION (TO_NCHAR(item.value), 'italian');
    END LOOP;
END;
```

## MAP\_CHARSET Function

This function maps the following:

- An Oracle character set name to an IANA character set name.
- An IANA character set name to an Oracle character set name.
- An Oracle character set to an e-mail safe character set name.

### Syntax

```
UTL_I18N.MAP_CHARSET (
    charset    IN VARCHAR2,
    context    IN PLS_INTEGER DEFAULT GENERIC_CONTEXT,
    flag       IN PLS_INTEGER DEFAULT ORACLE_TO_IANA)
RETURN VARCHAR2;
```

### Parameters

**Table 225–12 MAP\_CHARSET Function Parameters**

Parameter	Description
charset	Specifies the character set name to be mapped. The mapping is case-insensitive.
context	GENERIC_CONTEXT   MAIL_CONTEXT GENERIC_CONTEXT: The mapping is between an Oracle character set name and an IANA character set name. This is the default value. MAIL_CONTEXT: The mapping is between an Oracle character set name and an email safe character set name.
flag	<ul style="list-style-type: none"> <li>■ ORACLE_TO_IANA   IANA_TO_ORACLE if GENERIC_CONTEXT is set                ORACLE_TO_IANA: Map from an Oracle character set name to an IANA character set name. This is the default.                IANA_TO_ORACLE: Map from an IANA character set name to an Oracle character set name.</li> <li>■ MAIL_GENERIC   MAIL_WINDOWS if MAIL_CONTEXT is set                MAIL_GENERIC: Map from an Oracle character set name to an email safe character set name on a non-Windows platform.                MAIL_WINDOWS: Map from an Oracle character set name to an email safe character set name on a Windows platform.</li> </ul>

### Usage Notes

An e-mail safe character set is an Oracle character set that is commonly used by applications when they submit e-mail messages. The character set is usually used to convert contents in the database character set to e-mail safe contents. To specify the character set name in the mail header, you should use the corresponding IANA character set name obtained by calling the MAP\_CHARSET function with the ORACLE\_TO\_IANA option, providing the e-mail safe character set name as input.

For example, no e-mail client recognizes message contents in the WE8DEC character set, whose corresponding IANA name is DEC-MCS. If WE8DEC is passed to the MAP\_CHARSET function with the MAIL\_CONTEXT option, then the function returns WE8ISO8859P1. Its corresponding IANA name, ISO-8859-1, is recognized by most e-mail clients.



The steps in this example are as follows:

1. Call the `MAP_CHARSET` function with the `MAIL_CONTEXT | MAIL_GENERIC` option with the database character set name, `WE8DEC`. The result is `WE8ISO8859P1`.
2. Convert the contents stored in the database to `WE8ISO8859P1`.
3. Call the `MAP_CHARSET` function with the `ORACLE_TO_IANA | GENERIC_CONTEXT` option with the e-mail safe character set, `WE8ISO8859P1`. The result is `ISO-8859-1`.
4. Specify `ISO-8859-1` in the mail header when the e-mail message is submitted.

The function returns a character set name if a match is found. If no match is found or if the flag is invalid, then it returns `NULL`.

---



---

**Note:** Many Oracle character sets can map to one e-mail safe character set. There is no function that maps an e-mail safe character set to an Oracle character set name.

---



---

## Examples

### Generic Context

```
UTL_I18N.MAP_CHARSET('iso-8859-1',UTL_I18N.GENERIC_CONTEXT,UTL_I18N.IANA_TO_
ORACLE)
```

This returns 'WE8ISO8859P1'.

### Context

```
UTL_I18N.MAP_CHARSET('WE8DEC', utl_i18n.mail_context, utl_i18n.mail_generic)
```

This returns 'WE8ISO8859P1'.

**See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle character sets

## MAP\_FROM\_SHORT\_LANGUAGE Function

This function maps an Oracle short language name to an Oracle language name.

### Syntax

```
UTL_I18N.MAP_FROM_SHORT_LANGUAGE (
    language          IN VARCHAR2 CHARACTER SET ANY_CS)
RETURN VARCHAR2;
```

### Parameters

**Table 225–13 MAP\_FROM\_SHORT\_LANGUAGE Function Parameters**

Parameter	Description
language	Specifies a valid short language name. It is case-insensitive.

### Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

### Examples

Returns the default linguistic sort name for the customer with the ID of 9000. Note that the table `customers` is from the `oe` user in the Common Schema. Because the customer's language preference is stored using a short language name, you need to convert to a full language name by calling the `GET_DEFAULT_LINGUISTIC_SORT` procedure.

```
DECLARE
    short_n VARCHAR2(10);
    ling_n VARCHAR2(50);
BEGIN
    SELECT nls_language INTO short
    FROM customers WHERE customer_id = 9000;
    ling_n := UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT (
        UTL_I18N.MAP_FROM_SHORT_LANGUAGE(short_n));
    DBMS_OUTPUT.PUT_LINE(ling_n);
END;
/
```

## MAP\_LANGUAGE\_FROM\_ISO Function

This function returns an Oracle language name from an ISO locale name.

### Syntax

```
UTL_I18N.MAP_LANGUAGE_FROM_ISO(  
    isolocale IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 225–14** MAP\_LANGUAGE\_FROM\_ISO Function Parameters

Parameter	Description
isolocale	Specifies the ISO locale. The mapping is case-insensitive.

### Usage Notes

If the user specifies an invalid locale string, then the function returns a NULL string.

If the user specifies a locale string that includes only the language (for example, en\_ instead of en\_US), then the function returns the default language name for the specified language (for example, American).

### Examples

```
UTL_I18N.MAP_LANGUAGE_FROM_ISO('en_US')
```

This returns 'American'.

**See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle languages

## MAP\_LOCALE\_TO\_ISO Function

This function returns an ISO locale name from an Oracle language name and an Oracle territory name. A valid string must include at least one of the following: a valid Oracle language name or a valid Oracle territory name.

### Syntax

```
UTL_I18N.MAP_LOCALE_TO_ISO (  
    ora_language  IN VARCHAR2,  
    ora_territory IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 225–15** MAP\_LOCALE\_TO\_ISO Function Parameters

Parameter	Description
ora_language	Specifies an Oracle language name. It is case-insensitive.
ora_territory	Specifies an Oracle territory name. It is case-insensitive.

### Usage Notes

If the user specifies an invalid string, then the function returns a NULL string.

### Examples

```
UTL_I18N.MAP_LOCALE_TO_ISO('American','America')
```

This returns 'en\_US'.

**See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle languages and territories

## MAP\_TERRITORY\_FROM\_ISO Function

This function returns an Oracle territory name from an ISO locale.

### Syntax

```
UTL_I18N.MAP_TERRITORY_FROM_ISO (
    isolocale IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 225–16** MAP\_TERRITORY\_FROM\_ISO Function Parameters

Parameter	Description
isolocale	Specifies the ISO locale. The mapping is case-insensitive.

### Usage Notes

If the user specifies an invalid locale string, then the function returns a NULL string.

If the user specifies a locale string that includes only the territory (for example, `_fr` instead of `fr_fr`), then the function returns the default territory name for the specified territory (for example, `France`).

### Examples

```
UTL_I18N.MAP_TERRITORY_FROM_ISO('en-US')
```

This returns 'America'.

**See Also:** *Oracle Database Globalization Support Guide* for a list of valid Oracle territories

## MAP\_TO\_SHORT\_LANGUAGE Function

This function maps an Oracle language name to an Oracle short language name.

### Syntax

```
UTL_I18N.MAP_TO_SHORT_LANGUAGE (  
    language    IN VARCHAR2 CHARACTER SET ANY_CS)  
RETURN VARCHAR2;
```

### Parameters

**Table 225–17** MAP\_TO\_SHORT\_LANGUAGE Function Parameters

Parameter	Description
language	Specifies a valid full language name. It is case-insensitive.

### Usage Notes

If the user specifies an invalid language name, then the function returns a NULL string.

### Examples

Returns the short language name for the language.

```
DECLARE  
    retval VARCHAR2(100);  
BEGIN  
    retval := UTL_I18N.MAP_TO_SHORT_LANGUAGE('american');  
    DBMS_OUTPUT.PUT_LINE(retval);  
END;  
/  
  
US
```

## RAW\_TO\_CHAR Functions

This function converts RAW data from a valid Oracle character set to a VARCHAR2 string in the database character set.

The function is overloaded. The different forms of functionality are described along with the syntax declarations.

### Syntax

Buffer Conversion:

```
UTL_I18N.RAW_TO_CHAR(
    data          IN RAW,
    src_charset   IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Piecewise conversion converts raw data into character data piece by piece:

```
UTL_I18N.RAW_TO_CHAR (
    data          IN RAW,
    src_charset   IN VARCHAR2 DEFAULT NULL,
    scanned_length OUT PLS_INTEGER,
    shift_status  IN OUT PLS_INTEGER)
RETURN VARCHAR2;
```

### Parameters

**Table 225–18 RAW\_TO\_CHAR Function Parameters**

Parameter	Description
data	Specifies the RAW data to be converted to a VARCHAR2 string
src_charset	Specifies the character set that the RAW data was derived from. If src_charset is NULL, then the database character set is used.
scanned_length	Specifies the number of bytes of source data scanned
shift_status	Specifies the shift status at the end of the scan. The user must set it to SHIFT_IN the first time it is called in piecewise conversion.  <b>Note:</b> ISO 2022 character sets use escape sequences instead of shift characters to indicate the encoding method. shift_status cannot hold the encoding method information that is provided by the escape sequences for the next function call. As a result, this function cannot be used to reconstruct ISO 2022 character from raw data in a piecewise way unless each unit of input can be guaranteed to be a closed string. A closed string begins and ends in a 7-bit escape state.

### Usage Notes

If the user specifies an invalid character set, NULL data, or data whose length is 0, then the function returns a NULL string.

### Examples

#### Buffer Conversion

```
UTL_I18N.RAW_TO_CHAR(hextoraw('6162636465666C2AA'), 'utf8')
```

This returns the following string in the database character set:

```
'abcde' || chr(170)
```

### Piecewise Conversion

```
UTL_I18N.RAW_TO_CHAR(hextoraw('6162636465666C2AA'),'utf8',shf,slen)
```

This expression returns the following string in the database character set:

```
'abcde' || chr(170)
```

It also sets shf to SHIFT\_IN and slen to 8.

The following example converts data from the Internet piece by piece to the database character set.

```
rvalue RAW(1050);
nvalue VARCHAR2(1024);
conversion_state PLS_INTEGER = 0;
converted_len PLS_INTEGER;
rtemp RAW(10) = '';
conn utl_tcp.connection;
tlen PLS_INTEGER;

...
conn := utl_tcp.open_connection ( remote_host => 'localhost',
                                remote_port => 2000);

LOOP
    tlen := utl_tcp.read_raw(conn, rvalue, 1024);
    rvalue := utl_raw.concat(rtemp, rvalue);
    nvalue := utl_i18n.raw_to_char(rvalue, 'JA16SJIS', converted_len,
conversion_stat);
    if (converted_len < utl_raw.length(rvalue) )
    then
        rtemp := utl_raw.substr(rvalue, converted_len+1);
    else
        rtemp := '';
    end if;
    /* do anything you want with nvalue */
    /* e.g htp.prn(nvalue); */
END LOOP;
utl_tcp.close_connection(conn);
EXCEPTION
    WHEN utl_tcp.end_of_input THEN
        utl_tcp.close_connection(conn);
END;
```



## RAW\_TO\_NCHAR Functions

This function converts RAW data from a valid Oracle character set to an NVARCHAR2 string in the national character set.

The function is overloaded. The different forms of functionality are described along with the syntax declarations.

### Syntax

Buffer Conversion:

```
UTL_I18N.RAW_TO_NCHAR (
    data          IN RAW,
    src_charset   IN VARCHAR2 DEFAULT NULL)
RETURN NVARCHAR2;
```

Piecewise conversion converts raw data into character data piece by piece:

```
UTL_I18N.RAW_TO_NCHAR (
    data          IN RAW,
    src_charset   IN VARCHAR2 DEFAULT NULL,
    scanned_length OUT PLS_INTEGER,
    shift_status  IN OUT PLS_INTEGER)
RETURN NVARCHAR2;
```

### Parameters

**Table 225–19 RAW\_TO\_NCHAR Function Parameters**

Parameter	Description
data	Specifies the RAW data to be converted to an NVARCHAR2 string
src_charset	Specifies the character set that the RAW data was derived from. If src_charset is NULL, then the database character set is used.
scanned_length	Specifies the number of bytes of source data scanned
shift_status	Specifies the shift status at the end of the scan. The user must set it to SHIFT_IN the first time it is called in piecewise conversion.  <b>Note:</b> ISO 2022 character sets use escape sequences instead of shift characters to indicate the encoding method. shift_status cannot hold the encoding method information that is provided by the escape sequences for the next function call. As a result, this function cannot be used to reconstruct ISO 2022 character from raw data in a piecewise way unless each unit of input can be guaranteed to be a closed string. A closed string begins and ends in a 7-bit escape state.

### Usage Notes

If the user specifies an invalid character set, NULL data, or data whose length is 0, then the function returns a NULL string.

### Examples

#### Buffer Conversion

```
UTL_I18N.RAW_TO_NCHAR(hextoraw('6162636465666C2AA'), 'utf8')
```

This returns the following string in the national character set:

```
'abcde' || chr(170)
```

### Piecewise Conversion

```
UTL_I18N.RAW_TO_NCHAR(hextoraw('6162636465666C2AA'),'utf8', shf, slen)
```

This expression returns the following string in the national character set:

```
'abcde' || chr(170)
```

It also sets `shf` to `SHIFT_IN` and `slen` to 8.

The following example converts data from the Internet piece by piece to the national character set.

```
rvalue RAW(1050);
nvalue NVARCHAR2(1024);
conversion_state PLS_INTEGER = 0;
converted_len PLS_INTEGER;
rtemp RAW(10) = '';
conn utl_tcp.connection;
tlen PLS_INTEGER;

...
conn := utl_tcp.open_connection ( remote_host => 'localhost',
                                remote_port => 2000);

LOOP
    tlen := utl_tcp.read_raw(conn, rvalue, 1024);
    rvalue := utl_raw.concat(rtemp, rvalue);
    nvalue := utl_i18n.raw_to_nchar(rvalue, 'JA16SJIS', converted_len,
conversion_stat);
    if (converted_len < utl_raw.length(rvalue) )
    then
        rtemp := utl_raw.substr(rvalue, converted_len+1);
    else
        rtemp := '';
    end if;
    /* do anything you want with nvalue */
    /* e.g htp.prn(nvalue); */
END LOOP;
utl_tcp.close_connection(conn);
EXCEPTION
    WHEN utl_tcp.end_of_input THEN
        utl_tcp.close_connection(conn);
END;
```

## STRING\_TO\_RAW Function

This function converts a VARCHAR2 or NVARCHAR2 string to another valid Oracle character set and returns the result as RAW data.

### Syntax

```
UTL_I18N.STRING_TO_RAW(
    data          IN VARCHAR2 CHARACTER SET ANY_CS,
    dst_charset   IN VARCHAR2 DEFAULT NULL)
RETURN RAW;
```

### Parameters

**Table 225–20** *STRING\_TO\_RAW Function Parameters*

Parameter	Description
data	Specifies the VARCHAR2 or NVARCHAR2 string to convert.
dst_charset	Specifies the destination character set. If <code>dst_charset</code> is NULL, then the database character set is used for CHAR data and the national character set is used for NCHAR data.

### Usage Notes

If the user specifies an invalid character set, a NULL string, or a string whose length is 0, then the function returns a NULL string.

### Examples

```
DECLARE
    r raw(50);
    s varchar2(20);
BEGIN
    s:='abcdef' || chr(170);
    r:=utl_i18n.string_to_raw(s, 'utf8');
    dbms_output.put_line(rawtohex(r));
end;
/
```

This returns a hex value of '616263646566C2AA'.

## TRANSLITERATE Function

This function performs script transliteration. In this release, the TRANSLITERATE function only supports Japanese Kana conversion.

### Syntax

```
UTL_I18N.TRANSLITERATE (
  data IN VARCHAR2 CHARACTER SET ANY_CS,
  name IN VARCHAR2)
RETURN VARCHAR2 CHARACTER SET data%CHARSET;
```

### Parameters

**Table 225–21 TRANSLITERATE Function Parameters**

Parameter	Description
data	Specifies the data to be converted. Either CHAR or NCHAR data type can be specified.
name	Specifies the transliteration name string. For a list of valid names, see <a href="#">Table 225–22</a> .

### Constants

These options specify Japanese Kana conversions.

**Table 225–22 TRANSLITERATE Function Constants**

Constant Name	Value	Description
KANA_FWKATAKANA	'kana_ fwkatakana'	Converts any type of Kana character to a fullwidth Katakana character.
KANA_HWKATAKANA	'kana_ hwkatakana'	Converts any type of Kana character to a halfwidth Katakana character.
KANA_HIRAGANA	'kana_hiragana'	Converts any type of Kana character to a fullwidth Hiragana character.
FWKATAKANA_ HWKATAKANA	'fwkatakana_ hwkatakana'	Converts only fullwidth Katakana characters to halfwidth Katakana characters.
FWKATAKANA_ HIRAGANA	'fwkatakana_ hiragana'	Converts only fullwidth Katakana characters to fullwidth Hiragana characters.
HWKATAKANA_ FWKATAKANA	'hwkatakana_ fwkatakana'	Converts only halfwidth Katakana characters to fullwidth Katakana characters.
HWKATAKANA_ HIRAGANA	'hwkatakana_ hiragana'	Converts only halfwidth Katakana characters to fullwidth Hiragana characters.
HIRAGANA_ FWKATAKANA	'hiragana_ fwkatakana'	Converts only fullwidth Hiragana characters to fullwidth Katakana characters.
HIRAGANA_ HWKATAKANA	'hiragana_ hwkatakana'	Converts only fullwidth Hiragana characters to halfwidth Katakana characters.

### Usage Notes

The function returns the converted string.

## Examples

Given a table `japanese_emp`, containing an `NVARCHAR2` column `ename`, the following statement can be used to normalize all the kana names in `ename` to hiragana:

```
UPDATE japanese_emp
   SET ename = UTL_I18N.TRANSLITERATE (ename, 'kana_hiragana');
```

Figure shows how this output might look.

**Figure 225-1 Loading Locale-Specific Data to the Database**

```

タナカ
たなか
タナカ
↓
たなか
たなか
たなか

```

The image `kana_small_arrow.gif` is partially described in the preceding text. It shows how three versions of the word "Tanaka" can be standardized and converted into one form.

\*\*\*\*\*

The following statement normalizes one kana name to hiragana:

```
DECLARE
  Name  japanese_emp.ename%TYPE;
  Eno   CONSTANT  NUMBER(4) := 1;
BEGIN
  SELECT ename INTO name FROM japanese_emp WHERE enumber = eno;
  name := UTL_I18N.TRANSLITERATE(name, UTL_I18N.KANA_HIRAGANA);
  UPDATE japanese_emp SET ename = name WHERE enumber = eno;
EXCEPTION
  WHEN UTL_I18N.UNSUPPORTED_TRANSLITERATION THEN
    DBMS_OUTPUT.PUT_LINE('transliteration not supported');
END;
/
```

## UNESCAPE\_REFERENCE Function

This function returns a string from an input string that contains character references. It decodes each character reference to the corresponding character value.

**See Also:** ["ESCAPE\\_REFERENCE Function"](#) on page 225-9 for more information about escape sequences

### Syntax

```
UTL_I18N.UNESCAPE_REFERENCE (  
    str IN VARCHAR2 CHARACTER SET ANY_CS)  
RETURN VARCHAR2 CHARACTER SET str%CHARSET;
```

### Parameters

**Table 225–23 UNESCAPE\_REFERENCE Function Parameters**

Parameter	Description
str	Specifies the input string

### Usage Notes

If the user specifies a NULL string or a string whose length is 0, then the function returns a NULL string. If the function fails, then it returns the original string.

### Examples

```
UTL_I18N.UNESCAPE_REFERENCE('hello &lt; &#xe5;')
```

This returns 'hello < | |chr(229).'

The `UTL_INADDR` package provides a PL/SQL procedures to support internet addressing. It provides an API to retrieve host names and IP addresses of local and remote hosts.

This chapter contains the following topics:

- [Using UTL\\_INADDR](#)
  - Security Model
  - Exceptions
  - Examples
- [Summary of UTL\\_INADDR Subprograms](#)

## Using UTL\_INADDR

- [Security Model](#)
- [Exceptions](#)
- [Examples](#)



## Security Model

This package is an invoker's rights package, which means that the invoking user must be granted the `connect` privilege in the access control list assigned to the remote network host to which he or she wishes to connect.

---

---

**Note:** For more information about managing fine-grained access, see *Oracle Database Security Guide*.

---

---

## Exceptions

**Table 226–1** *Exception - Internet Address Package*

<b>Number</b>	<b>Exception</b>	<b>Description</b>
ORA-24247	NETWORK_ACCESS_DENIED	Access to network is denied.
ORA-29257	UNKNOWN_HOST	The host is unknown.

## Examples

Retrieve the local host name and IP address.

```
SET serveroutput on
BEGIN
  DBMS_OUTPUT.PUT_LINE(UTL_INADDR.GET_HOST_NAME); -- get local host name
  DBMS_OUTPUT.PUT_LINE(UTL_INADDR.GET_HOST_ADDRESS); -- get local IP addr
END;
/
```

## Summary of UTL\_INADDR Subprograms

**Table 226–2 UTL\_INADDR Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">GET_HOST_ADDRESS Function</a> on page 226-7	Retrieves the IP address of the local or remote host given its name
<a href="#">GET_HOST_NAME Function</a> on page 226-8	Retrieves the name of the local or remote host given its IP address

## GET\_HOST\_ADDRESS Function

This function retrieves the IP address of the specified host.

### Syntax

```
UTL_INADDR.GET_HOST_ADDRESS (
    host IN VARCHAR2 DEFAULT NULL)
RETURN host_address VARCHAR2;
```

### Parameters

**Table 226–3** *GET\_HOST\_ADDRESS Function Parameters*

Parameter	Description
host	The name of the host to retrieve the IP address.

### Return Values

**Table 226–4** *GET\_HOST\_ADDRESS Function Return Values*

Parameter	Description
host_address	The IP address of the specified host, or that of the local host if host is NULL.

### Exceptions

UNKNOWN\_HOST: The specified IP address is unknown

### Usage Notes

The permission to obtain the host name or IP address of the current host is controlled by the `resolve` privilege on `LOCALHOST`.

## GET\_HOST\_NAME Function

This function retrieves the name of the local or remote host given its IP address.

### Syntax

```
UTL_INADDR.GET_HOST_NAME (  
    ip IN VARCHAR2 DEFAULT NULL)  
RETURN host_name VARCHAR2;
```

### Parameters

**Table 226–5 GET\_HOST\_NAME Function Parameters**

Parameter	Description
ip	The IP address of the host used to determine its host name. If ip is not NULL, the official name of the host with its domain name is returned. If this is NULL, the name of the local host is returned and the name does not contain the domain to which the local host belongs.

### Return Values

**Table 226–6 GET\_HOST\_NAME Function Return Values**

Parameter	Description
host_name	The name of the local or remote host of the specified IP address

### Exceptions

UNKNOWN\_HOST: The specified IP address is unknown

### Usage Notes

The permission to obtain the host name or IP address of the current host is controlled by the `resolve` privilege granted through `DBMS_NETWORK_ACL_ADMIN` on `LOCALHOST`.

The UTL\_IDENT package specifies which Database or client PL/SQL is running.

This chapter contains the following topics:

- [Using UTL\\_IDENT](#)
  - Overview
  - Security Model
  - Constants

## Using UTL\_IDENT

---

This section contains topics which relate to using the UTL\_IDENT package.

- [Overview](#)
- [Security Model](#)
- [Constants](#)



## Overview

The UTL\_IDENT package is intended for use for conditional compilation of PL/SQL packages that are supported by Oracle, TimesTen Database, and clients such as Oracle Forms.

## Security Model

The `UTL_IDENT` package runs as the package owner `SYS`. The public synonym `UTL_IDENT`, and `EXECUTE` permission on this package is granted to `PUBLIC`.

## Constants

The UTL\_IDENT package uses the constants shown in [Table 227-1, "UTL\\_IDENT Constants"](#).

**Table 227-1 UTL\_IDENT Constants**

Constant	Type	Value	Description
IS_ORACLE_SERVER	BOOLEAN	TRUE/FALSE	Stipulates if Oracle Server or not
IS_ORACLE_CLIENT	BOOLEAN	TRUE/FALSE	Stipulates if Oracle Client or not
IS_TIMESTEN	BOOLEAN	TRUE/FALSE	Stipulates if TimesTen or not
IS_ORACLE_FORMS	BOOLEAN	TRUE/FALSE	Stipulates if Oracle Forms or not



UTL\_LMS retrieves and formats error messages in different languages.

**See Also:** *Oracle Database Globalization Support Guide*

This chapter contains the following topics:

- [Using UTL\\_LMS](#)
  - Security Model
- [Summary of UTL\\_LMS Subprograms](#)

## Using UTL\_LMS

This section contains topics which relate to using the UTL\_LMS package.

- [Security Model](#)

## Security Model

This package must be created as the user `SYS`.

## Summary of UTL\_LMS Subprograms

**Table 228–1 UTL\_LMS Package Subprograms**

<b>Function</b>	<b>Description</b>
<a href="#">FORMAT_MESSAGE Function</a> on page 228-5	Formats a retrieved error message
<a href="#">GET_MESSAGE Function</a> on page 228-6	Retrieves an error message based on error number, product, facility, language, and message specified



## FORMAT\_MESSAGE Function

This function formats a message retrieved by the `GET_MESSAGE` function and returns the formatted message. If the function fails, then it returns a `NULL` result.

The following table shows special characters that can be used in the format string.

Special Character	Description
'%s'	Substitute the next string argument
'%d'	Substitute the next integer argument
'%%'	Represents the special character %

### Syntax

```
UTL_LMS.FORMAT_MESSAGE (
  format IN VARCHAR2 CHARACTER SET ANY_CS,
  args   IN VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL)
RETURN VARCHAR2 CHARACTER SET format%CHARSET;
```

### Parameters

**Table 228–2** *FORMAT\_MESSAGE Procedure Parameters*

Parameter	Description
format	Specifies the string to format
args	Specifies the list of arguments

### Examples

```
DECLARE
  s varchar2(200);
  i pls_integer;
BEGIN
  i:= utl_lms.get_message(26052, 'rdbms', 'ora', 'french', s);
  dbms_output.put_line('before format, message is: '||s);
  dbms_output.put_line('formatted message is: '||utl_lms.format_message(s, 9,
'my_column_name');
END;
/
```

The following is an unformatted message:

Type %d non pris en charge pour l'expression SQL sur la colonne %s.

The following is the formatted message:

Type 9 non pris en charge pour l'expression SQL sur la colonne my\_column\_name.

## GET\_MESSAGE Function

This function retrieves an Oracle error message. The user can define user-specific error messages with the `lmsgen` utility.

It returns 0 when it is successful. It returns -1 when it fails.

**See Also:** *Oracle Database Globalization Support Guide* for more information about the `lmsgen` utility

### Syntax

```
UTL_LMS.GET_MESSAGE (
    errnum    IN PLS_INTEGER,
    product   IN VARCHAR2,
    facility  IN VARCHAR2,
    language  IN VARCHAR2,
    message   OUT NOCOPY VARCHAR2CHARACTER SET ANY_CS)
RETURN PLS_INTEGER;
```

### Parameters

**Table 228–3** GET\_MESSAGE Function Parameters

Parameter	Description
<code>errnum</code>	Specifies the error number. Example: '972' (for ORA-00972)
<code>product</code>	Specifies the product to which the error message applies Example: 'rdbms'
<code>facility</code>	Specifies the error message prefix Example: 'ora'
<code>language</code>	Specifies the language of the message. The parameter is case-insensitive. The default is NULL, which causes <code>GET_MESSAGE</code> to use the value of the <code>NLS_LANGUAGE</code> session parameter.
<code>message</code>	Specifies the output buffer for the retrieved message

### Usage Notes

If the `language` parameter is set to NULL, then the value of the `NLS_LANGUAGE` session parameter is used as the default.

### Examples

```
DECLARE
    s varchar2(200);
    i pls_integer;
BEGIN
    i:=utl_lms.get_message(601, 'rdbms', 'oci', 'french', s);
    dbms_output.put_line('OCI--00601 is: '||s);
END
/
```

The following output results:

```
OCI--00601 is: Echec du processus de nettoyage.
```

The UTL\_MAIL package is a utility for managing email which includes commonly used email features, such as attachments, CC, and BCC.

This chapter contains the following topics:

- [Using UTL\\_MAIL](#)
  - Security Model
  - Operational Notes
  - Rules and Limits
- [Summary of UTL\\_MAIL Subprograms](#)

## Using UTL\_MAIL

- [Security Model](#)
- [Operational Notes](#)
- [Rules and Limits](#)

## Security Model

UTL\_MAIL is not installed by default because of the SMTP\_OUT\_SERVER configuration requirement and the security exposure this involves. In installing UTL\_MAIL, you should take steps to prevent the port defined by SMTP\_OUT\_SERVER being swamped by data transmissions .

This package is now an invoker's rights package and the invoking user will need the connect privilege granted in the access control list assigned to the remote network host to which he wants to connect.

---

---

**Note:** For more information, see Managing Fine-grained Access to External Network Services in Oracle Database Security Guide

---

---

## Operational Notes

You must both install UTL\_MAIL and define the SMTP\_OUT\_SERVER.

- To install UTL\_MAIL:

```
sqlplus sys/<pwd>  
SQL> @$ORACLE_HOME/rdbms/admin/utlmail.sql  
SQL> @$ORACLE_HOME/rdbms/admin/prvtmail.plb
```

- You define the SMTP\_OUT\_SERVER parameter in the `init.ora` rdbms initialization file. However, if SMTP\_OUT\_SERVER is not defined, this invokes a default of DB\_DOMAIN which is guaranteed to be defined to perform appropriately.

## Rules and Limits

Use `UTL_MAIL` only within the context of the ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary-Coded Decimal Interchange Code) codes.

## Summary of UTL\_MAIL Subprograms

**Table 229–1 UTL\_MAIL Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">SEND Procedure</a> on page 229-7	Packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients
<a href="#">SEND_ATTACH_RAW Procedure</a> on page 229-8	Represents the SEND Procedure overloaded for RAW attachments
<a href="#">SEND_ATTACH_VARCHAR2 Procedure</a> on page 229-9	Represents the SEND Procedure overloaded for VARCHAR2 attachments



## SEND Procedure

This procedure packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients. It hides the SMTP API and exposes a one-line email facility for ease of use.

### Syntax

```
UTL_MAIL.SEND (
  sender      IN   VARCHAR2 CHARACTER SET ANY_CS,
  recipients  IN   VARCHAR2 CHARACTER SET ANY_CS,
  cc          IN   VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  bcc         IN   VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  subject     IN   VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  message     IN   VARCHAR2 CHARACTER SET ANY_CS,
  mime_type   IN   VARCHAR2 DEFAULT 'text/plain; charset=us-ascii',
  priority    IN   PLS_INTEGER DEFAULT 3,
  replyto     IN   VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

### Parameters

**Table 229–2 SEND Procedure Parameters**

Parameter	Description
sender	Email address of the sender
recipients	Email addresses of the recipient(s), separated by commas
cc	Email addresses of the CC recipient(s), separated by commas, default is NULL
bcc	Email addresses of the BCC recipient(s), separated by commas, default is NULL
subject	String to be included as email subject string, default is NULL
message	Text message body
mime_type	Mime type of the message, default is 'text/plain; charset=us-ascii'
priority	Message priority, which maps to the X-priority field. 1 is the highest priority and 5 the lowest. The default is 3.
replyto	Defines to whom the reply email is to be sent

## SEND\_ATTACH\_RAW Procedure

This procedure is the SEND Procedure overloaded for RAW attachments.

### Syntax

```
UTL_MAIL.SEND_ATTACH_RAW (
  sender          IN  VARCHAR2 CHARACTER SET ANY_CS,
  recipients      IN  VARCHAR2 CHARACTER SET ANY_CS,
  cc              IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  bcc             IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  subject         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  message         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  mime_type       IN  VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                    DEFAULT 'text/plain; charset=us-ascii',
  priority        IN  PLS_INTEGER DEFAULT 3,
  attachment      IN  RAW,
  att_inline      IN  BOOLEAN DEFAULT TRUE,
  att_mime_type   IN  VARCHAR2 CHARACTER SET ANY_CS
                    DEFAULT 'text/plain; charset=us-ascii',
  att_filename    IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  replyto         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

### Parameters

**Table 229-3 SEND\_ATTACH\_RAW Procedure Parameters**

Parameter	Description
sender	Email address of the sender
recipients	Email addresses of the recipient(s), separated by commas
cc	Email addresses of the CC recipient(s), separated by commas, default is NULL
bcc	Email addresses of the BCC recipient(s), separated by commas, default is NULL
subject	String to be included as email subject string, default is NULL
message	Text message body
mime_type	Mime type of the message, default is 'text/plain; charset=us-ascii'
priority	Message priority, which maps to the X-priority field. 1 is the highest priority and 5 the lowest. The default is 3.
attachment	RAW attachment
att_inline	Specifies whether the attachment is viewable inline with the message body, default is TRUE
att_mime_type	Mime type of the attachment, default is 'application/octet'
att_filename	String specifying a filename containing the attachment, default is NULL
replyto	Defines to whom the reply email is to be sent

## SEND\_ATTACH\_VARCHAR2 Procedure

This procedure is the SEND Procedure overloaded for VARCHAR2 attachments.

### Syntax

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
  sender          IN    VARCHAR2 CHARACTER SET ANY_CS,
  recipients      IN    VARCHAR2 CHARACTER SET ANY_CS,
  cc              IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  bcc             IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  subject         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  message         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  mime_type       IN    VARCHAR2 CHARACTER SET ANY_CS
                                DEFAULT 'text/plain; charset=us-ascii',
  priority        IN    PLS_INTEGER DEFAULT 3,
  attachment      IN    VARCHAR2 CHARACTER SET ANY_CS, ,
  att_inline      IN    BOOLEAN DEFAULT TRUE,
  att_mime_type   IN    VARCHAR2 CHARACTER SET ANY_CS
                                DEFAULT 'text/plain; charset=us-ascii',
  att_filename    IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  replyto        IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

### Parameters

**Table 229–4 SEND\_ATTACH\_VARCHAR2 Procedure Parameters**

Parameter	Description
sender	Email address of the sender
recipients	Email addresses of the recipient(s), separated by commas
cc	Email addresses of the CC recipient(s), separated by commas, default is NULL
bcc	Email addresses of the BCC recipient(s), separated by commas, default is NULL
subject	String to be included as email subject string, default is NULL
message	Text message body
mime_type	Mime type of the message, default is 'text/plain; charset=us-ascii'
priority	Message priority, which maps to the X-priority field. 1 is the highest priority and 5 the lowest. The default is 3.
attachment	Text attachment
att_inline	Specifies whether the attachment is inline, default TRUE
att_mime_type	Mime type of the attachment, default is 'text/plain; charset=us-ascii'
att_filename	String specifying a filename containing the attachment, default is NULL
replyto	Defines to whom the reply email is to be sent



The UTL\_MATCH package facilitates matching two records. This is typically used to match names, such as two First Names or two Last Names.

This chapter contains the following topics:

- [Using UTL\\_MATCH](#)
  - Overview
  - Security Model
- [Summary of UTL\\_MATCH Subprograms](#)

## Using UTL\_MATCH

---

- [Overview](#)
- [Security Model](#)

## Overview

"Edit Distance" also known as "Levenshtein Distance" (named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965), is a measure of Similarity between two strings, s1 and s2. The distance is the number of insertions, deletions or substitutions required to transform s1 to s2.

The Edit Distance between strings "shackleford" and "shackelford" = 2

The "Jaro-Winkler algorithm" is another way of calculating Edit distance between two strings. This method, developed at the U.S. Census, is a String Comparator measure that gives values of partial agreement between two strings. The string comparator accounts for length of strings and partially accounts for typical human errors made in alphanumeric strings.

Table 230–1 shows similarity values returned by Jaro-Winkler and Edit Distance

**Table 230–1 Comparison between normalized values returned by Jaro-Winkler and Edit Distance algorithms**

String 1	String 2	Jaro Winkler	Edit Distance
Dunningham	Cunnigham	89	80
Abroms	Abrams	92	83
Lampley	Campley	90	86
Marhta	Martha	96	67
Jonathon	Jonathan	95	88
Jeraldine	Geraldine	92	89

## Security Model

The `UTL_MATCH` package runs with definer's rights. `UTL_MATCH` must be created under `SYS`. Operations provided by this package are performed with `SYS` privileges.



---

## Summary of UTL\_MATCH Subprograms

---

**Table 230–2 DBMS\_ALERT Package Subprograms**

Subprogram	Description
<a href="#">EDIT_DISTANCE Function</a> on page 230-6	Calculates the number of changes required to transform string-1 into string-2
<a href="#">EDIT_DISTANCE_SIMILARITY Function</a> on page 230-7	Calculates the number of changes required to transform string-1 into string-2, returning a value between 0 (no match) and 100 (perfect match)
<a href="#">JARO_WINKLER Function</a> on page 230-8	Calculates the measure of agreement between string-1 and string-2
<a href="#">JARO_WINKLER_SIMILARITY Function</a> on page 230-9	Calculates the measure of agreement between string-1 and string-2, returning a value between 0 (no match) and 100 (perfect match)

## EDIT\_DISTANCE Function

This function calculates the number of insertions, deletions or substitutions required to transform string-1 into string-2.

### Syntax

```
UTL_MATCH.EDIT_DISTANCE (  
    s1 IN VARCHAR2,  
    s2 IN VARCHAR2)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 230–3** *EDIT\_DISTANCE Function Parameters*

Parameter	Description
s1	The string to be transformed
s2	The string into which s1 is to be transformed

### Examples

```
SELECT UTL_MATCH.EDIT_DISTANCE('shackleford', 'shackelford') FROM DUAL;  
-----  
returns 2
```

## EDIT\_DISTANCE\_SIMILARITY Function

This function calculates the number of insertions, deletions or substitutions required to transform string-1 into string-2, and returns the Normalized value of the Edit Distance between two Strings. The value is typically between 0 (no match) and 100 (perfect match).

### Syntax

```
UTL_MATCH.EDIT_DISTANCE_SIMILARITY (
    s1 IN VARCHAR2,
    s2 IN VARCHAR2)
RETURN PLS_INTEGER;
```

### Parameters

**Table 230–4** *EDIT\_DISTANCE\_SIMILARITY Function Parameters*

Parameter	Description
s1	The string to be transformed
s2	The string into which s1 is to be transformed

### Examples

```
SELECT UTL_MATCH.EDIT_DISTANCE_SIMILARITY('shackleford', 'shackelford') FROM DUAL;
-----
returns 82
```

## JARO\_WINKLER Function

This function calculates the measure of agreement between two strings.

### Syntax

```
UTL_MATCH.JARO_WINKLER (  
    s1 IN VARCHAR2,  
    s2 IN VARCHAR2)  
RETURN BINARY_DOUBLE;
```

### Parameters

**Table 230–5 JARO\_WINKLER Function Parameters**

Parameter	Description
s1	Input
s2	input

### Examples

```
SELECT UTL_MATCH.JARO_WINKLER('shackleford', 'shackelford') FROM DUAL;  
-----  
returns 9.818E-001
```

## JARO\_WINKLER\_SIMILARITY Function

This function calculates the measure of agreement between two strings, and returns a score between 0 (no match) and 100 (perfect match).

### Syntax

```
UTL_MATCH.JARO_WINKLER (  
    s1 IN VARCHAR2,  
    s2 IN VARCHAR2)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 230–6** *JARO\_WINKLER Function Parameters*

Parameter	Description
s1	Input
s2	input

### Examples

```
SELECT UTL_MATCH.JARO_WINKLER_SIMILARITY('shackleford', 'shackelford') FROM DUAL;  
-----  
returns 98
```



The UTL\_NLA package exposes a subset of the BLAS and LAPACK (Version 3.0) operations on vectors and matrices represented as VARRAYs.

This chapter contains the following topics:

- [Using UTL\\_NLA](#)
  - Overview
  - Rules and Limits
  - Security Model
- [Subprogram Groups](#)
  - BLAS Level 1 (Vector-Vector Operations) Subprograms
  - BLAS Level 2 (Matrix-Vector Operations) Subprograms
  - BLAS Level 3 (Matrix-Matrix Operations) Subprograms
  - LAPACK Driver Routines (Linear Equations) Subprograms
  - LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
- [Summary of UTL\\_NLA Subprograms](#)

## Using UTL\_NLA

This section contains topics which relate to using the UTL\_NLA package.

- [Overview](#)
- [Rules and Limits](#)
- [Security Model](#)



## Overview

The UTL\_NLA package exposes a subset of the BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage )(Version 3.0) operations on vectors and matrices represented as VARRAYS.

### Standards

For more information on the BLAS and LAPACK standards see

<http://www.netlib.org/blas/>

<http://www.netlib.org/lapack/>

### Required Expertise

Users of this package are expected to have a sound grasp of linear algebra in general and of the BLAS and LAPACK libraries in particular.

### Implementation

The mapping between BLAS and LAPACK procedures and their corresponding PL/SQL calls is one-to-one.

- All BLAS functions have the `BLAS_` prefix (for example, the [BLAS\\_ASUM Functions](#)). The subroutines and functions in BLAS are mapped to PL/SQL procedures and functions, respectively.
- All LAPACK functions have the `LAPACK_` prefix (for example, the [LAPACK\\_GBSV Procedures](#)). The subroutines in LAPACK are mapped to PL/SQL procedures. Procedures that perform the same operation but differ only on the datatype of the arguments have the same overloaded names.

The mapping between BLAS and LAPACK procedure parameters and those of their corresponding PL/SQL subprograms is almost one-to-one.

- Also in the PL/SQL interface for LAPACK, all `/work/` arguments have been removed. The UTL\_NLA package manages the allocation and de-allocation of all work areas required by the libraries.
- A new optional parameter, `pack`, has been added to the end of each LAPACK procedure that specifies if the matrix has been linearized in the row-major or column-major (default) format.

## Rules and Limits

Vectors and matrices are stored in `VARRAYs` with a maximum size of one million entries. Given this restriction, `UTL_NLA` vectors can be up to one million entries but matrices need to be of size  $R \times C \leq 1,000,000$ .

## Security Model

The UTL\_NLA package is owned by user SYS and is installed as part of database installation. Execution privilege on the package is granted to public. The routines in the package are run with invokers' rights (run with the privileges of the current user).

---

## Subprogram Groups

- [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#)
- [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#)
- [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#)
- [LAPACK Driver Routines \(Linear Equations\) Subprograms](#)
- [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#)

## BLAS Level 1 (Vector-Vector Operations) Subprograms

**Table 231-1** *BLAS Level 1 (Vector-Vector Operations) Subprograms*

<b>Subprogram</b>	<b>Description</b>
<a href="#">BLAS_ASUM Functions</a> on page 231-18	Computes the sum of the absolute values of the vector components
<a href="#">BLAS_AXPY Procedures</a> on page 231-19	Copies $\alpha X + Y$ into vector $Y$
<a href="#">BLAS_COPY Procedures</a> on page 231-20	Copies the contents of vector $X$ to vector $Y$
<a href="#">BLAS_DOT Functions</a> on page 231-21	Returns the dot (scalar) product of two vectors $X$ and $Y$
<a href="#">BLAS_IAMAX Functions</a> on page 231-31	Computes the index of the first element of a vector that has the largest absolute value
<a href="#">BLAS_NRM2 Functions</a> on page 231-32	Computes the vector 2-norm (Euclidean norm)
<a href="#">BLAS_ROT Procedures</a> on page 231-33	Returns the plane rotation of points
<a href="#">BLAS_ROTG Procedures</a> on page 231-34	Returns the Givens rotation of points
<a href="#">BLAS_SCAL Procedures</a> on page 231-35	Scales a vector by a constant
<a href="#">BLAS_SWAP Procedures</a> on page 231-44	Swaps the contents of two vectors each of size $n$

## BLAS Level 2 (Matrix-Vector Operations) Subprograms

**Table 231–2** *BLAS Level 2 (Matrix-Vector Operations) Subprograms*

Subprogram	Description
<a href="#">BLAS_GBMV Procedures</a> on page 231-22	Performs the matrix-vector operation $y := \alpha A x + \beta y$ or $y := \alpha A' x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are vectors and $A$ is an $m$ by $n$ band matrix, with $k_l$ sub-diagonals and $k_u$ super-diagonals
<a href="#">BLAS_GEMV Procedures</a> on page 231-27	Performs the matrix-vector operations $y := \alpha A x + \beta y$ or $y := \alpha A' x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are vectors and $A$ is an $m$ by $n$ matrix
<a href="#">BLAS_GER Procedures</a> on page 231-29	Performs a rank 1 operation $A := \alpha x y' + A$ where $\alpha$ is a scalar, $x$ is an $m$ element vector, $y$ is an $n$ element vector and $A$ is an $m$ by $n$ matrix
<a href="#">BLAS_SBMV Procedures</a> on page 231-42	Performs a matrix-vector operation $y := \alpha A x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric band matrix, with $k$ super-diagonals
<a href="#">BLAS_SPMV Procedures</a> on page 231-36	Performs a matrix-vector operation $y := \alpha A x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric matrix, supplied in packed form
<a href="#">BLAS_SPR Procedures</a> on page 231-38	Performs a symmetric rank 1 operation $A := \alpha x x' + A$ where $\alpha$ is a real scalar, $x$ is an $n$ element vector, and $A$ is an $n$ by $n$ symmetric matrix, supplied in packed form
<a href="#">BLAS_SPR2 Procedures</a> on page 231-40	Performs a symmetric rank 2 operation $A := \alpha x y' + \alpha y x' + A$ where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n$ by $n$ symmetric matrix, supplied in packed form
<a href="#">BLAS_SBMV Procedures</a> on page 231-42	Performs a matrix-vector operation $y := \alpha A x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric band matrix, with $k$ super-diagonals
<a href="#">BLAS_SYMV Procedures</a> on page 231-47	Performs a matrix-vector operation $y := \alpha A x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric matrix
<a href="#">BLAS_SYR Procedures</a> on page 231-49	Performs a symmetric rank 1 operation $A := \alpha x x' + A$ where $\alpha$ is a real scalar, $x$ is an $n$ element vector, and $A$ is an $n$ by $n$ symmetric matrix
<a href="#">BLAS_SYR2 Procedures</a> on page 231-51	Performs a symmetric rank 2 operation $A := \alpha x y' + \alpha y x' + A$ where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n$ by $n$ symmetric matrix
<a href="#">BLAS_TBMV Procedures</a> on page 231-58	Performs a matrix-vector operation $x := A x$ or $A' x = b$ where $x$ is an $n$ element vector and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals
<a href="#">BLAS_TBSV Procedures</a> on page 231-60	Solves one of the systems of equation $A x = b$ or $A' x = b$ where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals

**Table 231-2 (Cont.) BLAS Level 2 (Matrix-Vector Operations) Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">BLAS_TPMV Procedures</a> on page 231-62	Performs a matrix-vector operation $x := A*x$ or $x := A' *x$ where $x$ is an $n$ element vector and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form
<a href="#">BLAS_TPSV Procedures</a> on page 231-64	Solves one of the systems of equation $A*x = b$ or $A' *x = b$ where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form
<a href="#">BLAS_TRMV Procedures</a> on page 231-69	Performs a matrix-vector operation $x := A*x$ or $x := A' *x$ where $x$ is an $n$ element vector and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix
<a href="#">BLAS_TRSV Procedures</a> on page 231-74	Solves one of the systems of equation $A*x = b$ or $A' *x = b$ where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix

## BLAS Level 3 (Matrix-Matrix Operations) Subprograms

**Table 231–3** *BLAS Level 3 (Matrix-Matrix Operations) Subprograms*

Subprogram	Description
<a href="#">BLAS_GEMM Procedures</a> on page 231-27	Performs one of the matrix-vector operations $C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ where $\text{op}(X)$ is one of $\text{op}(X) = X$ or $\text{op}(X) = X'$ where $\alpha$ and $\beta$ are scalars, and $A$ , $B$ and $C$ are matrices, with $\text{op}(A)$ an $m$ by $k$ matrix, $\text{op}(B)$ a $k$ by $n$ matrix and $C$ an $m$ by $n$ matrix
<a href="#">BLAS_SYMM Procedures</a> on page 231-45	Performs one of the matrix-vector operations $C := \alpha * A * B + \beta * C$ or $C := \alpha * B * A + \beta * C$ where $\alpha$ and $\beta$ are scalars, $A$ is a symmetric matrix, and $B$ and $C$ are $m$ by $n$ matrices
<a href="#">BLAS_SYR2K Procedures</a> on page 231-53	Performs one of the symmetric rank2 k operations $C := \alpha * A * B' + \alpha * B * A' + \beta * C$ or $C := \alpha * A' * B + \alpha * B' * A + \beta * C$ where $\alpha$ and $\beta$ are scalars, $C$ is an $n$ by $n$ symmetric matrix and $A$ and $B$ are $n$ by $k$ matrices in the first case and $k$ by $n$ matrices in the second case
<a href="#">BLAS_SYRK Procedures</a> on page 231-56	Performs one of the symmetric rank k operations $C := \alpha * A * A' + \beta * C$ or $C := \alpha * A' * A + \beta * C$ where $\alpha$ and $\beta$ are scalars, $C$ is an $n$ by $n$ symmetric matrix and $A$ is an $n$ by $k$ matrix in the first case and a $k$ by $n$ matrix in the second case
<a href="#">BLAS_TRMM Procedures</a> on page 231-66	Performs one of the matrix-vector operations $B := \alpha * \text{op}(A) * B$ or $B := \alpha * B * \text{op}(A)$ where $\alpha$ is a scalar, $B$ is an $m$ by $n$ matrix, $A$ is a unit, or non-unit, upper or lower triangular matrix and $\text{op}(A)$ is one of two alternatives
<a href="#">BLAS_TRSM Procedures</a> on page 231-71	Performs one of the matrix-vector operations $\text{op}(A) * X = \alpha * B$ or $X * \text{op}(A) = \alpha * B$ where $\alpha$ is a scalar, $X$ and $B$ are $m$ by $n$ matrices, $A$ is a unit, or non-unit, upper or lower triangular matrix, $\text{op}(A)$ is one of two alternatives. The matrix $X$ is overwritten on $B$



## LAPACK Driver Routines (Linear Equations) Subprograms

**Table 231–4** LAPACK Driver Routines (Linear Equations) Subprograms

Subprogram	Description
<a href="#">LAPACK_GBSV Procedures</a> on page 231-76	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The LU decomposition with partial pivoting and row interchanges is used to factor $A$ .
<a href="#">LAPACK_GESV Procedures</a> on page 231-85	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The LU decomposition with partial pivoting and row interchanges is used to factor $A$ .
<a href="#">LAPACK_GTSV Procedures</a> on page 231-93	This procedure solves the equation $a * x = b$ where $a$ is an $n$ by $n$ tridiagonal matrix, by Gaussian elimination with partial pivoting.
<a href="#">LAPACK_PBSV Procedures</a> on page 231-95	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite band matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The Cholesky decomposition is used to factor $A$ .
<a href="#">LAPACK_POSV Procedures</a> on page 231-97	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The Cholesky decomposition is used to factor $A$ .
<a href="#">LAPACK_PPSV Procedures</a> on page 231-99	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite matrix stored in packed format and $x$ and $b$ are $n$ by $nrhs$ matrices. The Cholesky decomposition is used to factor $A$ .
<a href="#">LAPACK_PTSV Procedures</a> on page 231-101	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite tridiagonal matrix, and $x$ and $b$ are $n$ by $nrhs$ matrices.
<a href="#">LAPACK_SPSV Procedures</a> on page 231-111	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric matrix stored in packed format, and $x$ and $b$ are $n$ by $nrhs$ matrices. The diagonal pivoting method is used to factor $A$ .
<a href="#">LAPACK_SYSV Procedures</a> on page 231-121	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric matrix, and $x$ and $b$ are $n$ by $nrhs$ matrices. The diagonal pivoting method is used to factor $A$ .

## LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

**Table 231–5 LAPACK Driver Routines (LLS and Eigenvalue Problems)**

Subprogram	Description
<a href="#">LAPACK_GEES Procedures</a> on page 231-78	Computes for an $n$ by $n$ real nonsymmetric matrix $A$ , the eigenvalues, the real Schur form $T$ , and, optionally, the matrix of Schur vectors $Z$ . This gives the Schur factorization $A = Z^*T^*(Z^{**T})$ .
<a href="#">LAPACK_GEEV Procedures</a> on page 231-90	Computes for an $n$ by $n$ real nonsymmetric matrix $A$ , the eigenvalues and, optionally, the left and/or right eigenvectors.
<a href="#">LAPACK_GELS Procedures</a> on page 231-80	Solves overdetermined or underdetermined real linear systems involving an $m$ by $n$ matrix $A$ , or its transpose, using a QR or LQ factorization of $A$ . It is assumed that $A$ has full rank.
<a href="#">LAPACK_GESDD Procedures</a> on page 231-82	Computes the singular value decomposition (SVD) of a real $m$ by $n$ matrix $A$ , optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.
<a href="#">LAPACK_GESVD Procedures</a> on page 231-87	Computes the singular value decomposition (SVD) of a real $m$ by $n$ matrix $A$ , optionally computing the left and/or right singular vectors. The SVD is written $A = U * SIGMA * \text{transpose}(V)$ .
<a href="#">LAPACK_SBEV Procedures</a> on page 231-103	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix $A$
<a href="#">LAPACK_SBEVD Procedures</a> on page 231-105	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ . If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.
<a href="#">LAPACK_SPEV Procedures</a> on page 231-107	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ in packed storage
<a href="#">LAPACK_SPEVD Procedures</a> on page 231-109	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.
<a href="#">LAPACK_STEV Procedures</a> on page 231-113	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$
<a href="#">LAPACK_STEVD Procedures</a> on page 231-115	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$ . If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.
<a href="#">LAPACK_SYEV Procedures</a> on page 231-117	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$
<a href="#">LAPACK_SYEVD Procedures</a> on page 231-119	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ . If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

## Summary of UTL\_NLA Subprograms

**Table 231–6 UTL\_NLA Package Subprograms**

Subprogram	Description	Group
<a href="#">BLAS_ASUM Functions</a> on page 231-18	Computes the sum of the absolute values of the vector components	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_AXPY Procedures</a> on page 231-19	Copies $\alpha X + Y$ into vector $Y$	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_COPY Procedures</a> on page 231-20	Copies the contents of vector $X$ to vector $Y$	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_DOT Functions</a> on page 231-21	Returns the dot (scalar) product of two vectors $X$ and $Y$	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_GBMV Procedures</a> on page 231-22	Performs the matrix-vector operation $y := \alpha A * x + \beta y$ or $y := \alpha A' * x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are vectors and $A$ is an $m$ by $n$ band matrix, with $k_1$ sub-diagonals and $k_u$ super-diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_GEMM Procedures</a> on page 231-27	Performs one of the matrix-vector operations where $\alpha$ and $\beta$ are scalars, and $A$ , $B$ and $C$ are matrices, with $op(A)$ an $m$ by $k$ matrix, $op(B)$ a $k$ by $n$ matrix and $C$ an $m$ by $n$ matrix	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
<a href="#">BLAS_GEMV Procedures</a> on page 231-27	Performs the matrix-vector operations $y := \alpha A * x + \beta y$ or $y := \alpha A' * x + \beta y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are vectors and $A$ is an $m$ by $n$ matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_GER Procedures</a> on page 231-29	Performs a rank 1 operation $A := \alpha x * y' + A$ where $\alpha$ is a scalar, $x$ is an $m$ element vector, $y$ is an $n$ element vector and $A$ is an $m$ by $n$ matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_IAMAX Functions</a> on page 231-31	Computes the index of the first element of a vector that has the largest absolute value	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_NRM2 Functions</a> on page 231-32	Computes the vector 2-norm (Euclidean norm)	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_ROT Procedures</a> on page 231-33	Returns the plane rotation of points	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_ROTG Procedures</a> on page 231-34	Returns the Givens rotation of points	BLAS Level 1 (Vector-Vector Operations) Subprograms

**Table 231–6 (Cont.) UTL\_NLA Package Subprograms**

Subprogram	Description	Group
<a href="#">BLAS_SBMV Procedures</a> on page 231-42	Performs a matrix-vector operation $y := \alpha * A * x + \beta * y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric band matrix, with $k$ super-diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SCAL Procedures</a> on page 231-35	Scales a vector by a constant	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_SPMV Procedures</a> on page 231-36	Performs a matrix-vector operation $y := \alpha * A * x + \beta * y$ where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SPR Procedures</a> on page 231-38	Performs a symmetric rank 1 operation $A := \alpha * x * x' + A$ where $\alpha$ is a real scalar, $x$ is an $n$ element vector, and $A$ is an $n$ by $n$ symmetric matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SPR2 Procedures</a> on page 231-40	Performs a symmetric rank 2 operation where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n$ by $n$ symmetric matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SWAP Procedures</a> on page 231-44	Swaps the contents of two vectors each of size $n$	BLAS Level 1 (Vector-Vector Operations) Subprograms
<a href="#">BLAS_SYMM Procedures</a> on page 231-45	Performs one of the matrix-vector operations where $\alpha$ and $\beta$ are scalars, $A$ is a symmetric matrix, and $B$ and $C$ are $m$ by $n$ matrices	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
<a href="#">BLAS_SYMV Procedures</a> on page 231-47	Performs a matrix-vector operation where $\alpha$ and $\beta$ are scalars, $x$ and $y$ are $n$ element vectors and $A$ is an $n$ by $n$ symmetric matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SYR Procedures</a> on page 231-49	Performs a symmetric rank 1 operation where $\alpha$ is a real scalar, $x$ is an $n$ element vector, and $A$ is an $n$ by $n$ symmetric matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SYR2 Procedures</a> on page 231-51	Performs a symmetric rank 2 operation where $\alpha$ is a scalar, $x$ and $y$ are $n$ element vectors, and $A$ is an $n$ by $n$ symmetric matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_SYR2K Procedures</a> on page 231-53	Performs one of the symmetric rank2 $k$ operations where $\alpha$ and $\beta$ are scalars, $C$ is an $n$ by $n$ symmetric matrix and $A$ and $B$ are $n$ by $k$ matrices in the first case and $k$ by $n$ matrices in the second case	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
<a href="#">BLAS_SYRK Procedures</a> on page 231-56	Performs one of the symmetric rank $k$ operations where $\alpha$ and $\beta$ are scalars, $C$ is an $n$ by $n$ symmetric matrix and $A$ is an $n$ by $k$ matrix in the first case and a $k$ by $n$ matrix in the second case	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
<a href="#">BLAS_TBMV Procedures</a> on page 231-58	Performs a matrix-vector operation where $x$ is an $n$ element vector and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms

**Table 231–6 (Cont.) UTL\_NLA Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">BLAS_TBSV Procedures</a> on page 231-60	Solves one of the systems of equation where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_TPMV Procedures</a> on page 231-62	Performs a matrix-vector operation where $x$ is an $n$ element vector and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_TPSV Procedures</a> on page 231-64	Solves one of the systems of equation where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix, supplied in packed form	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_TRMM Procedures</a> on page 231-66	Performs one of the matrix-vector operations where $\alpha$ is a scalar, $B$ is an $m$ by $n$ matrix, $A$ is a unit, or non-unit, upper or lower triangular matrix and $op(A)$ is one of two alternatives	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_TRMV Procedures</a> on page 231-69	Performs a matrix-vector operation where $x$ is an $n$ element vector and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">BLAS_TRSM Procedures</a> on page 231-71	Performs one of the matrix-vector operations $op(A) * X = \alpha * B$ or $X * op(A) = \alpha * B$ where $\alpha$ is a scalar, $X$ and $B$ are $m$ by $n$ matrices, $A$ is a unit, or non-unit, upper or lower triangular matrix, $op(A)$ is one of two alternatives. The matrix $X$ is overwritten on $B$	BLAS Level 3 (Matrix-Matrix Operations) Subprograms
<a href="#">BLAS_TRSV Procedures</a> on page 231-74	Solves one of the systems of equation where $b$ and $x$ are $n$ element vectors and $A$ is an $n$ by $n$ unit, or non-unit, upper or lower triangular matrix	BLAS Level 2 (Matrix-Vector Operations) Subprograms
<a href="#">LAPACK_GBSV Procedures</a> on page 231-76	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ matrix and $x$ and $b$ are $n$ by $n$ rhs matrices. The LU decomposition with partial pivoting and row interchanges is used to factor $A$ .	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_GEES Procedures</a> on page 231-78	Computes for an $n$ by $n$ real nonsymmetric matrix $A$ , the eigenvalues, the real Schur form $T$ , and, optionally, the matrix of Schur vectors $Z$ . This gives the Schur factorization $A = Z * T * (Z * * T)$ .	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_GEEV Procedures</a> on page 231-90	Computes for an $n$ by $n$ real nonsymmetric matrix $A$ , the eigenvalues and, optionally, the left and/or right eigenvectors.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_GELS Procedures</a> on page 231-80	Solves overdetermined or underdetermined real linear systems involving an $m$ by $n$ matrix $A$ , or its transpose, using a QR or LQ factorization of $A$ . It is assumed that $A$ has full rank.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_GESDD Procedures</a> on page 231-82	Computes the singular value decomposition (SVD) of a real $m$ by $n$ matrix $A$ , optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

**Table 231–6 (Cont.) UTL\_NLA Package Subprograms**

Subprogram	Description	Group
<a href="#">LAPACK_GESV Procedures</a> on page 231-85	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The LU decomposition with partial pivoting and row interchanges is used to factor $A$ .	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_GESVD Procedures</a> on page 231-87	Computes the singular value decomposition (SVD) of a real $m$ by $n$ matrix $A$ , optionally computing the left and/or right singular vectors. The SVD is written $A = U * SIGMA * transpose(V)$ .	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_GTSV Procedures</a> on page 231-93	This procedure solves the equation $a * x = b$ where $a$ is an $n$ by $n$ tridiagonal matrix, by Gaussian elimination with partial pivoting.	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_PBSV Procedures</a> on page 231-95	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite band matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The Cholesky decomposition is used to factor $A$ .	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_POSV Procedures</a> on page 231-97	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite matrix and $x$ and $b$ are $n$ by $nrhs$ matrices. The Cholesky decomposition is used to factor $A$ .	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_PPSV Procedures</a> on page 231-99	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite matrix stored in packed format and $x$ and $b$ are $n$ by $nrhs$ matrices. The Cholesky decomposition is used to factor $A$ .	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_PTSSV Procedures</a> on page 231-101	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric positive definite tridiagonal matrix, and $x$ and $b$ are $n$ by $nrhs$ matrices.	LAPACK Driver Routines (Linear Equations) Subprograms
<a href="#">LAPACK_SBEV Procedures</a> on page 231-103	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix $A$	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_SBEVD Procedures</a> on page 231-105	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ . If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_SPEV Procedures</a> on page 231-107	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ in packed storage	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms
<a href="#">LAPACK_SPEVD Procedures</a> on page 231-109	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms

**Table 231–6 (Cont.) UTL\_NLA Package Subprograms**

<b>Subprogram</b>	<b>Description</b>	<b>Group</b>
<a href="#">LAPACK_SPSV Procedures</a> on page 231-111	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric matrix stored in packed format, and $x$ and $b$ are $n$ by $nrhs$ matrices. The diagonal pivoting method is used to factor $A$ .	<a href="#">LAPACK Driver Routines (Linear Equations) Subprograms</a>
<a href="#">LAPACK_STEV Procedures</a> on page 231-113	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$	<a href="#">LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</a>
<a href="#">LAPACK_STEVD Procedures</a> on page 231-115	Computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix $A$ . If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	<a href="#">LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</a>
<a href="#">LAPACK_SYEVD Procedures</a> on page 231-119	Computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix $A$ . If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.	<a href="#">LAPACK Driver Routines (LLS and Eigenvalue Problems) Subprograms</a>
<a href="#">LAPACK_SYSV Procedures</a> on page 231-121	This procedure computes the solution to a real system of linear equations $a * x = b$ where $a$ is an $n$ by $n$ symmetric matrix, and $x$ and $b$ are $n$ by $nrhs$ matrices. The diagonal pivoting method is used to factor $A$ .	<a href="#">LAPACK Driver Routines (Linear Equations) Subprograms</a>

## BLAS\_ASUM Functions

This procedure computes the sum of the absolute values of the vector components.

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_ASUM (
  n      IN      POSITIVEN,
  x      IN      UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN)
RETURN BINARY_DOUBLE;
```

```
UTL_NLA.BLAS_ASUM (
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_DOUBLE,
  x      IN      UTL_NLA_ARRAY_FLT)
RETURN BINARY_FLOAT
```

### Parameters

**Table 231–7** *BLAS\_ASUM Function Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least ( 1 + ( n - 1 ) * abs( incx ) )
incx	Specifies the increment for the elements of x. incx must not be zero.



## BLAS\_AXPY Procedures

This procedure copies  $\alpha X + Y$  into vector  $Y$ .

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_AXPY (
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_DOUBLE,
  x      IN      UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  y      IN OUT  UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN);
```

```
UTL_NLA.BLAS_AXPY (
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_DOUBLE,
  x      IN      UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  y      IN OUT  UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN);
```

### Parameters

**Table 231–8** *BLAS\_AXPY Procedure Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors $x$ and $y$ . $n$ must be at least zero.
alpha	Specifies the scalar $\alpha$ .
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $( 1 + ( n - 1 ) * \text{abs}( \text{incx} ) )$
incx	Specifies the increment for the elements of $x$ . $\text{incx}$ must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least $( 1 + ( n - 1 ) * \text{abs}( \text{incy} ) )$
incy	Specifies the increment for the elements of $y$ . $\text{incy}$ must not be zero.

## BLAS\_COPY Procedures

This procedure copies the contents of vector X to vector Y.

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_COPY (
    n      IN      POSITIVEN,
    x      IN      UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    y      IN OUT  UTL_NLA_ARRAY_DBL,
    incy   IN      POSITIVEN);
```

```
UTL_NLA.BLAS_COPY (
    n      IN      POSITIVEN,
    x      IN      UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    y      IN OUT  UTL_NLA_ARRAY_FLT,
    incy   IN      POSITIVEN);
```

### Parameters

**Table 231–9** *BLAS\_COPY Procedure Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least ( 1 + ( n - 1 ) * abs( incx ) )
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least ( 1 + ( n - 1 ) * abs( incy ) )
incy	Specifies the increment for the elements of y. incy must not be zero.

## BLAS\_DOT Functions

This function returns the dot (scalar) product of two vectors  $X$  and  $Y$ .

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_DOT (
  n      IN   POSITIVEN,
  x      IN   UTL_NLA_ARRAY_DBL,
  incx   IN   POSITIVEN,
  y      IN   UTL_NLA_ARRAY_DBL,
  incy   IN   POSITIVEN)
RETURN BINARY_DOUBLE;
```

```
UTL_NLA.BLAS_DOT (
  n      IN   POSITIVEN,
  x      IN   UTL_NLA_ARRAY_FLT,
  incx   IN   POSITIVEN,
  y      IN   UTL_NLA_ARRAY_FLT,
  incy   IN   POSITIVEN)
RETURN BINARY_FLOAT;
```

### Parameters

**Table 231–10** *BLAS\_DOT Function Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors $x$ and $y$ . $n$ must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least ( 1 + ( n - 1 ) * abs( incx ) )
incx	Specifies the increment for the elements of $x$ . $incx$ must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least ( 1 + ( n - 1 ) * abs( incy ) )
incy	Specifies the increment for the elements of $y$ . $incy$ must not be zero.

## BLAS\_GBMV Procedures

This procedure performs one of the matrix-vector operations

$$y := \alpha * A * x + \beta * y$$

or

$$y := \alpha * A' * x + \beta * y$$

where  $\alpha$  and  $\beta$  are scalars,  $x$  and  $y$  are vectors and  $A$  is an  $m$  by  $n$  band matrix, with  $k_l$  sub-diagonals and  $k_u$  super-diagonals.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```

UTL_NLA.BLAS_GEMV (
    trans IN      flag,
    m     IN      POSITIVEN,
    n     IN      POSITIVEN,
    kl    IN      NATURALN,
    ku    IN      NATURALN,
    alpha IN      SCALAR_DOUBLE,
    a     IN      UTL_NLA_ARRAY_DBL,
    lda   IN      POSITIVEN,
    x     IN      UTL_NLA_ARRAY_DBL,
    incx  IN      POSITIVEN,
    beta  IN      SCALAR_DOUBLE,
    y     IN OUT  UTL_NLA_ARRAY_DBL,
    incy  IN      POSITIVEN,
    pack  IN      flag DEFAULT 'C');

```

```

UTL_NLA.BLAS_GEMV (
    trans IN      flag,
    m     IN      POSITIVEN,
    n     IN      POSITIVEN,
    kl    IN      NATURALN,
    ku    IN      NATURALN,
    alpha IN      SCALAR_FLOAT,
    a     IN      UTL_NLA_ARRAY_FLT,
    lda   IN      POSITIVEN,
    x     IN      UTL_NLA_ARRAY_FLT,
    incx  IN      POSITIVEN,
    beta  IN      SCALAR_FLOAT,
    y     IN OUT  UTL_NLA_ARRAY_FLT,
    incy  IN      POSITIVEN,
    pack  IN      flag DEFAULT 'C');

```

## Parameters

**Table 231–11** *BLAS\_GBMV Procedure Parameters*

Parameter	Description
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>■ trans = 'N' or 'n' := <math>\alpha * A * x + \beta * y</math></li> <li>■ trans = 'T' or 't' := <math>\alpha * A^T * x + \beta * y</math></li> <li>■ trans = 'C' or 'c' := <math>\alpha * A^H * x + \beta * y</math></li> </ul>
m	Specifies the number of rows of the matrix A. m must be at least zero.
n	Specifies the number of columns of the matrix A. n must be at least zero.
k1	Specifies the number of sub-diagonals of the matrix A. k1 must satisfy $0 \leq k1$ .
ku	Specifies the number of super-diagonals of the matrix A. ku must satisfy $0 \leq ku$ .
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). <p>Before entry, the leading <math>(k1 + ku + 1)</math> by n part of the array A must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row <math>(ku+1)</math> of the array, the first super-diagonal starting at position 2 in row ku, the first sub-diagonal starting at position 1 in row <math>(ku+2)</math>, and so on.</p> <p>Elements in the array A that do not correspond to elements in the band matrix (such as the top left ku by ku triangle) are not referenced.</p>
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least $(k1+ku+1)$ .
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n - 1) * \text{abs}(incx))$ when trans = 'N' or 'n' and at least $(1 + (m - 1) * \text{abs}(incx))$ otherwise. Before entry, the incremented array X must contain the vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (m - 1) * \text{abs}(incy))$ when trans = 'N' or 'n' and at least $(1 + (n - 1) * \text{abs}(incy))$ otherwise. Before entry with beta nonzero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.

**Table 231–11 (Cont.) BLAS\_GBMV Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"><li>▪ 'C': column-major (default)</li><li>▪ 'R': row-major</li></ul>

## BLAS\_GEMM Procedures

This procedure performs one of the matrix-matrix operations

$$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$$

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$

or

$\text{op}(X) = X'$

where  $\alpha$  and  $\beta$  are scalars, and  $A$ ,  $B$  and  $C$  are matrices, with  $\text{op}(A)$  an  $m$  by  $k$  matrix,  $\text{op}(B)$  a  $k$  by  $n$  matrix and  $C$  an  $m$  by  $n$  matrix.

**See Also:** [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#)  
on page 231-10 for other subprograms in this group

### Syntax

```

UTL_NLA.BLAS_GEMM (
  transa IN      flag,
  transb IN      flag,
  m          IN  POSITIVEN,
  n          IN  POSITIVEN,
  k          IN  POSITIVEN,
  alpha     IN   SCALAR_DOUBLE,
  a         IN   UTL_NLA_ARRAY_DBL,
  lda      IN   POSITIVEN,
  b         IN   UTL_NLA_ARRAY_DBL,
  ldb      IN   POSITIVEN,
  beta     IN   SCALAR_DOUBLE,
  c         IN OUT UTL_NLA_ARRAY_DBL,
  ldc      IN   POSITIVEN,
  pack     IN   flag DEFAULT 'C');

```

```

UTL_NLA.BLAS_GEMM (
  transa IN      flag,
  transb IN      flag,
  m          IN  POSITIVEN,
  n          IN  POSITIVEN,
  k          IN  POSITIVEN,
  alpha     IN   SCALAR_FLOAT,
  a         IN   UTL_NLA_ARRAY_FLT,
  lda      IN   POSITIVEN,
  b         IN   UTL_NLA_ARRAY_FLT,
  ldb      IN   POSITIVEN,
  beta     IN   SCALAR_FLOAT,
  c         IN OUT UTL_NLA_ARRAY_FLT,
  ldc      IN   POSITIVEN,
  pack     IN   flag DEFAULT 'C');

```

## Parameters

**Table 231–12** *BLAS\_GEMM Procedure Parameters*

Parameter	Description
transa	Specifies the form of $\text{op}(A)$ to be used in the matrix multiplication as follows: <ul style="list-style-type: none"> <li>transa = 'N' or 'n': <math>\text{op}(A) = A</math></li> <li>transa = 'T' or 't': <math>\text{op}(A) = A^T</math></li> <li>transa = 'C' or 'c': <math>\text{op}(A) = A^C</math></li> </ul>
transb	Specifies the form of $\text{op}(B)$ to be used in the matrix multiplication as follows: <ul style="list-style-type: none"> <li>transb = 'N' or 'n': <math>\text{op}(B) = B</math></li> <li>transb = 'T' or 't': <math>\text{op}(B) = B^T</math></li> <li>transb = 'C' or 'c': <math>\text{op}(B) = B^C</math></li> </ul>
m	Specifies the number of rows of the matrix $\text{op}(A)$ and of the matrix $C$ . $m$ must be at least zero.
n	Specifies the number of columns of the matrix $\text{op}(B)$ and of the matrix $C$ . $n$ must be at least zero.
k	Specifies the rows of the matrix $\text{op}(A)$ and the number of columns of the matrix $\text{op}(B)$ . $k$ must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, ka) where ka is k when transa = 'N' or 'n', and is m otherwise. Before entry with transa = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When transa = 'N' or 'n', lda must be at least $\max(1, k)$ .
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, kb) where kb is n when transb = 'N' or 'n', and is k otherwise. Before entry with transb = 'N' or 'n', the leading k by n part of the array b must contain the matrix B, otherwise the leading n by k part of the array b must contain the matrix B.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. When transb = 'N' or 'n', ldb must be at least $\max(1, n)$ .
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then c need not be set on input.
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the arrayC is overwritten by the m by n matrix $(\alpha * \text{op}(A) * \text{op}(B) + \beta * C)$ .
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least $\max(1, m)$ .
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>'C': column-major (default)</li> <li>'R': row-major</li> </ul>



## BLAS\_GEMV Procedures

This procedure performs one of the matrix-vector operations

$$y := \alpha * A * x + \beta * y$$

or

$$y := \alpha * A' * x + \beta * y$$

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_GEMV (
  trans  IN    flag,
  m      IN    POSITIVEN,
  n      IN    POSITIVEN,
  alpha  IN    SCALAR_DOUBLE,
  a      IN    UTL_NLA_ARRAY_DBL,
  lda    IN    POSITIVEN,
  x      IN    UTL_NLA_ARRAY_DBL,
  incx   IN    POSITIVEN,
  beta   IN    SCALAR_DOUBLE,
  y      IN OUT UTL_NLA_ARRAY_DBL,
  incy   IN    POSITIVEN,
  pack   IN    flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_GEMV (
  trans  IN    flag,
  m      IN    POSITIVEN,
  n      IN    POSITIVEN,
  alpha  IN    SCALAR_FLOAT,
  a      IN    UTL_NLA_ARRAY_FLT,
  lda    IN    POSITIVEN,
  x      IN    UTL_NLA_ARRAY_FLT,
  incx   IN    POSITIVEN,
  beta   IN    SCALAR_FLOAT,
  y      IN OUT UTL_NLA_ARRAY_FLT,
  incy   IN    POSITIVEN,
  pack   IN    flag DEFAULT 'C');
```

### Parameters

**Table 231–13** *BLAS\_GEMV Procedure Parameters*

Parameter	Description
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>■ trans = 'N' or 'n', <math>y := \alpha * A * x + \beta * y</math></li> <li>■ trans = 'T' or 't', <math>y := \alpha * A' * x + \beta * y</math></li> <li>■ trans = 'C' or 'c', <math>y := \alpha * A * x + \beta * y</math></li> </ul>
m	Specifies the number of rows of the matrix A. m must be at least zero.

**Table 231–13 (Cont.) BLAS\_GEMV Procedure Parameters**

Parameter	Description
n	Specifies the number of columns of the matrix A. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry, the leading m by n part of the array a must contain the matrix of coefficients.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1, m).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n - 1) * \text{abs}(\text{incx}))$ when trans = 'N' or 'n' and at least $(1 + (m - 1) * \text{abs}(\text{incx}))$ otherwise. Before entry, the incremented array X must contain the vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.
y	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (m - 1) * \text{abs}(\text{incy}))$ when trans = 'N' or 'n' and at least $(1 + (n - 1) * \text{abs}(\text{incy}))$ otherwise. Before entry with beta nonzero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
incy	Specifies the increment for the elements of y. Must not be zero.
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_GER Procedures

This procedure performs the rank 1 operation

$$A := \alpha * x * y' + A$$

where  $\alpha$  is a scalar,  $x$  is an  $m$  element vector,  $y$  is an  $n$  element vector and  $A$  is an  $m$  by  $n$  matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_GER (
  m      IN      POSITIVEN,
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_DBL,
  x      IN OUT  UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  y      IN      UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_GER (
  m      IN      POSITIVEN,
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_FLT,
  x      IN OUT  UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  y      IN      UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–14** *BLAS\_GER Procedure Parameters*

Parameter	Description
$m$	Specifies the number of rows of the matrix $A$ . $m$ must be at least zero.
$n$	Specifies the number of columns of the matrix $A$ . $n$ must be at least zero.
$\alpha$	Specifies the scalar $\alpha$ .
$x$	UTL_NLA_ARRAY_FLT/DBL of dimension at least $( 1 + ( m - 1 ) * \text{abs}( \text{incx} ) )$ Before entry, the incremented array $X$ must contain the $m$ element vector $x$ .
$\text{incx}$	Specifies the increment for the elements of $x$ . $\text{incx}$ must not be zero.

**Table 231–14 (Cont.) BLAS\_GER Procedure Parameters**

Parameter	Description
y	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least  <math>( 1 + ( n - 1 ) * \text{abs}( \text{incy} ) )</math></p> <p>Before entry, the incremented array Y must contain the m element vector y.</p>
incy	<p>Specifies the increment for the elements of y. incx must not be zero.</p>
a	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n).</p> <p>Before entry, the leading m by n part of the array a must contain the matrix of coefficients. On exit, a is overwritten by the updated matrix.</p>
lda	<p>Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least  <math>\text{max}( 1, m )</math></p>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_IAMAX Functions

This function computes the index of first element of a vector that has the largest absolute value.

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_IAMAX (
  n      IN  POSITIVEN,
  x      IN  UTL_NLA_ARRAY_DBL,
  incx   IN  POSITIVEN,
  RETURN POSITIVEN;
```

```
UTL_NLA.BLAS_IAMAX (
  n      IN  POSITIVEN,
  x      IN  UTL_NLA_ARRAY_FLT,
  incx   IN  POSITIVEN,
  RETURN POSITIVEN;
```

### Parameters

**Table 231–15** *BLAS\_IAMAX Function Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least ( 1 + ( n - 1 ) * abs( incx ) )
incx	Specifies the increment for the elements of x. incx must not be zero.

## BLAS\_NRM2 Functions

This function computes the vector 2-norm (Euclidean norm).

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_NRM2 (
    n      IN      POSITIVEN,
    x      IN      UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN)
RETURN BINARY_DOUBLE;
```

```
UTL_NLA.BLAS_NRM2 (
    n      IN      POSITIVEN,
    x      IN      UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN)
RETURN BINARY_FLOAT;
```

### Parameters

**Table 231–16** *BLAS\_NRM2 Function Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least ( 1 + ( n - 1 ) * abs ( incx ) )
incx	Specifies the increment for the elements of x. incx must not be zero.

## BLAS\_ROT Procedures

This procedure returns the plane rotation of points.

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_ROT (
  n      IN      POSITIVEN,
  x      IN OUT  UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  y      IN OUT  UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN,
  c      IN      SCALAR_DOUBLE,
  s      IN      SCALAR_DOUBLE) ;
```

```
UTL_NLA.BLAS_ROT (
  n      IN      POSITIVEN,
  x      IN OUT  UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  y      IN OUT  UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN,
  c      IN      SCALAR_DOUBLE,
  s      IN      SCALAR_DOUBLE) ;
```

### Parameters

**Table 231–17** *BLAS\_ROT Procedure Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1+(n-1) * \text{abs}(\text{incx}))$
incx	Specifies the increment for the elements of x. incx must not be zero.
y	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least $(1+(n-1) * \text{abs}(\text{incy}))$
incy	Specifies the increment for the elements of y. incy must not be zero.
c	SCALAR_FLOAT/DOUBLE.Specifies the scalar C.
s	SCALAR_FLOAT/DOUBLE.Specifies the scalar S.

## BLAS\_ROTG Procedures

This procedure returns the Givens rotation of points.

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_ROTG (
  a  IN OUT  SCALAR_DOUBLE,
  b  IN OUT  SCALAR_DOUBLE,
  c  IN OUT  SCALAR_DOUBLE,
  s  IN OUT  SCALAR_DOUBLE);
```

```
UTL_NLA.BLAS_ROTG (
  a  IN OUT  SCALAR_FLOAT,
  b  IN OUT  SCALAR_FLOAT,
  c  IN OUT  SCALAR_FLOAT,
  s  IN OUT  SCALAR_FLOAT);
```

### Parameters

**Table 231–18** *BLAS\_ROT G Procedure Parameters*

Parameter	Description
a	SCALAR_FLOAT/DOUBLE. Specifies the scalar A.
b	SCALAR_FLOAT/DOUBLE. Specifies the scalar B.
c	SCALAR_FLOAT/DOUBLE. Specifies the scalar C.
s	SCALAR_FLOAT/DOUBLE. Specifies the scalar S.



## BLAS\_SCAL Procedures

This procedure scales a vector by a constant.

**See Also:** [BLAS Level 1 \(Vector-Vector Operations\) Subprograms](#) on page 231-7 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SCAL (
  n      IN  POSITIVEN,
  alpha  IN  SCALAR_DOUBLE,
  x      IN  OUT UTL_NLA_ARRAY_DBL,
  incx   IN  POSITIVEN);
```

```
UTL_NLA.BLAS_SCAL (
  n      IN  POSITIVEN,
  alpha  IN  SCALAR_FLOAT,
  x      IN  OUT UTL_NLA_ARRAY_FLT,
  incx   IN  POSITIVEN);
```

### Parameters

**Table 231–19** *BLAS\_SCAL Procedure Parameters*

Parameter	Description
n	Specifies the number of elements of the vectors x and y. n must be at least zero.
alpha	Specifies the scalar alpha.
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incx}))$
incx	Specifies the increment for the elements of x. incx must not be zero.

## BLAS\_SPMV Procedures

This procedure performs the matrix-vector operation

$$y := \alpha * A * x + \beta * y$$

where  $\alpha$  and  $\beta$  are scalars,  $x$  and  $y$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SPMV (
    uplo    IN        flag,
    n       IN        POSITIVEN,
    alpha   IN        SCALAR_DOUBLE,
    ap      IN        UTL_NLA_ARRAY_DBL,
    x       IN        UTL_NLA_ARRAY_DBL,
    incx    IN        POSITIVEN,
    beta    IN        SCALAR_DOUBLE,
    y       IN OUT    UTL_NLA_ARRAY_DBL,
    incy    IN        POSITIVEN,
    pack    IN        flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SPMV (
    uplo    IN        flag,
    n       IN        POSITIVEN,
    alpha   IN        SCALAR_FLOAT,
    ap      IN        UTL_NLA_ARRAY_FLT,
    x       IN        UTL_NLA_ARRAY_FLT,
    incx    IN        POSITIVEN,
    beta    IN        SCALAR_FLOAT,
    y       IN OUT    UTL_NLA_ARRAY_FLT,
    incy    IN        POSITIVEN,
    pack    IN        flag DEFAULT 'C');
```

### Parameters

**Table 231–20** *BLAS\_SPMV Procedure Parameters*

Parameter	Description
uplo	Specifies the upper or lower triangular part of the matrix $A$ is supplied in the packed array $AP$ : <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u'. The upper triangular part of <math>A</math> is supplied in <math>AP</math>.</li> <li>■ uplo = 'L' or 'l'. The lower triangular part of <math>A</math> is supplied in <math>AP</math>.</li> </ul>
n	Specifies the order of the matrix $A$ . $n$ must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar $\alpha$ .

**Table 231–20 (Cont.) BLAS\_SPMV Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>((n*(n+1))/2)</math></p> <p>Before entry with <code>uplo = 'U'</code> or <code>'u'</code>, the array <code>ap</code> must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(1,1)</code>, <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(1,2)</code> and <code>a(2,2)</code> respectively, and so on.</p> <p>Before entry with <code>uplo = 'L'</code> or <code>'l'</code>, the array <code>ap</code> must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(2,1)</code> and <code>ap(3)</code> contain <code>a(3,1)</code> and <code>a(3,2)</code> respectively, and so on.</p>
x	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>(1+(n-1)*abs(incx))</math></p> <p>Before entry, the incremented array <code>X</code> must contain the <code>n</code> element vector <code>x</code>.</p>
incx	Specifies the increment for the elements of <code>x</code> . Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar <code>beta</code> . When <code>beta</code> is supplied as zero then <code>Y</code> need not be set on input.
y	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>(1+(n-1)*abs(incy))</math></p> <p>Before entry, the incremented array <code>Y</code> must contain the <code>n</code> element vector <code>y</code>. On exit, <code>Y</code> is overwritten by the updated vector <code>y</code>.</p>
incy	Specifies the increment for the elements of <code>y</code> . Must not be zero.
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SPR Procedures

This procedure performs the rank 1 operation

$A := \alpha * x * x' + A$

where  $\alpha$  is a real scalar,  $x$  is an  $n$  element vector, and  $A$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SPR (
    uplo   IN      flag,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DBL,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    ap     IN OUT  UTL_NLA_ARRAY_DBL,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SPR (
    uplo   IN      flag,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLT,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    ap     IN OUT  UTL_NLA_ARRAY_FLT,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–21** *BLAS\_SPR Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the matrix $A$ is supplied in the packed array $ap$ : <ul style="list-style-type: none"> <li>uplo = 'U' or 'u': The upper triangular part of <math>A</math> is supplied in <math>ap</math>.</li> <li>uplo = 'L' or 'l': The lower triangular part of <math>A</math> is supplied in <math>ap</math>.</li> </ul>
n	Specifies the order of the matrix $A$ . $n$ must be at least zero.
alpha	Specifies the scalar $\alpha$ .
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incx}))$ Before entry, the incremented array $X$ must contain the $m$ element vector $x$ .
incx	Specifies the increment for the elements of $x$ . $\text{incx}$ must not be zero.

**Table 231–21 (Cont.) BLAS\_SPR Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>((n * (n + 1)) / 2)</math></p> <p>Before entry with <code>uplo = 'U'</code> or <code>'u'</code>, the array <code>ap</code> must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(1,1)</code>, <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(1,2)</code> and <code>a(2,2)</code> respectively, and so on. On exit, the array <code>ap</code> is overwritten by the upper triangular part of the updated matrix.</p> <p>Before entry with <code>uplo = 'L'</code> or <code>'l'</code>, the array <code>ap</code> must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(1,1)</code>, <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(2,1)</code> and <code>a(3,1)</code> respectively, and so on. On exit, the array <code>ap</code> is overwritten by the lower triangular part of the updated matrix</p>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SPR2 Procedures

This procedure performs the rank 2 operation

$$A := \alpha * x * y' + \alpha * y * x' + A$$

where  $\alpha$  is a scalar,  $x$  and  $y$  are  $n$  element vectors, and  $A$  is an  $n$  by  $n$  symmetric matrix, supplied in packed form.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```

UTL_NLA.BLAS_SPR2 (
    uplo   IN      flag,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DBL,
    x      IN      UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    y      IN      UTL_NLA_ARRAY_DBL,
    incy   IN      POSITIVEN,
    a      IN OUT  UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
    
```

```

UTL_NLA.BLAS_SPR2 (
    uplo   IN      flag,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLT,
    x      IN      UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    y      IN      UTL_NLA_ARRAY_FLT,
    incy   IN      POSITIVEN,
    a      IN OUT  UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
    
```

### Parameters

**Table 231–22** *BLAS\_SPR2 Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the matrix $A$ is supplied in the packed array $ap$ : <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u': The upper triangular part of <math>A</math> is supplied in <math>ap</math>.</li> <li>■ uplo = 'L' or 'l': The lower triangular part of <math>A</math> is supplied in <math>ap</math>.</li> </ul>
n	Specifies the order of the matrix $A$ . $n$ must be at least zero.
alpha	Specifies the scalar $\alpha$ .
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incx}))$ Before entry, the incremented array $X$ must contain the $m$ element vector $x$ .

**Table 231–22 (Cont.) BLAS\_SPR2 Procedure Parameters**

Parameter	Description
<code>incx</code>	Specifies the increment for the elements of <code>x</code> . <code>incx</code> must not be zero.
<code>y</code>	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incy}))$ Before entry, the incremented array <code>X</code> must contain the <code>m</code> element vector <code>y</code> .
<code>incy</code>	Specifies the increment for the elements of <code>y</code> . <code>incy</code> must not be zero.
<code>ap</code>	UTL_NLA_ARRAY_FLT/DBL of dimension at least $((n * (n+1)) / 2)$ Before entry with <code>uplo = 'U'</code> or <code>'u'</code> , the array <code>ap</code> must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>ap(1)</code> contains <code>a(1,1)</code> , <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(1,2)</code> and <code>a(2,2)</code> respectively, and so on. On exit, the array <code>ap</code> is overwritten by the upper triangular part of the updated matrix. Before entry with <code>uplo = 'L'</code> or <code>'l'</code> , the array <code>ap</code> must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(1,1)</code> , <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(2,1)</code> and <code>a(3,1)</code> respectively, and so on. On exit, the array <code>ap</code> is overwritten by the lower triangular part of the updated matrix
<code>lda</code>	Specifies the first dimension of <code>a</code> as declared in the calling (sub) program. <code>lda</code> must be at least $(k + 1)$ .
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SBMV Procedures

This procedure performs the matrix-vector operation

$$y := \alpha * A * x + \beta * y$$

where `alpha` and `beta` are scalars, `x` and `y` are `n` element vectors and `A` is an `n` by `n` symmetric band matrix, with `k` super-diagonals.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SBMV (
    uplo    IN        flag,
    n       IN        POSITIVEN,
    k       IN        NATURALN,
    alpha   IN        SCALAR_DOUBLE,
    a       IN        UTL_NLA_ARRAY_DBL,
    lda     IN        POSITIVEN,
    x       IN        UTL_NLA_ARRAY_DBL,
    incx    IN        POSITIVEN,
    beta    IN        SCALAR_DOUBLE,
    y       IN OUT    UTL_NLA_ARRAY_DBL,
    incy    IN        POSITIVEN,
    pack    IN        flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SBMV (
    uplo    IN        flag,
    n       IN        POSITIVEN,
    k       IN        NATURALN,
    alpha   IN        SCALAR_FLOAT,
    a       IN        UTL_NLA_ARRAY_FLT,
    lda     IN        POSITIVEN,
    x       IN        UTL_NLA_ARRAY_FLT,
    incx    IN        POSITIVEN,
    beta    IN        SCALAR_FLOAT,
    y       IN OUT    UTL_NLA_ARRAY_FLT,
    incy    IN        POSITIVEN,
    pack    IN        flag DEFAULT 'C');
```

### Parameters

**Table 231–23** *BLAS\_SBMV Procedure Parameters*

Parameter	Description
<code>uplo</code>	Specifies whether the upper or lower triangular part of the band matrix <code>A</code> is being supplied: <ul style="list-style-type: none"> <li>▪ <code>uplo = 'U'</code> or <code>'u'</code>. The upper triangular part of <code>A</code> is supplied.</li> <li>▪ <code>uplo = 'L'</code> or <code>'l'</code>. The lower triangular part of <code>A</code> is supplied.</li> </ul>
<code>n</code>	Specifies the order of the matrix <code>A</code> . <code>n</code> must be at least zero.
<code>k</code>	Specifies the number of super-diagonals of the matrix <code>A</code> . <code>k</code> must satisfy <code>0 ≤ k</code> .



**Table 231–23 (Cont.) BLAS\_SBMV Procedure Parameters**

Parameter	Description
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n).</p> <p>Before entry with <code>uplo = 'U' or 'u'</code>, the leading (k+1) by n part of the array A must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced.</p> <p>Before entry with <code>uplo = 'L' or 'l'</code>, the leading (k+1) by n part of the array A must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced.</p> <p>Unchanged on exit</p>
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k + 1).
x	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incx))</p> <p>Before entry, the incremented array X must contain the n element vector x.</p>
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.
y	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least (1+(n-1)*abs(incy))</p> <p>Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.</p>
incy	Specifies the increment for the elements of y. Must not be zero.
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SWAP Procedures

This procedure swaps the contents of two vectors each of size  $n$ .

### Syntax

```
UTL_NLA.BLAS_SWAP (
  n      IN      POSITIVEN,
  x      IN OUT  UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  y      IN OUT  UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN);
```

```
UTL_NLA.BLAS_SWAP (
  n      IN      POSITIVEN,
  x      IN OUT  UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  y      IN OUT  UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN);
```

### Parameters

**Table 231–24** *BLAS\_SWAP Procedure Parameters*

Parameter	Description
$n$	Specifies the number of elements of the vectors $x$ and $y$ . $n$ must be at least zero.
$x$	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incx}))$
$\text{incx}$	Specifies the increment for the elements of $x$ . $\text{incx}$ must not be zero.
$y$	UTL_NLA_ARRAY_FLT/DBL of DIMENSION at least $(1 + (n-1) * \text{abs}(\text{incy}))$
$\text{incy}$	Specifies the increment for the elements of $y$ . $\text{incy}$ must not be zero.

## BLAS\_SYMM Procedures

This procedure performs one of the matrix-matrix operations

```
C := alpha*A*B + beta*C
```

or

```
C := alpha*B*A + beta*C
```

where alpha and beta are scalars, A is a symmetric matrix, and B and C are m by n matrices.

**See Also:** [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#) on page 231-10 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SYMM (
    side   IN      flag,
    uplo   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DOUBLE,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    b      IN      UTL_NLA_ARRAY_DBL,
    ldb    IN      POSITIVEN,
    beta   IN      SCALAR_DOUBLE,
    c      IN OUT  UTL_NLA_ARRAY_DBL,
    ldc    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SYMM (
    side   IN      flag,
    uplo   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLOAT,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    b      IN      UTL_NLA_ARRAY_FLT,
    ldb    IN      POSITIVEN,
    beta   IN      SCALAR_FLOAT,
    c      IN OUT  UTL_NLA_ARRAY_FLT,
    ldc    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–25** *BLAS\_SYMM Procedure Parameters*

Parameter	Description
side	Specifies whether the symmetric matrix A appears on the left or right in the operation: <ul style="list-style-type: none"> <li>■ side = 'L' or 'l' : C := alpha*A*B + beta*C</li> <li>■ side = 'R' or 'r' : C := alpha*B*A + beta*C</li> </ul>

**Table 231–25 (Cont.) BLAS\_SYMM Procedure Parameters**

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array A is to be referenced: <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u' : Only the upper triangular part of the symmetric matrix is to be referenced.</li> <li>■ uplo = 'L' or 'l' : Only the lower triangular part of the symmetric matrix is to be referenced.</li> </ul>
m	Specifies the number of rows of the matrix C. m must be at least zero.
n	Specifies the number of columns of the matrix C. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, ka) where ka is m when side = 'L' or 'l', and is n otherwise.  Before entry with side = 'L' or 'l', the leading m by m part of the array A must contain the symmetric matrix, such that when uplo = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when uplo = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.  Before entry with side = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least $\max(1, m)$ , otherwise lda must be at least $\max(1, n)$ .
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb, n).  Before entry, the leading m by n part of the array B must contain the matrix B.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least $\max(1, m)$ .
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then c need not be set on input.
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n). Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n updated matrix.
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least $\max(1, m)$ .
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SYMV Procedures

This procedure performs the matrix-vector operation

$$y := \alpha * A * x + \beta * y$$

where  $\alpha$  and  $\beta$  are scalars,  $x$  and  $y$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  symmetric matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SYMV (
  uplo   IN      flag,
  n       IN      POSITIVEN,
  alpha  IN      SCALAR_DOUBLE,
  a       IN      UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  x       IN      UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  beta   IN      SCALAR_DOUBLE,
  y       IN OUT  UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SYMV (
  uplo   IN      flag,
  n       IN      POSITIVEN,
  alpha  IN      SCALAR_FLOAT,
  a       IN      UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  x       IN      UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  beta   IN      SCALAR_FLOAT,
  y       IN OUT  UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–26** *BLAS\_SYMV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array $A$ is to be referenced: <ul style="list-style-type: none"> <li>▪ uplo = 'U' or 'u'. Only the upper triangular part of <math>A</math> is to be referenced.</li> <li>▪ uplo = 'L' or 'l'. Only the lower triangular part of <math>A</math> is to be referenced.</li> </ul>
n	Specifies the order of the matrix $A$ . $n$ must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar $\alpha$ .

**Table 231–26 (Cont.) BLAS\_SYMV Procedure Parameters**

Parameter	Description
a	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced.</p> <p>Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.</p>
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1, n).
x	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>(1 + (n-1) * \text{abs}(\text{incx}))</math></p> <p>Before entry, the incremented array X must contain the n element vector x.</p>
incx	Specifies the increment for the elements of x. Must not be zero.
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta. When beta is supplied as zero then y need not be set on input.
y	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>(1 + (n-1) * \text{abs}(\text{incy}))</math></p> <p>Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.</p>
incy	Specifies the increment for the elements of y. Must not be zero.
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SYR Procedures

This procedure performs the rank 1 operation

$$A := \text{alpha} * x * x' + A$$

where  $\text{alpha}$  is a real scalar,  $x$  is an  $n$  element vector, and  $A$  is an  $n$  by  $n$  symmetric matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SYR (
  uplo   IN      flag,
  n       IN      POSITIVEN,
  alpha  IN      SCALAR_DBL,
  x       IN OUT  UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  a       IN OUT  UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SYR (
  uplo   IN      flag,
  n       IN      POSITIVEN,
  alpha  IN      SCALAR_FLT,
  x       IN OUT  UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  a       IN OUT  UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231-27** *BLAS\_SYR Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array $A$ is to be referenced: <ul style="list-style-type: none"> <li>▪ <code>uplo = 'U'</code> or <code>'u'</code>: Only the upper triangular part of <math>A</math> is to be referenced.</li> <li>▪ <code>uplo = 'L'</code> or <code>'l'</code>: Only the lower triangular part of <math>A</math> is to be referenced.</li> </ul>
n	Specifies the order of the matrix $A$ . $n$ must be at least zero.
alpha	Specifies the scalar $\text{alpha}$ .
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incx}))$ Before entry, the incremented array $x$ must contain the $m$ element vector $x$ .
incx	Specifies the increment for the elements of $x$ . $\text{incx}$ must not be zero.

**Table 231–27 (Cont.) BLAS\_SYR Procedure Parameters**

Parameter	Description
a	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n)</p> <p>Before entry with <code>uplo = 'U' or 'u'</code>, the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced. On exit, the upper triangular part of the array A is overwritten by the upper triangular part of the updated matrix.</p> <p>Before entry with <code>uplo = 'L' or 'l'</code>, the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced. On exit, the lower triangular part of the array A is overwritten by the lower triangular part of the updated matrix.</p>
lda	<p>Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least</p> <p><code>max( 1, n )</code></p>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>



## BLAS\_SYR2 Procedures

This procedure performs the rank 2 operation

$$A := \alpha * x * y' + \alpha * y * x' + A$$

where  $\alpha$  is a scalar,  $x$  and  $y$  are  $n$  element vectors, and  $A$  is an  $n$  by  $n$  symmetric matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_SYR2 (
  uplo   IN      flag,
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_DBL,
  x      IN      UTL_NLA_ARRAY_DBL,
  incx   IN      POSITIVEN,
  y      IN      UTL_NLA_ARRAY_DBL,
  incy   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_SYR2 (
  uplo   IN      flag,
  n      IN      POSITIVEN,
  alpha  IN      SCALAR_FLT,
  x      IN      UTL_NLA_ARRAY_FLT,
  incx   IN      POSITIVEN,
  y      IN      UTL_NLA_ARRAY_FLT,
  incy   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–28** *BLAS\_SYR2 Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array $A$ is to be referenced: <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u' : Only the upper triangular part of <math>A</math> is to be referenced.</li> <li>■ uplo = 'L' or 'l' : Only the lower triangular part of <math>A</math> is to be referenced.</li> </ul>
n	Specifies the order of the matrix $A$ . $n$ must be at least zero.
alpha	Specifies the scalar $\alpha$ .
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $( 1 + ( n - 1 ) * \text{abs}( \text{incx} ) )$ Before entry, the incremented array $X$ must contain the $m$ element vector $x$ .

**Table 231–28 (Cont.) BLAS\_SYR2 Procedure Parameters**

Parameter	Description
<code>incx</code>	Specifies the increment for the elements of <code>x</code> . <code>incx</code> must not be zero.
<code>y</code>	UTL_NLA_ARRAY_FLT/DBL of dimension at least $( 1 + ( n - 1 ) * \text{abs}( \text{incy} ) )$ Before entry, the incremented array <code>Y</code> must contain the <code>m</code> element vector <code>y</code> .
<code>incy</code>	Specifies the increment for the elements of <code>y</code> . <code>incy</code> must not be zero.
<code>a</code>	UTL_NLA_ARRAY_FLT/DBL of DIMENSION ( <code>lda</code> , <code>n</code> ) With <code>uplo = 'U'</code> or <code>'u'</code> , the leading <code>n</code> by <code>n</code> upper triangular part of the array <code>A</code> must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of <code>A</code> is not referenced. On exit, the upper triangular part of the array <code>A</code> is overwritten by the upper triangular part of the updated matrix. With <code>uplo = 'L'</code> or <code>'l'</code> , the leading <code>n</code> by <code>n</code> lower triangular part of the array <code>A</code> must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of <code>A</code> is not referenced. On exit, the lower triangular part of the array <code>A</code> is overwritten by the lower triangular part of the updated matrix.
<code>lda</code>	Specifies the first dimension of <code>a</code> as declared in the calling (sub) program. <code>lda</code> must be at least $\text{max}( 1, n )$
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SYR2K Procedures

This procedure performs one of the symmetric rank2 k operations

$$C := \alpha * A * B' + \alpha * B * A' + \beta * C$$

or

$$C := \alpha * A' * B + \alpha * B' * A + \beta * C$$

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

**See Also:** [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#) on page 231-10 for other subprograms in this group

### Syntax

```

UTL_NLA.BLAS_SYR2K (
  uplo   IN      flag,
  trans  IN      flag,
  n      IN      POSITIVEN,
  k      IN      POSITIVEN,
  alpha  IN      SCALAR_DOUBLE,
  a      IN      UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  b      IN      UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  beta   IN      SCALAR_DOUBLE,
  c      IN OUT  UTL_NLA_ARRAY_DBL,
  ldc    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');

```

```

UTL_NLA.BLAS_SYR2K (
  uplo   IN      flag,
  trans  IN      flag,
  n      IN      POSITIVEN,
  k      IN      POSITIVEN,
  alpha  IN      SCALAR_FLOAT,
  a      IN      UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  b      IN OUT  UTL_NLA_ARRAY_FLT,
  ldb    IN      POSITIVEN,
  beta   IN      SCALAR_FLOAT,
  c      IN OUT  UTL_NLA_ARRAY_FLT,
  ldc    IN      POSITIVEN,
  pack   IN      flag DEFAULT 'C');

```

## Parameters

**Table 231–29** *BLAS\_SYR2K Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array C is to be referenced: <ul style="list-style-type: none"> <li>uplo = 'U' or 'u' : Only the upper triangular part of C is to be referenced.</li> <li>uplo = 'L' or 'l' : Only the lower triangular part of C is to be referenced.</li> </ul>
trans	Specifies the operations to be performed: <ul style="list-style-type: none"> <li>trans = 'N' or 'n': <math>C := \alpha * A * B' + \alpha * B * A' + \beta * C</math></li> <li>trans = 'T' or 't': <math>C := \alpha * A' * B + \alpha * B' * A + \beta * C</math></li> <li>trans = 'C' or 'c': <math>C := \alpha * A' * B + \alpha * B' * A + \beta * C</math></li> </ul>
n	Specifies the order of matrix C. n must be at least zero.
k	On entry with trans = 'N' or 'n', k specifies the number of columns of the matrices A and B. On entry with trans = 'T' or 't' or trans = 'C' or 'c', k specifies the number of rows of the matrices A and B. k must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, ka) where kb is k when trans = 'N' or 'n', and is n otherwise.  Before entry with trans = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When trans = 'N' or 'n', lda must be at least $\max(1, n)$ , otherwise lda must be at least $\max(1, k)$ .
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, kb) where kb is k when trans = 'N' or 'n', and is n otherwise.  Before entry with trans = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. When trans = 'N' or 'n', ldb must be at least $\max(1, n)$ , otherwise ldb must be at least $\max(1, k)$ .
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.

**Table 231–29 (Cont.) BLAS\_SYR2K Procedure Parameters**

Parameter	Description
c	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n).</p> <p>Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.</p> <p>Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.</p>
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least $\max(1, n)$ .
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_SYRK Procedures

This procedure performs one of the symmetric rank k operations

$$C := \alpha * A * A' + \beta * C$$

or

$$C := \alpha * A' * A + \beta * C$$

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

**See Also:** [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#) on page 231-10 for other subprograms in this group

### Syntax

```

UTL_NLA.BLAS_SYRK (
    uplo   IN        flag,
    trans  IN        flag,
    n      IN        POSITIVEN,
    k      IN        POSITIVEN,
    alpha  IN        SCALAR_DOUBLE,
    a      IN        UTL_NLA_ARRAY_DBL,
    lda    IN        POSITIVEN,
    beta   IN        SCALAR_DOUBLE,
    c      IN OUT    UTL_NLA_ARRAY_DBL,
    ldc    IN        POSITIVEN,
    pack   IN        flag DEFAULT 'C');

```

```

UTL_NLA.BLAS_SYRK (
    uplo   IN        flag,
    trans  IN        flag,
    n      IN        POSITIVEN,
    k      IN        POSITIVEN,
    alpha  IN        SCALAR_FLOAT,
    a      IN        UTL_NLA_ARRAY_FLT,
    lda    IN        POSITIVEN,
    beta   IN        SCALAR_FLOAT,
    c      IN OUT    UTL_NLA_ARRAY_DBL,
    ldc    IN        POSITIVEN,
    pack   IN        flag DEFAULT 'C');

```

### Parameters

**Table 231–30** *BLAS\_SYRK Procedure Parameters*

Parameter	Description
uplo	Specifies whether the upper or lower triangular part of the array C is to be referenced: <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u' : Only the upper triangular part of C is to be referenced.</li> <li>■ uplo = 'L' or 'l' : Only the lower triangular part of C is to be referenced.</li> </ul>

**Table 231–30 (Cont.) BLAS\_SYRK Procedure Parameters**

Parameter	Description
trans	Specifies the operations to be performed: <ul style="list-style-type: none"> <li>■ trans = 'N' or 'n' : <math>C := \alpha * A * A' + \beta * C</math></li> <li>■ trans = 'T' or 't' : <math>C := \alpha * A' * A + \beta * C</math></li> <li>■ trans = 'C' or 'c' : <math>C := \alpha * A' * A + \beta * C</math></li> </ul>
n	Specifies the order of matrix C. n must be at least zero.
k	On entry with trans = 'N' or 'n', k specifies the number of columns of the matrix A. On entry with trans = 'T' or 't' or trans = 'C' or 'c', k specifies the number of rows of the matrix A. k must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, ka) where ka is k when trans = 'N' or 'n', and is n otherwise.  Before entry with trans = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When trans = 'N' or 'n', lda must be at least $\max(1, n)$ , otherwise lda must be at least $\max(1, k)$ .
beta	SCALAR_FLOAT/DOUBLE. Specifies the scalar beta.
c	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldc, n).  Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.  Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.
ldc	Specifies the first dimension of C as declared in the calling (sub) program. ldc must be at least $\max(1, n)$ .
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_TBMV Procedures

This procedure performs the matrix-vector operations

```
x := A*x
```

or

```
x := A'*x
```

where  $x$  is an  $n$  element vector and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular band matrix, with  $(k+1)$  diagonals.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TBMV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    k      IN      NATURALN,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TBMV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    k      IN      NATURALN,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–31** *BLAS\_TBMV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u'. A is an upper triangular matrix.</li> <li>■ uplo = 'L' or 'l'. A is a lower triangular matrix.</li> </ul>
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>■ trans = 'N' or 'n' <math>x := A*x</math></li> <li>■ trans = 'T' or 't' <math>x := A'*x</math></li> <li>■ trans = 'C' or 'c' <math>x := A'*x</math></li> </ul>



**Table 231–31 (Cont.) BLAS\_TBMV Procedure Parameters**

Parameter	Description
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>■ diag = 'U' or 'u'. A is assumed to be unit triangular.</li> <li>■ diag = 'N' or 'n'. A is not assumed to be unit triangular.</li> </ul>
n	Specifies the order of the matrix A. n must be at least zero.
k	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>■ with uplo = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.</li> <li>■ with uplo = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A.</li> </ul> K must satisfy $0 \leq k$ .
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION ( lda, n ). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least (k+1).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n-1) * \text{abs}(\text{incx}))$ . Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_TBSV Procedures

This procedure solves one of the systems of equations

$$A*x = b$$

or

$$A'*x = b$$

where  $b$  and  $x$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular band matrix, with  $(k+1)$  diagonals.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```

UTL_NLA.BLAS_TBSV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    k      IN      NATURALN,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
    
```

```

UTL_NLA.BLAS_STBSV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    k      IN      NATURALN,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
    
```

### Parameters

**Table 231–32** *BLAS\_TBSV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: <ul style="list-style-type: none"> <li>uplo = 'U' or 'u'. A is an upper triangular matrix.</li> <li>uplo = 'L' or 'l'. A is a lower triangular matrix.</li> </ul>
trans	Specifies the equations to be solved: <ul style="list-style-type: none"> <li>trans = 'N' or 'n': <math>A*x = b</math></li> <li>trans = 'T' or 't': <math>A'*x = b</math></li> <li>trans = 'C' or 'c': <math>A'*x = b</math></li> </ul>

**Table 231–32 (Cont.) BLAS\_TBSV Procedure Parameters**

Parameter	Description
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>diag = 'U' or 'u' : A is assumed to be unit triangular.</li> <li>diag = 'N' or 'n' : A is not assumed to be unit triangular.</li> </ul>
n	Specifies the order of the matrix A. n must be at least zero.
k	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>with uplo = 'U' or 'u', K specifies the number of super-diagonals of the matrix A.</li> <li>with uplo = 'L' or 'l', K specifies the number of sub-diagonals of the matrix A.</li> </ul> K must satisfy $0 \leq k$ .
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced. Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced. Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.
lda	On entry, lda specifies the first dimension of A as declared in the calling (sub) program. lda must be at least (k+1).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $( 1 + ( n - 1 ) * \text{abs}( \text{incx} ) )$ Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.
incx	Specifies the increment for the elements of x. incx must not be zero.
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>'C': column-major (default)</li> <li>'R': row-major</li> </ul>

## Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

## BLAS\_TPMV Procedures

This procedure performs the matrix-vector operations

```
x := A*x
```

or

```
x := A'*x
```

where  $x$  is an  $n$  element vector and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TPMV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    ap     IN      UTL_NLA_ARRAY_DBL,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TBMV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    ap     IN      UTL_NLA_ARRAY_FLT,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–33** *BLAS\_TPMV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u'. A is an upper triangular matrix.</li> <li>■ uplo = 'L' or 'l'. A is a lower triangular matrix.</li> </ul>
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>■ trans = 'N' or 'n'. <math>x := A*x</math></li> <li>■ trans = 'T' or 't'. <math>x := A'*x</math></li> <li>■ trans = 'C' or 'c'. <math>x := A'*x</math></li> </ul>
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>■ diag = 'U' or 'u'. A is assumed to be unit triangular.</li> <li>■ diag = 'N' or 'n'. A is not assumed to be unit triangular.</li> </ul>
n	Specifies the order of the matrix A. n must be at least zero.

**Table 231–33 (Cont.) BLAS\_TPMV Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n).</p> <p>Before entry with uplo = 'U' or 'u', the leading (k+1) by n part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row k, and so on. The top left k by k triangle of the array A is not referenced.</p> <p>Before entry with uplo = 'L' or 'l', the leading (k+1) by n part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right k by k triangle of the array A is not referenced.</p> <p>Note that when diag = 'U' or 'u', the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity.</p>
x	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>(1 + (n-1) * \text{abs}(\text{incx}))</math>. Before entry, the incremented array X must contain the n element vector x. On exit, X is overwritten with the transformed vector x.</p>
incx	Specifies the increment for the elements of x. Must not be zero.
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_TPSV Procedures

This procedure solves one of the systems of equations

$$A*x = b$$

or

$$A'*x = b$$

where  $b$  and  $x$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TPSV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    ap     IN      UTL_NLA_ARRAY_DBL,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TPSV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    ap     IN      UTL_NLA_ARRAY_FLT,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–34** *BLAS\_TPSV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: <ul style="list-style-type: none"> <li>uplo = 'U' or 'u' : A is an upper triangular matrix.</li> <li>uplo = 'L' or 'l' : A is a lower triangular matrix.</li> </ul>
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>trans = 'N' or 'n' : <math>A*x = b</math></li> <li>trans = 'T' or 't' : <math>A'*x = b</math></li> <li>trans = 'C' or 'c' : <math>A'*x = b</math></li> </ul>
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>diag = 'U' or 'u' : A is assumed to be unit triangular.</li> <li>diag = 'N' or 'n' : A is not assumed to be unit triangular.</li> </ul>

**Table 231–34 (Cont.) BLAS\_TPSV Procedure Parameters**

Parameter	Description
n	Specifies the order of the matrix A. n must be at least zero.
ap	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>((n*(n+1))/2)</math></p> <p>Before entry with <code>uplo = 'U' or 'u'</code>, the array ap must contain the upper triangular matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(1,1)</code>, <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(1,2)</code> and <code>a(2,2)</code> respectively, and so on.</p> <p>Before entry with <code>uplo = 'L' or 'l'</code>, the array ap must contain the lower triangular matrix packed sequentially, column by column, so that <code>ap(1)</code> contains <code>a(1,1)</code>, <code>ap(2)</code> and <code>ap(3)</code> contain <code>a(2,1)</code> and <code>a(3,1)</code> respectively, and so on.</p> <p>Note that when <code>diag = 'U' or 'u'</code>, the diagonal elements of A are not referenced, but are assumed to be unity.</p>
x	<p>UTL_NLA_ARRAY_FLT/DBL of dimension at least <math>(1 + (n - 1) * \text{abs}(incx))</math></p> <p>Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, x is overwritten with the solution vector x.</p>
incx	Specifies the increment for the elements of x. incx must not be zero.
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

## BLAS\_TRMM Procedures

This procedure performs one of the matrix-matrix operations

```
B := alpha*op( A )*B
```

or

```
B := alpha*B*op( A )
```

where alpha is a scalar, B is an m by n matrix, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

```
op( A ) = A
```

or

```
op( A ) = A'
```

**See Also:** [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#) on page 231-10 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TRMM (
    side   IN      flag,
    uplo   IN      flag,
    transa IN      flag,
    diag   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DOUBLE,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    b      IN OUT  UTL_NLA_ARRAY_DBL,
    ldb    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TRMM (
    side   IN      flag,
    uplo   IN      flag,
    transa IN      flag,
    diag   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLOAT,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    b      IN OUT  UTL_NLA_ARRAY_FLT,
    ldb    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```



## Parameters

**Table 231–35** *BLAS\_TRMM Procedure Parameters*

Parameter	Description
side	Specifies whether the symmetric matrix A appears on the left or right in the operation: <ul style="list-style-type: none"> <li>side = 'L' or 'l' : <math>B := \alpha * op(A) * B</math></li> <li>side = 'R' or 'r' : <math>B := \alpha * B * op(A)</math></li> </ul>
uplo	Specifies whether the upper or lower triangular part of the array A is to be referenced: <ul style="list-style-type: none"> <li>uplo = 'U' or 'u' : A is an upper triangular matrix.</li> <li>uplo = 'L' or 'l' : A is a lower triangular matrix.</li> </ul>
transa	Specifies the form of op(A) to be used in the matrix multiplication as follows: <ul style="list-style-type: none"> <li>transa = 'N' or 'n' : <math>op(A) = A</math></li> <li>transa = 'T' or 't' : <math>op(A) = A'</math></li> <li>transa = 'C' or 'c' : <math>op(A) = A'</math></li> </ul>
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>diag = 'U' or 'u' . A is assumed to be unit triangular.</li> <li>diag = 'N' or 'n' . A is not assumed to be unit triangular.</li> </ul>
m	Specifies the number of rows of the B. m must be at least zero.
n	Specifies the number of columns of B. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, k) where k is m when side = 'L' or 'l', and is n when side = 'R' or 'r'.  Before entry with uplo = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix, and the strictly lower triangular part of A is not referenced.  Before entry with uplo = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced.  Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least $\max(1, m)$ , otherwise lda must be at least $\max(1, n)$ .
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb, n).  Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least $\max(1, m)$ .

**Table 231–35 (Cont.) BLAS\_TRMM Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"><li>▪ 'C': column-major (default)</li><li>▪ 'R': row-major</li></ul>

## BLAS\_TRMV Procedures

This procedure performs the matrix-vector operations

```
x := A*x
```

or

```
x := A'*x
```

where  $x$  is an  $n$  element vector and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TRMV (
  uplo   IN    flag,
  trans  IN    flag,
  diag   IN    flag,
  n      IN    POSITIVEN,
  a      IN    UTL_NLA_ARRAY_DBL,
  lda    IN    POSITIVEN,
  x      IN OUT UTL_NLA_ARRAY_DBL,
  incx   IN    POSITIVEN,
  pack   IN    flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TRMV (
  uplo   IN    flag,
  trans  IN    flag,
  diag   IN    flag,
  n      IN    POSITIVEN,
  a      IN    UTL_NLA_ARRAY_FLT,
  lda    IN    POSITIVEN,
  x      IN OUT UTL_NLA_ARRAY_FLT,
  incx   IN    POSITIVEN,
  pack   IN    flag DEFAULT 'C');
```

### Parameters

**Table 231–36** *BLAS\_TRMV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: <ul style="list-style-type: none"> <li>■ uplo = 'U' or 'u'. A is an upper triangular matrix.</li> <li>■ uplo = 'L' or 'l'. A is a lower triangular matrix.</li> </ul>
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>■ trans = 'N' or 'n' <math>x := A*x</math></li> <li>■ trans = 'T' or 't' <math>x := A'*x</math></li> <li>■ trans = 'C' or 'c' <math>x := A'*x</math></li> </ul>

**Table 231–36 (Cont.) BLAS\_TRMV Procedure Parameters**

Parameter	Description
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>■ diag = 'U' or 'u'. A is assumed to be unit triangular.</li> <li>■ diag = 'N' or 'n'. A is not assumed to be unit triangular.</li> </ul>
n	Specifies the order of the matrix A. n must be at least zero.
a	<p>UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n).</p> <p>Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced.</p> <p>Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced.</p> <p>Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity</p>
lda	Specifies the first dimension of a as declared in the calling (sub) program. lda must be at least max(1, n).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least (1 + (n-1) * as(incx)). Before entry, the incremented array X must contain the n element vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## BLAS\_TRSM Procedures

This procedure performs one of the matrix-matrix operations

$$\text{op}( A ) * X = \text{alpha} * B$$

or

$$X * \text{op}( A ) = \text{alpha} * B$$

where alpha is a scalar, X and B are m by n matrices, A is a unit, or non-unit, upper or lower triangular matrix and op(A) is one of

$$\text{op}( A ) = A$$

or

$$\text{op}( A ) = A'$$

The matrix X is overwritten on B.

**See Also:** [BLAS Level 3 \(Matrix-Matrix Operations\) Subprograms](#) on page 231-10 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TRSM (
    side   IN      flag,
    uplo   IN      flag,
    transa IN      flag,
    diag   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_DOUBLE,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    b      IN OUT  UTL_NLA_ARRAY_DBL,
    ldb    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TRSM (
    side   IN      flag,
    uplo   IN      flag,
    transa IN      flag,
    diag   IN      flag,
    m      IN      POSITIVEN,
    n      IN      POSITIVEN,
    alpha  IN      SCALAR_FLOAT,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    b      IN OUT  UTL_NLA_ARRAY_FLT,
    ldb    IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

## Parameters

**Table 231–37** *BLAS\_TRSM Procedure Parameters*

Parameter	Description
side	Specifies whether the symmetric matrix A appears on the left or right in the operation: <ul style="list-style-type: none"> <li>side = 'L' or 'l' : <math>op(A) * X = alpha * B</math></li> <li>side = 'R' or 'r' : <math>X * op(A) = alpha * B</math></li> </ul>
uplo	Specifies whether the upper or lower triangular part of the array A is to be referenced: <ul style="list-style-type: none"> <li>uplo = 'U' or 'u' : A is an upper triangular matrix.</li> <li>uplo = 'L' or 'l' : A is a lower triangular matrix.</li> </ul>
transa	Specifies the form of $op(A)$ to be used in the matrix multiplication as follows: <ul style="list-style-type: none"> <li>transa = 'N' or 'n' : <math>op(A) = A</math></li> <li>transa = 'T' or 't' : <math>op(A) = A'</math></li> <li>transa = 'C' or 'c' : <math>op(A) = A'</math></li> </ul>
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>diag = 'U' or 'u' . A is assumed to be unit triangular.</li> <li>diag = 'N' or 'n' . A is not assumed to be unit triangular.</li> </ul>
m	Specifies the number of rows of the B. m must be at least zero.
n	Specifies the number of columns of B. n must be at least zero.
alpha	SCALAR_FLOAT/DOUBLE. Specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, k) where k is m when side = 'L' or 'l', and is n when side = 'R' or 'r'.  Before entry with uplo = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix, and the strictly lower triangular part of A is not referenced.  Before entry with uplo = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced.  Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
lda	Specifies the first dimension of a as declared in the calling (sub) program. When side = 'L' or 'l', lda must be at least $\max(1, m)$ , otherwise lda must be at least $\max(1, n)$ .
b	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (ldb, n).  Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the solution matrix X.
ldb	Specifies the first dimension of b as declared in the calling (sub) program. ldb must be at least $\max(1, m)$ .

**Table 231–37 (Cont.) BLAS\_TRSM Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"><li>▪ 'C': column-major (default)</li><li>▪ 'R': row-major</li></ul>

## BLAS\_TRSV Procedures

This procedure solves one of the systems of equations

$$A*x = b$$

or

$$A'*x = b$$

where  $b$  and  $x$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  unit, or non-unit, upper or lower triangular matrix.

**See Also:** [BLAS Level 2 \(Matrix-Vector Operations\) Subprograms](#) on page 231-8 for other subprograms in this group

### Syntax

```
UTL_NLA.BLAS_TRSV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    a      IN      UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVEN,
    x      IN OUT  UTL_NLA_ARRAY_DBL,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.BLAS_TRSV (
    uplo   IN      flag,
    trans  IN      flag,
    diag   IN      flag,
    n      IN      POSITIVEN,
    a      IN      UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVEN,
    x      IN OUT  UTL_NLA_ARRAY_FLT,
    incx   IN      POSITIVEN,
    pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–38** *BLAS\_TRSV Procedure Parameters*

Parameter	Description
uplo	Specifies whether the matrix is an upper or lower triangular matrix: <ul style="list-style-type: none"> <li>uplo = 'U' or 'u'. A is an upper triangular matrix.</li> <li>uplo = 'L' or 'l'. A is a lower triangular matrix.</li> </ul>
trans	Specifies the operation to be performed: <ul style="list-style-type: none"> <li>trans = 'N' or 'n'. <math>A*x = b</math></li> <li>trans = 'T' or 't'. <math>A'*x = b</math></li> <li>trans = 'C' or 'c'. <math>A'*x = b</math></li> </ul>



**Table 231–38 (Cont.) BLAS\_TRSV Procedure Parameters**

Parameter	Description
diag	Specifies whether or not A is unit triangular: <ul style="list-style-type: none"> <li>■ diag = 'U' or 'u'. A is assumed to be unit triangular.</li> <li>■ diag = 'N' or 'n'. A is not assumed to be unit triangular.</li> </ul>
n	Specifies the order of the matrix A. n must be at least zero.
a	UTL_NLA_ARRAY_FLT/DBL of DIMENSION (lda, n). Before entry with uplo = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.
lda	Specifies the first dimension of A as declared in the calling (sub) program. lda must be at least max(1, n).
x	UTL_NLA_ARRAY_FLT/DBL of dimension at least $(1 + (n - 1) * \text{abs}(\text{incx}))$ Before entry, the incremented array X must contain the n element right-hand side vector b. On exit, X is overwritten with the solution vector x.
incx	Specifies the increment for the elements of x. Must not be zero.
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

### Usage Notes

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

## LAPACK\_GBSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where *a* is a band matrix of order *n* with *k*<sub>l</sub> sub diagonals and *k*<sub>u</sub> superdiagonals, and *x* and *b* are *n* by *nrhs* matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor *A* as

$$a = L * U$$

where *L* is a product of permutation and unit lower triangular matrices with *k*<sub>l</sub> sub diagonals, and *U* is upper triangular with *k*<sub>l</sub>+*k*<sub>u</sub> superdiagonals. The factored form of *a* is then used to solve the system of equations

$$a * x = b$$

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_GBSV (
  n      IN      POSITIVEN,
  kl     IN      NATURALN,
  ku     IN      NATURALN,
  nrhs   IN      POSITIVEN,
  ab     IN OUT  UTL_NLA_ARRAY_DBL,
  ldab   IN      POSITIVEN,
  ipiv   IN OUT  UTL_NLA_ARRAY_INT,
  b      IN OUT  UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  info   OUT    INTEGER,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_GBSV (
  n      IN      POSITIVEN,
  kl     IN      NATURALN,
  ku     IN      NATURALN,
  nrhs   IN      POSITIVEN,
  ab     IN OUT  UTL_NLA_ARRAY_FLT,
  ldab   IN      POSITIVEN,
  ipiv   IN OUT  UTL_NLA_ARRAY_INT,
  b      IN OUT  UTL_NLA_ARRAY_FLT,
  ldb    IN      POSITIVEN,
  info   OUT    INTEGER,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–39** LAPACK\_GBSV Procedure Parameters

Parameter	Description
<i>n</i>	The number of linear equations, equivalent to the order of the matrix <i>a</i> . <i>n</i> >= 0
<i>k</i> <sub>l</sub>	The number of sub diagonals within the band of <i>a</i> . <i>k</i> <sub>l</sub> >= 0.

**Table 231–39 (Cont.) LAPACK\_GBSV Procedure Parameters**

Parameter	Description
ku	The number of superdiagonals within the band of $a$ . $ku \geq 0$ .
nrhs	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .
ab	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n).</p> <p>On entry, the matrix <math>a</math> in band storage, in rows <math>kl+1</math> to <math>2*kl+ku+1</math>; rows 1 to <math>kl</math> of the array need not be set. The <math>j</math>-th column of <math>A</math> is stored in the <math>j</math>-th column of the array <math>ab</math>:</p> $ab(kl+ku+1+i-j, j) = a(i, j) \text{ for } \max(1, j-ku) \leq i \leq \min(n, j+kl)$ <p>On exit, details of the factorization: <math>U</math> is stored as an upper triangular band matrix with <math>kl+ku</math> superdiagonals in rows 1 to <math>kl+ku+1</math>, and the multipliers used during the factorization are stored in rows:</p> $kl+ku+2 \text{ to } 2*kl+ku+1$
ldab	<p>The leading dimension of the array <math>ab</math>.</p> $ldab \geq 2*kl+ku+1$
ipiv	<p>INTEGER array, DIMENSION (n).</p> <p>The pivot indices that define the permutation matrix <math>P</math>; row <math>i</math> of the matrix was interchanged with row <math>ipiv(i)</math>.</p>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs).</p> <p>On entry, the <math>n</math> by <math>nrhs</math> matrix of right hand side matrix <math>b</math>.</p> <p>On exit, if <math>info = 0</math>, the <math>n</math> by <math>nrhs</math> solution matrix <math>x</math>.</p>
ldb	<p>The leading dimension of the array <math>b</math>.</p> $ldb \geq \max(1, n)$
info	<ul style="list-style-type: none"> <li>▪ <math>= 0</math> : successful exit</li> <li>▪ <math>&lt; 0</math> : if <math>info = -i</math>, the <math>i</math>-th argument had an illegal value</li> <li>▪ <math>&gt; 0</math> : if <math>info = i</math>, <math>U(i, i)</math> is exactly zero. The factorization has been completed, but the factor <math>U</math> is exactly singular, and the solution has not been computed</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>▪ 'C': column-major (default)</li> <li>▪ 'R': row-major</li> </ul>

## LAPACK\_GEES Procedures

This procedure computes for an  $n$  by  $n$  real nonsymmetric matrix  $A$ , the eigenvalues, the real Schur form  $T$ , and, optionally, the matrix of Schur vectors  $Z$ . This gives the Schur factorization  $A = Z^*T^*(Z^*T^*)$ .

A matrix is in real Schur form if it is upper quasi-triangular with 1 by 1 and 2 by 2 blocks. 2 by 2 blocks will be standardized in the form

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix}$$

where  $b*c < 0$ . The eigenvalues of such a block are  $a \pm \sqrt{bc}$ .

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_GEES (
    jobvs    IN        flag,
    n        IN        POSITIVEN,
    a        IN OUT    UTL_NLA_ARRAY_DBL,
    lda      IN        POSITIVEN,
    wr       IN OUT    UTL_NLA_ARRAY_DBL,
    wi       IN OUT    UTL_NLA_ARRAY_DBL,
    vs       IN OUT    UTL_NLA_ARRAY_DBL,
    ldvs     IN        POSITIVEN,
    info     OUT        INTEGER,
    pack     IN        flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_GEES (
    jobvs    IN        flag,
    n        IN        POSITIVEN,
    a        IN OUT    UTL_NLA_ARRAY_FLT,
    lda      IN        POSITIVEN,
    wr       IN        OUT UTL_NLA_ARRAY_FLT,
    wi       IN        OUT UTL_NLA_ARRAY_FLT,
    vs       IN OUT    UTL_NLA_ARRAY_FLT,
    ldvs     IN        POSITIVEN,
    info     OUT        integer,
    pack     IN        flag DEFAULT 'C');
```

### Parameters

**Table 231–40** LAPACK\_GEES Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>■ 'N': Schur vectors are not computed.</li> <li>■ 'V': Schur vectors are computed.</li> </ul>
n	The order of the matrix $a$ . $N \geq 0$ .
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). <ul style="list-style-type: none"> <li>■ On entry, the <math>n</math> by <math>n</math> matrix <math>A</math>.</li> <li>■ On exit, <math>A</math> has been overwritten by its real Schur form <math>T</math>.</li> </ul>
lda	The leading dimension of the array $a$ . $lda \geq \max(1, n)$ .

**Table 231–40 (Cont.) LAPACK\_GEES Procedure Parameters**

Parameter	Description
wr	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <p>wr and wi contain the real and imaginary parts respectively of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.</p>
wi	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).</p> <p>wr and wi contain the real and imaginary parts respectively of the computed eigenvalues in the same order that they appear on the diagonal of the output Schur form T. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.</p>
vs	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <ul style="list-style-type: none"> <li>■ If jobvs = 'V', vs contains the orthogonal matrix Z of Schur vectors.</li> <li>■ If jobvs = 'N', vs is not referenced.</li> </ul>
ldvs	The leading dimension of the array vs. VS. ldvs >= 1. If jobvs = 'V', ldvs >= N
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, and i is &lt;= N: the QR algorithm failed to compute all the eigenvalues. Elements 1:ILO-1 and i+1:N of wr and wi contain those eigenvalues which have converged. If jobvs = 'V', vs contains the matrix which reduces A to its partially converged Schur form.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_GELS Procedures

This procedure solves overdetermined or underdetermined real linear systems involving an  $m$  by  $n$  matrix  $A$ , or its transpose, using a QR or LQ factorization of  $A$ . It is assumed that  $A$  has full rank.

The following options are provided:

1. If `TRANS = 'N'` and  $m \geq n$ : find the least squares solution of an overdetermined system, that is, solve the least squares problem.  

$$\text{minimize } || B - A * X ||$$
2. If `TRANS = 'N'` and  $m < n$ : find the minimum norm solution of an underdetermined system  $A * X = B$ .
3. If `TRANS = 'T'` and  $m \geq n$ : find the minimum norm solution of an underdetermined system  $A^{**T} * X = B$ .
4. If `TRANS = 'T'` and  $m < n$ : find the least squares solution of an overdetermined system, that is, solve the least squares problem  $\text{minimize } || B - A^{**T} * X ||$ .

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_GELS (
    trans    IN        flag,
    m        IN        POSITIVEN,
    n        IN        POSITIVEN,
    nrhs     IN        POSITIVEN,
    a        IN OUT    UTL_NLA_ARRAY_DBL,
    lda      IN        POSITIVEN,
    b        IN OUT    UTL_NLA_ARRAY_DBL,
    ldb      IN        POSITIVEN,
    info     OUT        INTEGER,
    pack     IN        flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_GELS (
    trans    IN        flag,
    m        IN        POSITIVEN,
    n        IN        POSITIVEN,
    nrhs     IN        POSITIVEN,
    a        IN OUT    UTL_NLA_ARRAY_FLT,
    lda      IN        POSITIVEN,
    b        IN OUT    UTL_NLA_ARRAY_FLT,
    ldb      IN        POSITIVEN,
    info     OUT        INTEGER,
    pack     IN        flag DEFAULT 'C');
```

### Parameters

**Table 231-41** LAPACK\_GELS Procedure Parameters

Parameter	Description
<code>trans</code>	<ul style="list-style-type: none"> <li>CHARACTER = 'N': The linear system involves <math>A</math>.</li> <li>CHARACTER = 'T': The linear system involves <math>A^{**T}</math>.</li> </ul>

**Table 231–41 (Cont.) LAPACK\_GELS Procedure Parameters**

Parameter	Description
m	The number of rows of the matrix a. $M \geq 0$ .
n	The number of columns of the matrix a. $N \geq 0$ .
nrhs	The number of right-hand sides, which is the number of columns of the matrix band x. $nrhs \geq 0$ .
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the matrix b of right hand side vectors, stored columnwise; b is m by nrhs if TRANS = 'N', or n by nrhs if trans = 'T'. On exit, if $m \geq n$ , a is overwritten by details of its QR factorization as returned by SGEQRF. If $m < n$ , A is overwritten by details of its LQ factorization as returned by SGELQF.
lda	The leading dimension of the array A. $lda \geq \max(1, m)$ .
b	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs). On entry, the matrix b of right hand side vectors, stored columnwise. b is m by nrhs if trans = 'n', or n by nrhs if trans = 'T'. On exit, b is overwritten by the solution vectors, stored columnwise: <ul style="list-style-type: none"> <li>■ If trans = 'n' and <math>m \geq n</math>, rows 1 to n of b contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements n+1 to m in that column.</li> <li>■ If trans = 'n' and <math>m &lt; n</math>, rows 1 to n of b contain the minimum norm solution vectors.</li> <li>■ If trans = 'T' and <math>m \geq n</math>, rows 1 to m of b contain the minimum norm solution vectors.</li> <li>■ If trans = 'T' and <math>m &lt; n</math>, rows 1 to m of b contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements m+1 to n in that column.</li> </ul>
ldb	The leading dimension of the array b. $ldb \geq \max(1, m, n)$
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> </ul>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_GESDD Procedures

This procedure computes the singular value decomposition (SVD) of a real  $m$  by  $n$  matrix  $A$ , optionally computing the left and right singular vectors. If singular vectors are desired, it uses a divide-and-conquer algorithm that makes mild assumptions about floating point arithmetic.

The SVD is written

$$A = U * \text{SIGMA} * \text{transpose}(V)$$

where  $\text{SIGMA}$  is an  $m$  by  $n$  matrix which is zero except for its  $\min(m, n)$  diagonal elements,  $U$  is an  $m$  by  $m$  orthogonal matrix, and  $V$  is an  $n$  by  $n$  orthogonal matrix. The diagonal elements of  $\text{SIGMA}$  are the singular values of  $A$ , they are real and non-negative, and are returned in descending order. The first  $\min(m, n)$  columns of  $U$  and  $V$  are the left and right singular vectors of  $A$ .

Note that the routine returns  $V^* * T$ , not  $V$ .

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```

UTL_NLA.LAPACK_GESDD (
    jobz   IN          flag,
    m      IN          POSITIVE,
    n      IN          POSITIVE,
    a      IN OUT     UTL_NLA_ARRAY_DBL,
    lda   IN          POSITIVE,
    s      IN OUT     UTL_NLA_ARRAY_DBL,
    u      IN OUT     UTL_NLA_ARRAY_DBL,
    ldu   IN          POSITIVE,
    vt    IN OUT     UTL_NLA_ARRAY_DBL,
    ldvt  IN          POSITIVE,
    info  OUT         INTEGER,
    pack  IN          flag DEFAULT 'C');

```

```

UTL_NLA.LAPACK_GESDD (
    jobz   IN          flag,
    m      IN          POSITIVE,
    n      IN          POSITIVE,
    a      IN OUT     UTL_NLA_ARRAY_FLT,
    lda   IN          POSITIVE,
    s      IN OUT     UTL_NLA_ARRAY_FLT,
    u      IN OUT     UTL_NLA_ARRAY_FLT,
    ldu   IN          POSITIVE,
    vt    IN OUT     UTL_NLA_ARRAY_FLT,
    ldvt  IN          POSITIVE,
    info  OUT         INTEGER,
    pack  IN          flag DEFAULT 'C');

```



## Parameters

**Table 231–42 LAPACK\_GESDD Procedure Parameters**

Parameter	Description
jobz	Specifies options for computing all or part of the matrix U: <ul style="list-style-type: none"> <li>▪ 'A': All <math>m</math> columns of <math>u</math> and all <math>n</math> rows of <math>V^{*}T</math> are returned in arrays <math>u</math> and <math>vt</math>.</li> <li>▪ 'S': The first <math>\min(m, n)</math> columns of <math>u</math> and the first <math>\min(m, n)</math> rows of <math>V^{*}T</math> are returned in the arrays <math>u</math> and <math>vt</math>.</li> <li>▪ 'O': The first <math>\min(m, n)</math> columns of <math>u</math> (the left singular vectors) are overwritten on the array <math>a</math>. <math>jobu</math> and <math>jobvt</math> cannot both be 'O'</li> <li>▪ 'N': No columns of <math>u</math> (no left singular vectors) are computed.</li> </ul>
m	The order of the matrix $a$ . $m \geq 0$ .
n	The order of the matrix $a$ . $n \geq 0$ .
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the $n$ by $n$ matrix $A$ . On exit: <ul style="list-style-type: none"> <li>▪ If <math>jobz = 'O'</math>, <math>a</math> is overwritten with the first <math>\min(m, n)</math> columns of <math>u</math> (the left singular vectors, stored columnwise).</li> <li>▪ If <math>m \geq n</math>, <math>a</math> is overwritten with the first <math>m</math> rows of <math>V^{*}T</math> (the right singular vectors, stored rowwise).</li> <li>▪ If <math>jobz \neq 'O'</math>, the contents of <math>a</math> are destroyed.</li> </ul>
lda	The leading dimension of the array $a$ . $lda \geq \max(1, m)$ .
s	UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( $\min(m, n)$ ). The singular values of $a$ , sorted so that $S(i) \geq S(i+1)$ .
u	UTL_NLA_ARRAY_FLT/DBL. $ucol = m$ if $jobz = 'A'$ or $jobz = 'O'$ and $m < n$ ; $ucol = \min(m, n)$ if $jobz = 'S'$ . <ul style="list-style-type: none"> <li>▪ If <math>jobz = 'A'</math> or <math>jobz = 'O'</math> and <math>m &lt; n</math>, <math>u</math> contains the <math>m</math> by <math>m</math> orthogonal matrix <math>u</math>.</li> <li>▪ If <math>jobz = 'S'</math>, <math>u</math> contains the first <math>\min(m, n)</math> columns of <math>u</math> (the left singular vectors, stored columnwise).</li> <li>▪ If <math>jobz = 'O'</math> and <math>m \geq n</math>, or <math>jobz = 'n'</math>, <math>u</math> is not referenced.</li> </ul>
ldu	The leading dimension of the array $U$ . $ldu \geq 1$ . If $jobz = 'S'$ or 'A', or $jobz = 'O'$ and $m < n$ , $ldu \geq m$ .
vt	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvt, n). <ul style="list-style-type: none"> <li>▪ If <math>jobz = 'A'</math> or <math>jobz = 'O'</math> and <math>m \geq n</math>, <math>vt</math> contains the <math>n</math> by <math>n</math> orthogonal matrix <math>V^{*}T</math>.</li> <li>▪ If <math>jobz = 'S'</math>, <math>vt</math> contains the first <math>\min(m, n)</math> rows of <math>V^{*}T</math> (the right singular vectors, stored rowwise).</li> <li>▪ If <math>jobz = 'O'</math> and <math>m &lt; n</math>, or <math>jobz = 'N'</math>, <math>vt</math> is not referenced.</li> </ul>

**Table 231–42 (Cont.) LAPACK\_GESDD Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
ldvt	The leading dimension of the array vt. $ldvt \geq 1$ . <ul style="list-style-type: none"> <li>■ If <math>jobz = 'A'</math>, or <math>jobz = 'O'</math> and <math>m \geq n</math>, <math>ldvt \geq n</math>.</li> <li>■ If <math>jobz = 'S'</math>, <math>ldvt \geq \min(m, n)</math>.</li> </ul>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : If <math>info = -i</math>, the <math>i</math>-th argument had an illegal value</li> <li>■ &gt; 0 : SBDSDC did not converge, updating process failed.</li> </ul>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_GESV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  matrix and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor  $A$  as

$$a = P * L * U$$

where  $P$  is a permutation matrix,  $L$  is unit lower triangular, and  $U$  is upper triangular. The factored form of  $a$  is then used to solve the system of equations

$$a * x = b$$

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_GESV (
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_DBL,
  lda    IN      POSITIVEN,
  ipiv   IN OUT  UTL_NLA_ARRAY_INT,
  b      IN OUT  UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_GESV (
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  a      IN OUT  UTL_NLA_ARRAY_FLT,
  lda    IN      POSITIVEN,
  ipiv   IN OUT  UTL_NLA_ARRAY_INT,
  b      IN OUT  UTL_NLA_ARRAY_FLT,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231-43** LAPACK\_GESV Procedure Parameters

Parameter	Description
$n$	The number of linear equations, equivalent to the order of the matrix $a$ . $n \geq 0$
$nrhs$	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .

**Table 231–43 (Cont.) LAPACK\_GESV Procedure Parameters**

Parameter	Description
a	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n).</p> <p>On entry, the n by n coefficient matrix a.</p> <p>On exit, the factors L and U from the factorization <math>a = P * L * U</math>; the unit diagonal elements of L are not stored.</p>
lda	<p>The leading dimension of the array a.</p> <p>lda &gt;= max(1, n)</p>
ipiv	<p>INTEGER array, DIMENSION (n).</p> <p>The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row ipiv(i).</p>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs).</p> <p>On entry, the n by nrhs matrix of right hand side matrix b.</p> <p>On exit, if info = 0, the n by nrhs solution matrix X.</p>
ldb	<p>The leading dimension of the array b.</p> <p>ldb &gt;= max(1, n)</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, U(i, i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_GESVD Procedures

This procedure computes the singular value decomposition (SVD) of a real  $m$  by  $n$  matrix  $A$ , optionally computing the left and/or right singular vectors. The SVD is written

$$A = U * \text{SIGMA} * \text{transpose}(V)$$

where  $\text{SIGMA}$  is an  $m$  by  $n$  matrix which is zero except for its  $\min(m, n)$  diagonal elements,  $U$  is an  $m$  by  $m$  orthogonal matrix, and  $V$  is an  $n$  by  $n$  orthogonal matrix. The diagonal elements of  $\text{SIGMA}$  are the singular values of  $A$ , they are real and non-negative, and are returned in descending order. The first  $\min(m, n)$  columns of  $U$  and  $V$  are the left and right singular vectors of  $A$ .

Note that the routine returns  $V^{*T}$ , not  $V$ .

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```

UTL_NLA.LAPACK_GESVD (
  jobu  IN      flag,
  jobvt IN      flag,
  m     IN      POSITIVEN,
  n     IN      POSITIVEN,
  a     IN OUT  UTL_NLA_ARRAY_DBL,
  lda   IN      POSITIVEN,
  s     IN OUT  UTL_NLA_ARRAY_DBL,
  u     IN OUT  UTL_NLA_ARRAY_DBL,
  ldu   IN      POSITIVEN,
  vt    IN OUT  UTL_NLA_ARRAY_DBL,
  ldvt  IN      POSITIVEN,
  info  OUT     INTEGER,
  pack  IN      flag DEFAULT 'C');

```

```

UTL_NLA.LAPACK_GESVD (
  jobu  IN      flag,
  jobvt IN      flag,
  m     IN      POSITIVEN,
  n     IN      POSITIVEN,
  a     IN OUT  UTL_NLA_ARRAY_FLT,
  lda   IN      POSITIVEN,
  s     IN OUT  UTL_NLA_ARRAY_FLT,
  u     IN OUT  UTL_NLA_ARRAY_FLT,
  ldu   IN      POSITIVEN,
  vt    IN OUT  UTL_NLA_ARRAY_FLT,
  ldvt  IN      POSITIVEN,
  info  OUT     INTEGER,
  pack  IN      flag DEFAULT 'C');

```

## Parameters

**Table 231–44** LAPACK\_GESVD Procedure Parameters

Parameter	Description
jobu	<p>Specifies options for computing all or part of the matrix U:</p> <ul style="list-style-type: none"> <li>▪ 'A': All <math>m</math> columns of U are returned in array U.</li> <li>▪ 'S': The first <math>\min(m, n)</math> columns of U (the left singular vectors) are returned in the array U.</li> <li>▪ 'O': The first <math>\min(m, n)</math> columns of U (the left singular vectors) are overwritten on the array a. jobu and jobvt cannot both be 'O'.</li> <li>▪ 'N': No columns of U (no left singular vectors) are computed.</li> </ul>
jobvt	<p>Specifies options for computing all or part of the matrix <math>V^{*}T</math>:</p> <ul style="list-style-type: none"> <li>▪ 'A': All <math>n</math> rows of <math>V^{*}T</math> are returned in the array vt.</li> <li>▪ 'S': The first <math>\min(m, n)</math> rows of <math>V^{*}T</math> (the right singular vectors) are returned in the array vt.</li> <li>▪ 'O': The first <math>\min(m, n)</math> rows of <math>V^{*}T</math> (the right singular vectors) are overwritten on the array a. jobvt and jobu cannot both be 'O'.</li> <li>▪ 'N': No rows of <math>V^{*}T</math> (no right singular vectors) are computed.</li> </ul>
m	The order of the matrix a. $M \geq 0$ .
n	The order of the matrix a. $N \geq 0$ .
a	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n).</p> <p>On entry, the <math>n</math> by <math>n</math> matrix A.</p> <p>On exit:</p> <ul style="list-style-type: none"> <li>▪ If jobu = 'O', A is overwritten with the first <math>\min(m, n)</math> columns of U (the left singular vectors, stored columnwise);</li> <li>▪ If jobvt = 'O', A is overwritten with the first <math>\min(m, n)</math> rows of <math>V^{*}T</math> (the right singular vectors, stored rowwise);</li> <li>▪ If jobu.ne.'O' and jobvt.ne.'O', the contents of A are destroyed.</li> </ul>
lda	The leading dimension of the array a. $lda \geq \max(1, n)$ .
s	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (min(m, n)).</p> <p>The singular values of A, sorted so that <math>S(i) \geq S(i+1)</math>.</p>
u	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldu, ucol). (ldu, m) if jobu = 'A' or (ldu, min(m, n)) if jobu = 'S'.</p> <ul style="list-style-type: none"> <li>▪ If jobu = 'A', U contains the <math>m</math> by <math>m</math> orthogonal matrix U.</li> <li>▪ If jobu = 'S', U contains the first <math>\min(m, n)</math> columns of U (the left singular vectors, stored columnwise).</li> <li>▪ If jobu = 'N' or 'O', U is not referenced.</li> </ul>
ldu	The leading dimension of the array U. $ldu \geq 1$ . If jobu = 'S' or 'a', $ldu \geq m$ .

**Table 231-44 (Cont.) LAPACK\_GESVD Procedure Parameters**

Parameter	Description
vt	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvt, n). <ul style="list-style-type: none"> <li>■ If jobvt = 'A', vt contains the n by n orthogonal matrix <math>V^{*}T</math>.</li> <li>■ If jobvt = 'S', vt contains the first min(m,n) rows of <math>V^{*}T</math> (the right singular vectors, stored rowwise).</li> <li>■ If jobvt = 'N' or 'O', vt is not referenced.</li> </ul>
ldvt	The leading dimension of the array vt. ldvt $\geq$ 1. <ul style="list-style-type: none"> <li>■ If jobvt = 'A', ldvt <math>\geq</math> n.</li> <li>■ If jobvt = 'S', ldvt <math>\geq</math> min(m,n).</li> </ul>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : If info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : If SBDSQR did not converge, info specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero.</li> </ul>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_GEEV Procedures

This procedure computes for an  $n$  by  $n$  real nonsymmetric matrix  $A$ , the eigenvalues and, optionally, the left and/or right eigenvectors.

- The right eigenvector  $v(j)$  of  $A$  satisfies  $A * v(j) = \text{lambda}(j) * v(j)$  where  $\text{lambda}(j)$  is its eigenvalue.
- The left eigenvector  $u(j)$  of  $A$  satisfies  $u(j)**H * A = \text{lambda}(j) * u(j)**H$  where  $u(j)**H$  denotes the conjugate transpose of  $u(j)$ .

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```

UTL_NLA.LAPACK_GEEV (
    jobvl  IN      flag,
    jobvr  IN      flag,
    n      IN      POSITIVE,
    a      IN OUT  UTL_NLA_ARRAY_DBL,
    lda    IN      POSITIVE,
    wr     IN OUT  UTL_NLA_ARRAY_DBL,
    wi     IN OUT  UTL_NLA_ARRAY_DBL,
    vl     IN OUT  UTL_NLA_ARRAY_DBL,
    ldvl   IN      POSITIVE,
    vr     IN OUT  UTL_NLA_ARRAY_DBL,
    ldvr   IN      POSITIVE,
    info   OUT     INTEGER,
    pack   IN      flag DEFAULT 'C');

```

```

UTL_NLA.LAPACK_GEEV (
    jobvl  IN      flag,
    jobvr  IN      flag,
    n      IN      POSITIVE,
    a      IN OUT  UTL_NLA_ARRAY_FLT,
    lda    IN      POSITIVE,
    wr     IN OUT  UTL_NLA_ARRAY_FLT,
    wi     IN OUT  UTL_NLA_ARRAY_FLT,
    vl     IN OUT  UTL_NLA_ARRAY_FLT,
    ldvl   IN      POSITIVE,
    vr     IN OUT  UTL_NLA_ARRAY_FLT,
    ldvr   IN      POSITIVE,
    info   OUT     INTEGER,
    pack   IN      flag DEFAULT 'C');

```

### Parameters

**Table 231–45** LAPACK\_GEEV Procedure Parameters

Parameter	Description
jobvl	<ul style="list-style-type: none"> <li>■ 'N': Left eigenvectors of <math>A</math> are not computed.</li> <li>■ 'V': Left eigenvectors of <math>A</math> are computed.</li> </ul>
jobvr	<ul style="list-style-type: none"> <li>■ 'N': Right eigenvectors of <math>A</math> are not computed.</li> <li>■ 'V': Right eigenvectors of <math>A</math> are computed.</li> </ul>



**Table 231–45 (Cont.) LAPACK\_GEEV Procedure Parameters**

Parameter	Description
n	The order of the matrix a. $N \geq 0$ .
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). <ul style="list-style-type: none"> <li>On entry, the n by n matrix A.</li> <li>On exit, A has been overwritten.</li> </ul>
lda	The leading dimension of the array a. $lda \geq \max(1, n)$ .
wr	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). wr and wi contain the real and imaginary parts respectively of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
wi	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). wr and wi contain the real and imaginary parts respectively of the computed eigenvalues. Complex conjugate pairs of eigenvalues will appear consecutively with the eigenvalue having the positive imaginary part first.
v1	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). <ul style="list-style-type: none"> <li>If jobv1 = 'V', the left eigenvectors u(j) are stored one after another in the columns of v1, in the same order as their eigenvalues.</li> <li>If jobvs = 'N', v1 is not referenced.</li> <li>If the j-th eigenvalue is real, then <math>u(j) = VL(:, j)</math>, the j-th column of v1.</li> <li>If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then <math>u(j) = VL(:, j) + i*VL(:, j+1)</math> and <math>u(j+1) = VL(:, j) - i*VL(:, j+1)</math>.</li> </ul>
ldv1	The leading dimension of the array v1. $ldv1 \geq 1$ . If jobv1 = 'v', $ldv1 \geq n$ .
vr	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldvr, n). <ul style="list-style-type: none"> <li>If jobvr = 'V', the right eigenvectors v(j) are stored one after another in the columns of vr, in the same order as their eigenvalues..</li> <li>If jobvr = 'N', vr is not referenced.</li> <li>If the j-th eigenvalue is real, then <math>v(j) = VR(:, j)</math>, the j-th column of vr.</li> <li>If the j-th and (j+1)-st eigenvalues form a complex conjugate pair, then <math>v(j) = VR(:, j) + i*VR(:, j+1)</math> and <math>v(j+1) = VR(:, j) - i*VR(:, j+1)</math>.</li> </ul>
ldvr	The leading dimension of the array vr. $ldvr \geq 1$ . If jobvr = 'V', $ldvr \geq N$
info	<ul style="list-style-type: none"> <li>= 0 : successful exit</li> <li>&lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>&gt; 0 : if info = i, and i is <math>\leq N</math>: the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed. Elements i+1:N of wr and wi contain eigenvalues which have converged..</li> </ul>

**Table 231–45 (Cont.) LAPACK\_GEEV Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"><li>▪ 'C': column-major (default)</li><li>▪ 'R': row-major</li></ul>

## LAPACK\_GTSV Procedures

This procedure solves the equation

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation  $a' * x = b$  may be solved by interchanging the order of the arguments  $du$  and  $d1$ .

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_GTSV (
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  d1     IN OUT  UTL_NLA_ARRAY_DBL,
  d      IN OUT  UTL_NLA_ARRAY_DBL,
  du     IN OUT  UTL_NLA_ARRAY_DBL,
  b      IN OUT  UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  info   OUT    INTEGER,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_GTSV (
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  d1     IN OUT  UTL_NLA_ARRAY_FLT,
  d      IN OUT  UTL_NLA_ARRAY_FLT,
  du     IN OUT  UTL_NLA_ARRAY_FLT,
  b      IN OUT  UTL_NLA_ARRAY_FLT,
  ldb    IN      POSITIVEN,
  info   OUT    INTEGER,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–46** LAPACK\_GTSV Procedure Parameters

Parameter	Description
$n$	The order of the matrix $a$ . $n \geq 0$
$nrhs$	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .
$d1$	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1). On entry, $d1$ must contain the (n-1) sub-diagonal elements of $a$ . On exit, $d1$ is overwritten by the (n-2) elements of the second super-diagonal of the upper triangular matrix $U$ from the LU factorization of $a$ , in $d1(1), \dots, d1(n-2)$ .
$d$	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). On entry, $d$ must contain the diagonal elements of $a$ . On exit, $d$ is overwritten by the $n$ diagonal elements of $U$ .

**Table 231–46 (Cont.) LAPACK\_GTSV Procedure Parameters**

Parameter	Description
du	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1).</p> <p>On entry, du must contain the (n-1) super-diagonal elements of a.</p> <p>On exit, du is overwritten by the (n-1) elements of the first super-diagonal of U.</p>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (LDB, nrhs).</p> <p>On entry, the n by nrhs matrix of right hand side matrix b.</p> <p>On exit, if info = 0, the n by nrhs solution matrix X.</p>
ldb	<p>The leading dimension of the array b.</p> <p>ldb &gt;= max (1, n)</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i , the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, U(i, i) is exactly zero, and the solution has not been computed. The factorization has not been completed unless i = n.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_PBSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  symmetric positive definite band matrix and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

The Cholesky decomposition is used to factor  $A$  as

$$A = U^{*T}U \text{ if } UPLO = 'U'$$

or

$$A = L * L^{*T} \text{ if } UPLO = 'L'$$

where  $U$  is an upper triangular matrix and  $L$  is a lower triangular matrix. The factored form of  $A$  is then used to solve the system of equations  $A * X = B$ .

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_PBSV (
  uplo      IN      flag,
  n         IN      POSITIVEN,
  kd        IN      NATURALN,
  nrhs      IN      POSITIVEN,
  ab        IN OUT  UTL_NLA_ARRAY_DBL,
  ldab      IN      POSITIVEN,
  b         IN OUT  UTL_NLA_ARRAY_DBL,
  ldb       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_PBSV (
  uplo      IN      flag,
  n         IN      POSITIVEN,
  kd        IN      NATURALN,
  nrhs      IN      POSITIVEN,
  ab        IN OUT  UTL_NLA_ARRAY_FLT,
  ldab      IN      POSITIVEN,
  b         IN OUT  UTL_NLA_ARRAY_FLT,
  ldb       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–47** LAPACK\_PBSV Procedure Parameters

Parameter	Description
uplo	<ul style="list-style-type: none"> <li>▪ uplo = 'U'. Upper triangular of <math>A</math> is stored.</li> <li>▪ uplo = 'L'. Lower triangular of <math>A</math> is stored.</li> </ul>
n	The number of linear equations, that is, the order of the matrix $a$ . $n \geq 0$

**Table 231–47 (Cont.) LAPACK\_PBSV Procedure Parameters**

Parameter	Description
kd	The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs >= 0.
ab	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n).</p> <p>On entry, the upper or lower triangle of the symmetric band matrix a, stored in the first kd+1 rows of the array. The j-th column of a is stored in the j-th column of the array ab as follows:</p> <ul style="list-style-type: none"> <li>■ if uplo = 'U', AB(KD+1+i-j, j) = A(i, j) for <math>\max(1, j-KD) \leq i \leq j</math>;</li> <li>■ if uplo = 'L', AB(1+i-j, j) = A(i, j) for <math>j \leq i \leq \min(N, j+KD)</math></li> </ul> <p>.See below for further details.</p> <p>On exit, if info = 0, the triangular factor U or L from the Cholesky factorization <math>A = U^{*T}U</math> or <math>A = L^{*L^{*T}}</math> of the band matrix A, in the same storage format as a.</p>
ldab	<p>The leading dimension of the array ab.</p> <p>ldab &gt;= kd+1</p>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs).</p> <p>On entry, the n by nrhs matrix of right hand side matrix b.</p> <p>On exit, if info = 0, the n by nrhs solution matrix X.</p>
ldb	<p>The leading dimension of the array b.</p> <p>ldb &gt;= max(1, n)</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, the leading minor of order i of a is not positive definite, so the factorization could not be completed, and the solution has not been computed.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_POSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  symmetric positive definite matrix and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

The Cholesky decomposition is used to factor  $A$  as

$$A = U^{**T} * U \text{ if } uplo = 'U'$$

or

$$A = L * L^{**T} \text{ if } UPLO = 'L'$$

where  $U$  is an upper triangular matrix and  $L$  is a lower triangular matrix. The factored form of  $A$  is then used to solve the system of equations  $A * X = B$ .

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_POSV (
  uplo      IN      flag,
  n         IN      POSITIVEN,
  nrhs      IN      POSITIVEN,
  a         IN OUT  UTL_NLA_ARRAY_DBL,
  lda       IN      POSITIVEN,
  b         IN OUT  UTL_NLA_ARRAY_DBL,
  ldb       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_POSV (
  uplo      IN      flag,
  n         IN      POSITIVEN,
  nrhs      IN      POSITIVEN,
  a         IN OUT  UTL_NLA_ARRAY_FLT,
  lda       IN      POSITIVEN,
  b         IN OUT  UTL_NLA_ARRAY_FLT,
  ldb       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–48** LAPACK\_POSV Procedure Parameters

Parameter	Description
uplo	<ul style="list-style-type: none"> <li>■ uplo = 'U'. Upper triangular of <math>A</math> is stored.</li> <li>■ uplo = 'L'. Lower triangular of <math>A</math> is stored.</li> </ul>
n	The number of linear equations, that is, the order of the matrix $a$ . $n \geq 0$
nrhs	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .

**Table 231–48 (Cont.) LAPACK\_POSV Procedure Parameters**

Parameter	Description
a	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n).</p> <p>If <code>uplo = 'U'</code>, the leading NRHS <math>n</math> by <math>n</math> upper triangular part of <code>a</code> contains the upper NRHS triangular part of the matrix <code>A</code>, and the strictly lower NRHS triangular part of <code>A</code> is not referenced.</p> <p>If <code>uplo = 'L'</code>, then <code>rhs</code> leading <math>n</math> by <math>n</math> lower triangular part of <code>a</code> contains the lower nrhs triangular part of the matrix <code>a</code>, and the strictly upper nrhs triangular part of <code>a</code> is not referenced.</p> <p>On exit, if <code>info = 0</code>, the factor <code>U</code> or <code>L</code> from the Cholesky factorization <math>A = U^*T*U</math> or <math>A = L*L^*T</math>.</p>
lda	<p>The leading dimension of the array <code>a</code>.</p> <p><math>lda \geq \max(1, n)</math></p>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs).</p> <p>On entry, the <math>n</math> by <code>nrhs</code> matrix of right hand side matrix <code>b</code>.</p> <p>On exit, if <code>info = 0</code>, the <math>n</math> by <code>nrhs</code> solution matrix <code>x</code>.</p>
ldb	<p>The leading dimension of the array <code>b</code>.</p> <p><math>ldb \geq \max(1, n)</math></p>
info	<ul style="list-style-type: none"> <li>▪ <code>= 0</code> : successful exit</li> <li>▪ <code>&lt; 0</code> : if <code>info = -i</code>, the <math>i</math>-th argument had an illegal value</li> <li>▪ <code>&gt; 0</code> : if <code>info = i</code>, the leading minor of order <math>i</math> of <code>a</code> is not positive definite, so the factorization could not be completed, and the solution has not been computed.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>▪ <code>'C'</code>: column-major (default)</li> <li>▪ <code>'R'</code>: row-major</li> </ul>



## LAPACK\_PPSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  symmetric positive definite matrix stored in packed format and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

The Cholesky decomposition is used to factor  $A$  as

$$A = U^{**T} * U \text{ if UPLO} = 'U'$$

or

$$A = L * L^{**T} \text{ if UPLO} = 'L'$$

where  $U$  is an upper triangular matrix and  $L$  is a lower triangular matrix. The factored form of  $A$  is then used to solve the system of equations  $A * X = B$ .

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_PPSV (
  uplo      IN      flag,
  n         IN      POSITIVEN,
  nrhs      IN      POSITIVEN,
  ap        IN OUT  UTL_NLA_ARRAY_DBL,
  b         IN OUT  UTL_NLA_ARRAY_DBL,
  ldb       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_PPSV (
  uplo      IN      flag,
  n         IN      POSITIVEN,
  nrhs      IN      POSITIVEN,
  ap        IN OUT  UTL_NLA_ARRAY_FLT,
  b         IN OUT  UTL_NLA_ARRAY_FLT,
  ldb       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231-49 LAPACK\_PPSV Procedure Parameters**

Parameter	Description
uplo	<ul style="list-style-type: none"> <li>■ uplo = 'U'. Upper triangular of <math>A</math> is stored.</li> <li>■ uplo = 'L'. Lower triangular of <math>A</math> is stored.</li> </ul>
n	The number of linear equations, that is, the order of the matrix $a$ . $n \geq 0$
nrhs	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .

**Table 231-49 (Cont.) LAPACK\_PPSV Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION <math>(n*(n+1)/2)</math>.</p> <p>On entry, the upper or lower triangle of the symmetric matrix <i>a</i>, packed columnwise in a linear array. The <i>j</i>-th column of <i>a</i> is stored in the array <i>ap</i> as follows:</p> <p>If <i>uplo</i> = 'U', <math>AP(i + (j-1)*j/2) = A(i, j)</math> for <math>1 \leq i \leq j</math>;</p> <p>If <i>uplo</i> = 'L', <math>AP(i + (j-1)*(2n-j)/2) = A(i, j)</math> for <math>j \leq i \leq n</math>;</p> <p>On exit, if <i>info</i> = 0, the factor <i>U</i> or 'L' from the Cholesky factorization <math>A = U**T*U</math> or <math>A = L*L**T</math> in the same storage format as <i>A</i>.</p>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (<i>ldb</i>, <i>nrhs</i>).</p> <p>On entry, the <i>n</i> by <i>nrhs</i> matrix of right hand side matrix <i>b</i>.</p> <p>On exit, if <i>info</i> = 0, the <i>n</i> by <i>nrhs</i> solution matrix <i>X</i>.</p>
ldb	<p>The leading dimension of the array <i>b</i>.</p> <p><math>ldb \geq \max(1, n)</math></p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if <i>info</i> = -<i>i</i>, the <i>i</i>-th argument had an illegal value</li> <li>■ &gt; 0 : if <i>info</i> = <i>i</i>, the leading minor of order <i>i</i> of <i>a</i> is not positive definite, so the factorization could not be completed, and the solution has not been computed.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_PTSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  symmetric positive definite tridiagonal matrix, and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

$a$  is factored as  $A = L * D * L^{*T}$ , and the factored form of  $a$  is then used to solve the system of equations.

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_PTSV (
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  d      IN OUT  UTL_NLA_ARRAY_DBL,
  e      IN OUT  UTL_NLA_ARRAY_DBL,
  b      IN OUT  UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_PTSV (
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  d      IN OUT  UTL_NLA_ARRAY_FLT,
  e      IN OUT  UTL_NLA_ARRAY_FLT,
  b      IN OUT  UTL_NLA_ARRAY_FLT,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–50** LAPACK\_PTSV Procedure Parameters

Parameter	Description
$n$	The order of the matrix $a$ . $N \geq 0$ .
$nrhs$	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .
$d$	UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( $n$ ). On entry, the $n$ diagonal elements of the tridiagonal matrix $a$ . On exit, the $n$ diagonal elements of the diagonal matrix $d$ from the factorization $A = L * D * L^{*T}$ .
$e$	UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( $n-1$ ). On entry, the ( $n-1$ ) subdiagonal elements of the tridiagonal matrix $a$ . On exit, the ( $n-1$ ) diagonal elements of the unit bidiagonal factor $L$ from the factorization $A = L * D * L^{*T}$ of $a$ . ( $e$ can also be regarded as the superdiagonal of the unit bidiagonal factor $U$ from the $U^{*T} * D * U$ factorization of $a$ )

**Table 231–50 (Cont.) LAPACK\_PTSV Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>b</code>	UTL_NLA_ARRAY_FLT/DBL, DIMENSION ( <code>ldb</code> , <code>nrhs</code> ). On entry, the <code>n</code> by <code>nrhs</code> matrix of right hand side matrix <code>b</code> . On exit, if <code>info = 0</code> , the <code>n</code> by <code>nrhs</code> solution matrix <code>X</code> .
<code>ldb</code>	The leading dimension of the array <code>b</code> . $ldb \geq \max(1, n)$
<code>info</code>	<ul style="list-style-type: none"> <li>▪ <code>= 0</code> : successful exit</li> <li>▪ <code>&lt; 0</code> : if <code>info = -i</code>, the <code>i</code>-th argument had an illegal value</li> <li>▪ <code>&gt; 0</code> : if <code>info = i</code>, the leading minor of order <code>i</code> of <code>a</code> is not positive definite, so the factorization could not be completed, and the solution has not been computed.</li> </ul>
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>▪ <code>'C'</code>: column-major (default)</li> <li>▪ <code>'R'</code>: row-major</li> </ul>

## LAPACK\_SBEV Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```

UTL_NLA.LAPACK_SBEV (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  kd        IN      NATURALN,
  ab        IN OUT  UTL_NLA_ARRAY_DBL,
  ldab      IN      POSITIVEN,
  w         IN OUT  UTL_NLA_ARRAY_DBL,
  z         IN OUT  UTL_NLA_ARRAY_DBL,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');

```

```

UTL_NLA.LAPACK_SBEV (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  kd        IN      NATURALN,
  ab        IN OUT  UTL_NLA_ARRAY_FLT,
  ldab      IN      POSITIVEN,
  w         IN OUT  UTL_NLA_ARRAY_FLT,
  z         IN OUT  UTL_NLA_ARRAY_FLT,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');

```

### Parameters

**Table 231–51** LAPACK\_SBEV Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
uplo	<ul style="list-style-type: none"> <li>▪ 'U': Upper triangle of A is stored.</li> <li>▪ 'L': Lower triangle of A is stored.</li> </ul>
n	The order of the matrix A. $N \geq 0$ .
kd	The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if uplo = 'L'. $kd \geq 0$ .

**Table 231–51 (Cont.) LAPACK\_SBEV Procedure Parameters**

Parameter	Description
ab	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n).</p> <p>On entry, the upper or lower triangle of the symmetric band matrix <math>A</math> stored in the first <math>kd+1</math> rows of the array. The <math>j</math>-th column of <math>A</math> is stored in the <math>j</math>-th column of the array <math>ab</math>:</p> <ul style="list-style-type: none"> <li>■ If <code>uplo = 'U'</code>, <math>ab(kd+1+i-j, j) = a(i, j)</math> for <math>\max(1, j-kd) \leq i \leq j</math>.</li> <li>■ If <code>uplo = 'L'</code>, <math>ab(1+i-j, j) = A(i, j)</math> for <math>j \leq i \leq \min(n, j+kd)</math>.</li> </ul> <p>On exit, <math>ab</math> is overwritten by values generated during the reduction to tridiagonal form:</p> <ul style="list-style-type: none"> <li>■ If <code>uplo = 'U'</code>, the diagonal and first superdiagonal of the tridiagonal matrix <math>T</math> are returned in rows <math>kd</math> and <math>kd+1</math> of <math>ab</math>.</li> <li>■ If <code>uplo = 'L'</code>, the diagonal and first subdiagonal of <math>T</math> are returned in the first two rows of <math>ab</math>.</li> </ul>
ldab	The leading dimension of the array $ab$ . $ldab \geq kd + 1$ .
w	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <p>If <code>info = 0</code>, the eigenvalues in ascending order.</p>
z	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <ul style="list-style-type: none"> <li>■ If <code>jobz = 'V'</code>, then if <code>info = 0</code>, <math>z</math> contains the orthonormal eigenvectors of the matrix <math>A</math>, with the <math>i</math>-th column of <math>z</math> holding the eigenvector associated with <math>w(i)</math>.</li> <li>■ If <code>jobz = 'N'</code>, then <math>z</math> is not referenced.</li> </ul>
ldz	The leading dimension of the array $z$ . $ldz \geq 1$ , and if <code>jobz = 'V'</code> , $ldz \geq \max(1, n)$ .
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if <code>info = -i</code>, the <math>i</math>-th argument had an illegal value</li> <li>■ &gt; 0 : if <code>info = i</code>, the algorithm failed to converge; <math>i</math> off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_SBEVD Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```

UTL_NLA.LAPACK_SBEVD (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  kd        IN      NATURALN,
  ab        IN OUT  UTL_NLA_ARRAY_DBL,
  ldab      IN      POSITIVEN,
  w         IN OUT  UTL_NLA_ARRAY_DBL,
  z         IN OUT  UTL_NLA_ARRAY_DBL,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');

```

```

UTL_NLA.LAPACK_SBEVD (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  kd        IN      NATURALN,
  ab        IN OUT  UTL_NLA_ARRAY_FLT,
  ldab      IN      POSITIVEN,
  w         IN OUT  UTL_NLA_ARRAY_FLT,
  z         IN OUT  UTL_NLA_ARRAY_FLT,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');

```

### Parameters

**Table 231–52** LAPACK\_SBEVD Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
uplo	<ul style="list-style-type: none"> <li>▪ 'U': Upper triangle of A is stored.</li> <li>▪ 'L': Lower triangle of A is stored.</li> </ul>
n	The order of the matrix a. $N \geq 0$ .
kd	The number of superdiagonals of the matrix A if uplo = 'U', or the number of subdiagonals if uplo = 'L'. $kd \geq 0$ .

**Table 231–52 (Cont.) LAPACK\_SBEVD Procedure Parameters**

Parameter	Description
ab	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldab, n).</p> <p>On entry, the upper or lower triangle of the symmetric band matrix <math>A</math> stored in the first <math>kd+1</math> rows of the array. The <math>j</math>-th column of <math>A</math> is stored in the <math>j</math>-th column of the array <math>ab</math>:</p> <ul style="list-style-type: none"> <li>■ If <code>uplo = 'U'</code>, <math>ab(kd+1+i-j, j) = a(i, j)</math> for <math>\max(1, j-kd) \leq i \leq j</math>.</li> <li>■ If <code>uplo = 'L'</code>, <math>AB(1+i-j, j) = A(i, j)</math> for <math>j \leq i \leq \min(n, j+kd)</math>.</li> </ul> <p>On exit, <math>ab</math> is overwritten by values generated during the reduction to tridiagonal form:</p> <ul style="list-style-type: none"> <li>■ If <code>uplo = 'U'</code>, the diagonal and first superdiagonal of the tridiagonal matrix <math>T</math> are returned in rows <math>kd</math> and <math>kd+1</math> of <math>ab</math>.</li> <li>■ If <code>uplo = 'L'</code>, the diagonal and first subdiagonal of <math>T</math> are returned in the first two rows of <math>ab</math>.</li> </ul>
ldab	The leading dimension of the array $ab$ . $ldab \geq kd + 1$ .
w	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).</p> <p>If <code>info = 0</code>, the eigenvalues in ascending order.</p>
z	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <ul style="list-style-type: none"> <li>■ If <code>jobz = 'V'</code>, then if <code>info = 0</code>, <math>z</math> contains the orthonormal eigenvectors of the matrix <math>A</math>, with the <math>i</math>-th column of <math>z</math> holding the eigenvector associated with <math>w(i)</math>.</li> <li>■ If <code>jobz = 'N'</code>, then <math>z</math> is not referenced.</li> </ul>
ldz	The leading dimension of the array $z$ . $ldz \geq 1$ , and if <code>jobz = 'V'</code> , $ldz \geq \max(1, n)$ .
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if <code>info = -i</code>, the <math>i</math>-th argument had an illegal value</li> <li>■ &gt; 0 : if <code>info = i</code>, the algorithm failed to converge; <math>i</math> off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>



## LAPACK\_SPEV Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A in packed storage.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_SPEV (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  ap        IN OUT  UTL_NLA_ARRAY_DBL,
  w         IN OUT  UTL_NLA_ARRAY_DBL,
  z         IN OUT  UTL_NLA_ARRAY_DBL,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_SPEV (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  ap        IN OUT  UTL_NLA_ARRAY_FLT,
  w         IN OUT  UTL_NLA_ARRAY_FLT,
  z         IN OUT  UTL_NLA_ARRAY_FLT,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–53** LAPACK\_SPEV Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
uplo	<ul style="list-style-type: none"> <li>▪ 'U': Upper triangle of A is stored.</li> <li>▪ 'L': Lower triangle of A is stored.</li> </ul>
n	The order of the matrix a. $N \geq 0$ .

**Table 231–53 (Cont.) LAPACK\_SPEV Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2).</p> <p>On entry, the upper or lower triangle of the symmetric matrix a packed columnwise in a linear array. The j-th column of a is stored in the array ap:</p> <ul style="list-style-type: none"> <li>■ If uplo = 'U', ap(i + (j-1)*j/2) = a(i, j) for 1&lt;=i&lt;=j.</li> <li>■ If uplo = 'L', ap(i + (j-1)*(2*n-j)/2) = a(i, j) for j&lt;=i&lt;=n.</li> </ul> <p>On exit, ap is overwritten by values generated during the reduction to tridiagonal form:</p> <ul style="list-style-type: none"> <li>■ If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A.</li> <li>■ If uplo = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.</li> </ul>
w	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <p>If info = 0, the eigenvalues in ascending order.</p>
z	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).</p> <ul style="list-style-type: none"> <li>■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i).</li> <li>■ If jobz = 'N', then z is not referenced.</li> </ul>
ldz	<p>The leading dimension of the array z. ldz &gt;= 1, and if jobz = 'V', ldz &gt;= max(1, n).</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_SPEVD Procedures

This procedure computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix *A* in packed storage. If eigenvectors are desired, it uses a divide and conquer algorithm. The divide and conquer algorithm makes very mild assumptions about floating point arithmetic.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_SPEVD (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  ap        IN OUT  UTL_NLA_ARRAY_DBL,
  w         IN OUT  UTL_NLA_ARRAY_DBL,
  z         IN OUT  UTL_NLA_ARRAY_DBL,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_SPEVD (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  ap        IN OUT  UTL_NLA_ARRAY_FLT,
  w         IN OUT  UTL_NLA_ARRAY_FLT,
  z         IN OUT  UTL_NLA_ARRAY_FLT,
  ldz       IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–54** *LAPACK\_SPEVD Procedure Parameters*

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
uplo	<ul style="list-style-type: none"> <li>▪ 'U': Upper triangle of <i>A</i> is stored.</li> <li>▪ 'L': Lower triangle of <i>A</i> is stored.</li> </ul>
n	The order of the matrix <i>a</i> . $N \geq 0$ .

**Table 231–54 (Cont.) LAPACK\_SPEVD Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2).</p> <p>On entry, the upper or lower triangle of the symmetric matrix a packed columnwise in a linear array. The j-th column of a is stored in the array ap:</p> <ul style="list-style-type: none"> <li>■ If uplo = 'U', ap(i + (j-1)*j/2) = a(i, j) for 1&lt;=i&lt;=j.</li> <li>■ If uplo = 'L', ap(i + (j-1)*(2*n-j)/2) = a(i, j) for j&lt;=i&lt;=n.</li> </ul> <p>On exit, ap is overwritten by values generated during the reduction to tridiagonal form:</p> <ul style="list-style-type: none"> <li>■ If uplo = 'U', the diagonal and first superdiagonal of the tridiagonal matrix T overwrite the corresponding elements of A.</li> <li>■ If uplo = 'L', the diagonal and first subdiagonal of T overwrite the corresponding elements of A.</li> </ul>
w	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n).</p> <p>If info = 0, the eigenvalues in ascending order.</p>
z	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n).</p> <ul style="list-style-type: none"> <li>■ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with w(i).</li> <li>■ If jobz = 'N', then z is not referenced.</li> </ul>
ldz	<p>The leading dimension of the array z. ldz &gt;= 1, and if jobz = 'V', ldz &gt;= max(1, n).</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_SPSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  symmetric matrix stored in packed format, and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

The diagonal pivoting method is used to factor  $A$  as

$$A = U * D * U^{*T}, \text{ if } UPLO = 'U'$$

or

$$A = L * D * L^{*T}, \text{ if } UPLO = 'L'$$

where  $U$  (or  $L$ ) is a product of permutation and unit upper (lower) triangular matrices, and  $D$  is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of  $A$  is then used to solve the system of equations  $A * X = B$ .

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_SPSV (
  uplo   IN      flag,
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  ap     IN OUT  UTL_NLA_ARRAY_DBL,
  ipiv   IN OUT  UTL_NLA_ARRAY_INT,
  b      IN OUT  UTL_NLA_ARRAY_DBL,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_SPSV (
  uplo   IN      flag,
  n      IN      POSITIVEN,
  nrhs   IN      POSITIVEN,
  ap     IN OUT  UTL_NLA_ARRAY_FLT,
  ipiv   IN OUT  UTL_NLA_ARRAY_INT,
  b      IN OUT  UTL_NLA_ARRAY_FLT,
  ldb    IN      POSITIVEN,
  info   OUT     INTEGER,
  pack   IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–55** LAPACK\_SPSV Procedure Parameters

Parameter	Description
uplo	<ul style="list-style-type: none"> <li>■ uplo = 'U'. Upper triangular of <math>A</math> is stored.</li> <li>■ uplo = 'L'. Lower triangular of <math>A</math> is stored.</li> </ul>
n	The number of linear equations, which is the order of the matrix $a$ . $N \geq 0$ .
nrhs	The number of right-hand sides, which is the number of columns of the matrix $b$ . $nrhs \geq 0$ .

**Table 231–55 (Cont.) LAPACK\_SPSV Procedure Parameters**

Parameter	Description
ap	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n*(n+1)/2).</p> <p>On entry, the upper or lower triangle of the symmetric matrix A, packed columnwise in a linear array. The j-th column of A is stored in the array ap as follows:</p> <ul style="list-style-type: none"> <li>■ uplo = 'U': AP(i + (j-1)*j/2) = A(i, j) for 1 ≤ i ≤ j</li> <li>■ uplo = 'L': AP(i + (j-1)*(2n-j)/2) = A(i, j) for j ≤ i ≤ n</li> </ul> <p>See below for further details.</p> <p>On exit, the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization <math>A = U*D*U^T</math> or <math>A = L*D*L^T</math> as computed by SSPTRF, stored as a packed triangular matrix in the same storage format as A.</p>
ipiv	<p>INTEGER array, DIMENSION (n).</p> <p>Details of the interchanges and the block structure of d, as determined by SSPTRF.</p> <ul style="list-style-type: none"> <li>■ If ipiv(k) &gt; 0, then rows and columns k and ipiv(k) were interchanged, and d(k, k) is a 1 by 1 diagonal block.</li> <li>■ If uplo = 'U' and ipiv(k) = ipiv(k-1) &lt; 0, then rows and columns k-1 and -ipiv(k) were interchanged and d(k-1:k, k-1:k) is a 2 by 2 diagonal block.</li> <li>■ If uplo = 'L' and ipiv(k) = ipiv(k+1) &lt; 0, then rows and columns k+1 and -ipiv(k) were interchanged and d(k:k+1, k:k+1) is a 2 by 2 diagonal block.</li> </ul>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs).</p> <p>On entry, the n by nrhs right hand side matrix b.</p> <p>On exit, if info = 0, the n by nrhs solution matrix X.</p>
ldb	<p>The leading dimension of the array b.</p> <p>ldb ≥ max(1, n)</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, d(i, i) is exactly zero. The factorization has been completed, but the block diagonal matrix d is exactly singular, so the solution could not be computed.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>

## LAPACK\_STEV Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix A.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_STEV (
  jobz      IN      flag,
  n         IN      POSITIVEN,
  d         IN OUT  UTL_NLA_ARRAY_DBL,
  e         IN OUT  UTL_NLA_ARRAY_DBL,
  z         IN OUT  UTL_NLA_ARRAY_DBL,
  ldz      IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_STEV (
  jobz      IN      flag,
  n         IN      POSITIVEN,
  d         IN OUT  UTL_NLA_ARRAY_FLT,
  e         IN OUT  UTL_NLA_ARRAY_FLT,
  z         IN OUT  UTL_NLA_ARRAY_FLT,
  ldz      IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–56** *LAPACK\_STEV Procedure Parameters*

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
n	The order of the matrix A. $N \geq 0$ .
d	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). <ul style="list-style-type: none"> <li>▪ On entry, the n diagonal elements of the tridiagonal matrix A.</li> <li>▪ On exit, if info = 0, the eigenvalues in ascending order.</li> </ul>
e	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). <ul style="list-style-type: none"> <li>▪ On entry, the (n-1) subdiagonal elements of the tridiagonal matrix A, stored in elements 1 to n-1 of e. e(n) need not be set, but is used by the subprogram.</li> <li>▪ On exit, the contents of e are destroyed.</li> </ul>
z	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). <ul style="list-style-type: none"> <li>▪ If jobz = 'V', then if info = 0, z contains the orthonormal eigenvectors of the matrix A, with the i-th column of z holding the eigenvector associated with d(i).</li> <li>▪ If jobz = 'N', then z is not referenced.</li> </ul>

**Table 231–56 (Cont.) LAPACK\_STEV Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
ldz	The leading dimension of the array z. $ldz \geq 1$ , and if <code>jobz = 'v'</code> , $ldz \geq \max(1, n)$ .
info	<ul style="list-style-type: none"> <li>▪ = 0 : successful exit</li> <li>▪ &lt; 0 : if <code>info = -i</code>, the <i>i</i>-th argument had an illegal value</li> <li>▪ &gt; 0 : if <code>info = i</code>, the algorithm failed to converge; <i>i</i> off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>▪ 'C': column-major (default)</li> <li>▪ 'R': row-major</li> </ul>



## LAPACK\_STEVD Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_STEVD (
  jobz      IN      flag,
  n         IN      POSITIVEN,
  d         IN OUT  UTL_NLA_ARRAY_DBL,
  e         IN OUT  UTL_NLA_ARRAY_DBL,
  z         IN OUT  UTL_NLA_ARRAY_DBL,
  ldz      IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_STEVD (
  jobz      IN      flag,
  n         IN      POSITIVEN,
  d         IN OUT  UTL_NLA_ARRAY_FLT,
  e         IN OUT  UTL_NLA_ARRAY_FLT,
  z         IN OUT  UTL_NLA_ARRAY_FLT,
  ldz      IN      POSITIVEN,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–57** LAPACK\_STEVD Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
n	The order of the matrix $A$ . $N \geq 0$ .
d	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). <ul style="list-style-type: none"> <li>▪ On entry, the <math>n</math> diagonal elements of the tridiagonal matrix <math>A</math>.</li> <li>▪ On exit, if <code>info = 0</code>, the eigenvalues in ascending order.</li> </ul>
e	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). <ul style="list-style-type: none"> <li>▪ On entry, the <math>(n-1)</math> subdiagonal elements of the tridiagonal matrix <math>A</math>, stored in elements 1 to <math>n-1</math> of <math>e</math>. <math>e(n)</math> need not be set, but is used by the subprogram.</li> <li>▪ On exit, the contents of <math>e</math> are destroyed.</li> </ul>
z	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldz, n). <ul style="list-style-type: none"> <li>▪ If <code>jobz = 'V'</code>, then if <code>info = 0</code>, <math>z</math> contains the orthonormal eigenvectors of the matrix <math>A</math>, with the <math>i</math>-th column of <math>z</math> holding the eigenvector associated with <math>d(i)</math>.</li> <li>▪ If <code>jobz = 'N'</code>, then <math>z</math> is not referenced.</li> </ul>

**Table 231–57 (Cont.) LAPACK\_STEVD Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
ldz	The leading dimension of the array z. $ldz \geq 1$ , and if <code>jobz = 'v'</code> , $ldz \geq \max(1, n)$ .
info	<ul style="list-style-type: none"> <li>▪ = 0 : successful exit</li> <li>▪ &lt; 0 : if <code>info = -i</code>, the <i>i</i>-th argument had an illegal value</li> <li>▪ &gt; 0 : if <code>info = i</code>, the algorithm failed to converge; <i>i</i> off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>▪ 'C': column-major (default)</li> <li>▪ 'R': row-major</li> </ul>

## LAPACK\_SYEV Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_SYEV (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  a         IN OUT  UTL_NLA_ARRAY_DBL,
  lda       IN      POSITIVEN,
  w         IN OUT  UTL_NLA_ARRAY_DBL,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_SYEV (
  jobz      IN      flag,
  uplo      IN      flag,
  n         IN      POSITIVEN,
  a         IN OUT  UTL_NLA_ARRAY_FLT,
  lda       IN      POSITIVEN,
  w         IN OUT  UTL_NLA_ARRAY_FLT,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–58** LAPACK\_SYEV Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>'N': Compute eigenvalues only.</li> <li>'V': Compute eigenvalues and eigenvectors.</li> </ul>
uplo	<ul style="list-style-type: none"> <li>'U': Upper triangle of A is stored.</li> <li>'L': Lower triangle of A is stored.</li> </ul>
n	The order of the matrix a. $N \geq 0$ .
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the symmetric matrix a: <ul style="list-style-type: none"> <li>If uplo = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a.</li> <li>If uplo = 'L', the leading n by n lower triangular part of a contains the lower triangular part of the matrix a.</li> </ul> On exit: <ul style="list-style-type: none"> <li>If jobz = 'V', then if info = 0, a contains the orthonormal eigenvectors of the matrix a.</li> <li>If jobz = 'N', then on exit the lower triangle (if uplo = 'L') or the upper triangle (if uplo = 'U') of a, including the diagonal, is destroyed.</li> </ul>
lda	The leading dimension of the array a. $lda \geq \max(1, n)$ .

**Table 231–58 (Cont.) LAPACK\_SYEV Procedure Parameters**

Parameter	Description
w	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If info = 0, the eigenvalues in ascending order.
info	<ul style="list-style-type: none"> <li>▪ = 0 : successful exit</li> <li>▪ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>▪ &gt; 0 : if info = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
pack	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>▪ 'C': column-major (default)</li> <li>▪ 'R': row-major</li> </ul>

## LAPACK\_SYEVD Procedures

This procedure computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A. If eigenvectors are desired, it uses a divide and conquer algorithm that makes mild assumptions about floating point arithmetic.

**See Also:** [LAPACK Driver Routines \(LLS and Eigenvalue Problems\) Subprograms](#) on page 231-12 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_SYEVD (
  jobz      IN      flag,
  uplo      IN      flag,
  n          IN      POSITIVEN,
  a          IN OUT  UTL_NLA_ARRAY_DBL,
  lda       IN      POSITIVEN,
  w          IN OUT  UTL_NLA_ARRAY_DBL,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_SYEVD (
  jobz      IN      flag,
  uplo      IN      flag,
  n          IN      POSITIVEN,
  a          IN OUT  UTL_NLA_ARRAY_FLT,
  lda       IN      POSITIVEN,
  w          IN OUT  UTL_NLA_ARRAY_FLT,
  info      OUT     INTEGER,
  pack      IN      flag DEFAULT 'C');
```

### Parameters

**Table 231–59** LAPACK\_SYEVD Procedure Parameters

Parameter	Description
jobz	<ul style="list-style-type: none"> <li>▪ 'N': Compute eigenvalues only.</li> <li>▪ 'V': Compute eigenvalues and eigenvectors.</li> </ul>
uplo	<ul style="list-style-type: none"> <li>▪ 'U': Upper triangle of A is stored.</li> <li>▪ 'L': Lower triangle of A is stored.</li> </ul>
n	The order of the matrix a. $N \geq 0$ .
a	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (lda, n). On entry, the symmetric matrix a: <ul style="list-style-type: none"> <li>▪ If uplo = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a.</li> <li>▪ If uplo = 'L', the leading n by n lower triangular part of a contains the lower triangular part of the matrix a.</li> </ul> On exit: <ul style="list-style-type: none"> <li>▪ If jobz = 'V', then if info = 0, a contains the orthonormal eigenvectors of the matrix a.</li> <li>▪ If jobz = 'N', then on exit the lower triangle (if uplo = 'L') or the upper triangle (if uplo = 'U') of a, including the diagonal, is destroyed.</li> </ul>

**Table 231–59 (Cont.) LAPACK\_SYEVD Procedure Parameters**

<b>Parameter</b>	<b>Description</b>
<code>lda</code>	The leading dimension of the array <code>a</code> . <code>lda &gt;= max(1, n)</code> .
<code>w</code>	UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n). If <code>info = 0</code> , the eigenvalues in ascending order.
<code>info</code>	<ul style="list-style-type: none"> <li>▪ <code>= 0</code> : successful exit</li> <li>▪ <code>&lt; 0</code> : if <code>info = -i</code>, the <code>i</code>-th argument had an illegal value</li> <li>▪ <code>&gt; 0</code> : if <code>info = i</code>, the algorithm failed to converge; <code>i</code> off-diagonal elements of an intermediate tridiagonal form did not converge to zero</li> </ul>
<code>pack</code>	(Optional) Flags the packing of the matrices: <ul style="list-style-type: none"> <li>▪ <code>'C'</code>: column-major (default)</li> <li>▪ <code>'R'</code>: row-major</li> </ul>

## LAPACK\_SYSV Procedures

This procedure computes the solution to a real system of linear equations

$$a * x = b$$

where  $a$  is an  $n$  by  $n$  symmetric matrix, and  $x$  and  $b$  are  $n$  by  $nrhs$  matrices.

The diagonal pivoting method is used to factor  $A$  as

$$A = U * D * U^{*T}, \text{ if } UPLO = 'U'$$

or

$$A = L * D * L^{*T}, \text{ if } UPLO = 'L'$$

where  $U$  (or  $L$ ) is a product of permutation and unit upper (lower) triangular matrices, and  $D$  is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The factored form of  $A$  is then used to solve the system of equations  $A * X = B$ .

**See Also:** [LAPACK Driver Routines \(Linear Equations\) Subprograms](#) on page 231-11 for other subprograms in this group

### Syntax

```
UTL_NLA.LAPACK_SYSV (
    uplo    IN        flag,
    n       IN        POSITIVEN,
    nrhs    IN        POSITIVEN,
    a       IN OUT    UTL_NLA_ARRAY_DBL,
    lda     IN        POSITIVEN,
    ipiv    IN OUT    UTL_NLA_ARRAY_INT,
    b       IN OUT    UTL_NLA_ARRAY_DBL,
    ldb     IN        POSITIVEN,
    info    OUT        INTEGER,
    pack    IN        flag DEFAULT 'C');
```

```
UTL_NLA.LAPACK_SYSV (
    uplo    IN        flag,
    n       IN        POSITIVEN,
    nrhs    IN        POSITIVEN,
    a       IN OUT    UTL_NLA_ARRAY_FLT,
    lda     IN        POSITIVEN,
    ipiv    IN OUT    UTL_NLA_ARRAY_INT,
    b       IN OUT    UTL_NLA_ARRAY_FLT,
    ldb     IN        POSITIVEN,
    info    OUT        INTEGER,
    pack    IN        flag DEFAULT 'C');
```

### Parameters

**Table 231–60** LAPACK\_SYSV Procedure Parameters

Parameter	Description
uplo	<ul style="list-style-type: none"> <li>▪ uplo = 'U'. Upper triangular of <math>A</math> is stored.</li> <li>▪ uplo = 'L'. Lower triangular of <math>A</math> is stored.</li> </ul>
n	The number of linear equations, which is the order of the matrix $a$ . $N \geq 0$ .

**Table 231–60 (Cont.) LAPACK\_SYSV Procedure Parameters**

Parameter	Description
nrhs	The number of right-hand sides, which is the number of columns of the matrix b. nrhs $\geq$ 0.
a	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (n-1).</p> <p>On entry, the symmetric matrix a. If UPLO = 'U', the leading n by n upper triangular part of a contains the upper triangular part of the matrix a, and the strictly lower triangular part of a is not referenced. If uplo = 'L', the leading n by n lower triangular part of a contains the lower triangular part of the matrix a, and the strictly upper triangular part of a is not referenced.</p> <p>On exit, if info = 0, the block diagonal matrix d and the multipliers used to obtain the factor U or L from the factorization <math>A = U^*D^*U^{**T}</math> or <math>A = L^*D^*L^{**T}</math> as computed by SSYTRF.</p>
lda	<p>The leading dimension of the array a.</p> <p>lda <math>\geq</math> max(1, n)</p>
ipiv	<p>INTEGER array, DIMENSION (ldb, nrhs).</p> <p>Details of the interchanges and the block structure of d, as determined by SSYTRF.</p> <ul style="list-style-type: none"> <li>■ If ipiv(k) &gt; 0, then rows and columns k and ipiv(k) were interchanged, and d(k, k) is a 1 by 1 diagonal block.</li> <li>■ If uplo = 'U' and ipiv(k) = ipiv(k-1) &lt; 0, then rows and columns k-1 and -ipiv(k) were interchanged and d(k-1:k, k-1:k) is a 2 by 2 diagonal block.</li> <li>■ If uplo = 'L' and ipiv(k) = ipiv(k+1) &lt; 0, then rows and columns k+1 and -ipiv(k) were interchanged and d(k:k+1, k:k+1) is a 2 by 2 diagonal block.</li> </ul>
b	<p>UTL_NLA_ARRAY_FLT/DBL, DIMENSION (ldb, nrhs).</p> <p>On entry, the n by nrhs matrix of right hand side matrix b.</p> <p>On exit, if info = 0, the n by nrhs solution matrix X.</p>
ldb	<p>The leading dimension of the array b.</p> <p>ldb <math>\geq</math> max(1, n)</p>
info	<ul style="list-style-type: none"> <li>■ = 0 : successful exit</li> <li>■ &lt; 0 : if info = -i, the i-th argument had an illegal value</li> <li>■ &gt; 0 : if info = i, d(i, i) is exactly zero. The factorization has been completed, but the block diagonal matrix d is exactly singular, so the solution could not be computed.</li> </ul>
pack	<p>(Optional) Flags the packing of the matrices:</p> <ul style="list-style-type: none"> <li>■ 'C': column-major (default)</li> <li>■ 'R': row-major</li> </ul>



The UTL\_RAW package provides SQL functions for manipulating RAW datatypes.

This chapter contains the following topics:

- [Using UTL\\_RAW](#)
  - Overview
  - Operational Notes
- [Summary of UTL\\_RAW Subprograms](#)

## Using UTL\_RAW

- [Overview](#)
- [Operational Notes](#)

## Overview

This package is necessary because normal SQL functions do not operate on RAWs, and PL/SQL does not allow overloading between a RAW and a CHAR datatype. UTL\_RAW also includes subprograms that convert various COBOL number formats to, and from, RAWs.

UTL\_RAW is not specific to the database environment, and it may actually be used in other environments. For this reason, the prefix UTL has been given to the package, instead of DBMS.

## Operational Notes

UTL\_RAW allows a RAW "record" to be composed of many elements. By using the RAW datatype, character set conversion will not be performed, keeping the RAW in its original format when being transferred through remote procedure calls.

With the RAW functions, you can manipulate binary data that was previously limited to the `hextoraw` and `rawtohex` functions.

---

---

**Note:** Notes on data types:

- The `PLS_INTEGER` and `BINARY_INTEGER` data types are identical. This document uses `BINARY_INTEGER` to indicate data types in reference information (such as for table types, record types, subprogram parameters, or subprogram return values), but may use either in discussion and examples.
  - The `INTEGER` and `NUMBER(38)` data types are also identical. This document uses `INTEGER` throughout.
- 
-

---

## Summary of UTL\_RAW Subprograms

**Table 232–1 UTL\_RAW Package Subprograms**

Subprogram	Description
<a href="#">BIT_AND Function</a> on page 232-7	Performs bitwise logical "and" of the values in RAW <i>r1</i> with RAW <i>r2</i> and returns the "anded" result RAW
<a href="#">BIT_COMPLEMENT Function</a> on page 232-8	Performs bitwise logical "complement" of the values in RAW <i>r</i> and returns the "complement'ed" result RAW
<a href="#">BIT_OR Function</a> on page 232-9	Performs bitwise logical "or" of the values in RAW <i>r1</i> with RAW <i>r2</i> and returns the "or'd" result RAW
<a href="#">BIT_XOR Function</a> on page 232-10	Performs bitwise logical "exclusive or" of the values in RAW <i>r1</i> with RAW <i>r2</i> and returns the "xor'd" result RAW
<a href="#">CAST_FROM_BINARY_DOUBLE Function</a> on page 232-11	Returns the RAW binary representation of a BINARY_DOUBLE value
<a href="#">CAST_FROM_BINARY_FLOAT Function</a> on page 232-13	Returns the RAW binary representation of a BINARY_FLOAT value
<a href="#">CAST_FROM_BINARY_INTEGER Function</a> on page 232-14	Returns the RAW binary representation of a BINARY_INTEGER value
<a href="#">CAST_FROM_NUMBER Function</a> on page 232-15	Returns the RAW binary representation of a NUMBER value
<a href="#">CAST_TO_BINARY_DOUBLE Function</a> on page 232-16	Casts the RAW binary representation of a BINARY_DOUBLE into a BINARY_DOUBLE
<a href="#">CAST_TO_BINARY_FLOAT Function</a> on page 232-18	Casts the RAW binary representation of a BINARY_FLOAT into a BINARY_FLOAT
<a href="#">CAST_TO_BINARY_INTEGER Function</a> on page 232-20	Casts the RAW binary representation of a BINARY_INTEGER into a BINARY_INTEGER
<a href="#">CAST_TO_NUMBER Function</a> on page 232-21	Casts the RAW binary representation of a NUMBER into a NUMBER
<a href="#">CAST_TO_NVARCHAR2 Function</a> on page 232-22	Converts a RAW value into a NVARCHAR2 value
<a href="#">CAST_TO_RAW Function</a> on page 232-23	Converts a RAW value into an NVARCHAR2 value
<a href="#">CAST_TO_VARCHAR2 Function</a> on page 232-24	Converts a VARCHAR2 value into a RAW value
<a href="#">COMPARE Function</a> on page 232-25	Compares RAW <i>r1</i> against RAW <i>r2</i>
<a href="#">CONCAT Function</a> on page 232-26	Concatenates up to 12 RAWs into a single RAW
<a href="#">CONVERT Function</a> on page 232-27	Converts RAW <i>r</i> from character set <i>from_charset</i> to character set <i>to_charset</i> and returns the resulting RAW
<a href="#">COPIES Function</a> on page 232-28	Returns <i>n</i> copies of <i>r</i> concatenated together
<a href="#">LENGTH Function</a> on page 232-29	Returns the length in bytes of a RAW <i>r</i>

**Table 232-1 (Cont.) UTL\_RAW Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">OVERLAY Function</a> on page 232-30	Overlays the specified portion of target RAW with overlay RAW, starting from byte position <code>pos</code> of target and proceeding for <code>len</code> bytes
<a href="#">REVERSE Function</a> on page 232-32	Reverses a byte sequence in RAW <code>r</code> from end to end
<a href="#">SUBSTR Function</a> on page 232-33	Returns <code>len</code> bytes, starting at <code>pos</code> from RAW <code>r</code>
<a href="#">TRANSLATE Function</a> on page 232-35	Translates the bytes in the input RAW <code>r</code> according to the bytes in the translation RAWs <code>from_set</code> and <code>to_set</code>
<a href="#">TRANSLITERATE Function</a> on page 232-37	Converts the bytes in the input RAW <code>r</code> according to the bytes in the transliteration RAWs <code>from_set</code> and <code>to_set</code>
<a href="#">XRANGE Function</a> on page 232-39	Returns a RAW containing all valid 1-byte encodings in succession, beginning with the value <code>start_byte</code> and ending with the value <code>end_byte</code>

## BIT\_AND Function

This function performs bitwise logical "and" of the values in RAW *r1* with RAW *r2* and returns the "anded" result RAW.

### Syntax

```
UTL_RAW.BIT_AND (
    r1 IN RAW,
    r2 IN RAW)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(bit_and, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–2 BIT\_AND Function Parameters**

Parameter	Description
<i>r1</i>	RAW to "and" with <i>r2</i>
<i>r2</i>	RAW to "and" with <i>r1</i>

### Return Values

**Table 232–3 BIT\_AND Function Return Values**

Return	Description
RAW	Containing the "and" of <i>r1</i> and <i>r2</i>
NULL	Either <i>r1</i> or <i>r2</i> input parameter was NULL

### Usage Notes

If *r1* and *r2* differ in length, the and operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

## BIT\_COMPLEMENT Function

This function performs bitwise logical "complement" of the values in RAW *r* and returns the complement'ed result RAW. The result length equals the input RAW *r* length.

### Syntax

```
UTL_RAW.BIT_COMPLEMENT (  
  r IN RAW)  
  RETURN RAW;
```

### Pragmas

```
pragma restrict_references(bit_complement, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–4 BIT\_COMPLEMENT Function Parameters**

Parameter	Description
<i>r</i>	RAW to perform "complement" operation

### Return Values

**Table 232–5 BIT\_COMPLEMENT Function Return Values**

Return	Description
RAW	The "complement" of <i>r1</i>
NULL	If <i>r</i> input parameter was NULL



## BIT\_OR Function

This function performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the or'd result RAW.

### Syntax

```
UTL_RAW.BIT_OR (
  r1 IN RAW,
  r2 IN RAW)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(bit_or, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–6 BIT\_OR Function Parameters**

Parameters	Description
r1	RAW to "or" with r2
r2	RAW to "or" with r1

### Return Values

**Table 232–7 BIT\_OR Function Return Values**

Return	Description
RAW	Containing the "or" of r1 and r2
NULL	Either r1 or r2 input parameter was NULL

### Usage Notes

If r1 and r2 differ in length, then the "or" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

## BIT\_XOR Function

This function performs bitwise logical "exclusive or" of the values in RAW *r1* with RAW *r2* and returns the xor'd result RAW.

### Syntax

```
UTL_RAW.BIT_XOR (  
    r1 IN RAW,  
    r2 IN RAW)  
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(bit_xor, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–8 BIT\_XOR Function Parameters**

Parameter	Description
<i>r1</i>	RAW to "xor" with <i>r2</i>
<i>r2</i>	RAW to "xor" with <i>r1</i>

### Return Values

**Table 232–9 BIT\_XOR Function Return Values**

Return	Description
RAW	Containing the "xor" of <i>r1</i> and <i>r2</i>
NULL	If either <i>r1</i> or <i>r2</i> input parameter was NULL

### Usage Notes

If *r1* and *r2* differ in length, then the "xor" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

## CAST\_FROM\_BINARY\_DOUBLE Function

This function returns the RAW binary representation of a BINARY\_DOUBLE value.

### Syntax

```
UTL_RAW.CAST_FROM_BINARY_DOUBLE(
  n          IN BINARY_DOUBLE,
  endianness IN PLS_INTEGER DEFAULT 1)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(cast_from_binary_double, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–10 CAST\_FROM\_BINARY\_DOUBLE Function Parameters**

Parameter	Description
n	BINARY_DOUBLE value
endianness	A BINARY_INTEGER value indicating the endianness. The function recognizes the defined constants <code>big_endian</code> (1), <code>little_endian</code> (2), and <code>machine_endian</code> (3). The default is <code>big_endian</code> . A setting of <code>machine_endian</code> has the same effect as <code>big_endian</code> on a big endian machine, or the same effect as <code>little_endian</code> on a little endian machine.

### Return Values

The binary representation of the BINARY\_DOUBLE value, or NULL if the input is NULL.

### Usage Notes

- An 8-byte `binary_double` value maps to the IEEE 754 double-precision format as follows:
  - byte 0: bit 63 ~ bit 56
  - byte 1: bit 55 ~ bit 48
  - byte 2: bit 47 ~ bit 40
  - byte 3: bit 39 ~ bit 32
  - byte 4: bit 31 ~ bit 24
  - byte 5: bit 23 ~ bit 16
  - byte 6: bit 15 ~ bit 8
  - byte 7: bit 7 ~ bit 0
- The parameter `endianness` describes how the bytes of BINARY\_DOUBLE are mapped to the bytes of RAW. In the following matrix, `rb0 ~ rb7` refer to the bytes in raw and `db0 ~ db7` refer to the bytes in BINARY\_DOUBLE.

	rb0	rb1	rb2	rb3	rb4	rb5	rb6	rb7
<b>big_endian</b>	db0	db1	db2	db3	db4	db5	db6	db7
<b>little_endian</b>	db7	db6	db5	db4	db3	db2	db1	db0

- In case of machine-endian, the 8 bytes of the BINARY\_DOUBLE argument are copied straight across into the RAW return value. The effect is the same if the user

has passed `big_endian` on a big-endian machine, or `little_endian` on a little-endian machine.

## CAST\_FROM\_BINARY\_FLOAT Function

This function returns the RAW binary representation of a BINARY\_FLOAT value.

### Syntax

```
UTL_RAW.CAST_FROM_BINARY_FLOAT(
  n          IN BINARY_FLOAT,
  endianness IN PLS_INTEGER DEFAULT 1)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(cast_from_binary_float, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–11 CAST\_FROM\_BINARY\_FLOAT Function Parameters**

Parameter	Description
n	BINARY_FLOAT value
endianness	A BINARY_INTEGER value indicating the endianness. The function recognizes the defined constants <code>big_endian</code> (1), <code>little_endian</code> (2), and <code>machine_endian</code> (3). The default is <code>big_endian</code> . A setting of <code>machine_endian</code> has the same effect as <code>big_endian</code> on a big endian machine, or the same effect as <code>little_endian</code> on a little endian machine.

### Return Values

The binary representation (RAW) of the BINARY\_FLOAT value, or NULL if the input is NULL.

### Usage Notes

- A 4-byte `binary_float` value maps to the IEEE 754 single-precision format as follows:
  - byte 0: bit 31 ~ bit 24
  - byte 1: bit 23 ~ bit 16
  - byte 2: bit 15 ~ bit 8
  - byte 3: bit 7 ~ bit 0
- The parameter `endianness` describes how the bytes of BINARY\_FLOAT are mapped to the bytes of RAW. In the following matrix, `rb0 ~ rb3` refer to the bytes in RAW and `fb0 ~ fb3` refer to the bytes in BINARY\_FLOAT.

	rb0	rb1	rb2	rb3
big_endian	fb0	fb1	fb2	fb3
little_endian	fb3	fb2	fb1	fb0

- In case of `machine_endian`, the 4 bytes of the BINARY\_FLOAT argument are copied straight across into the RAW return value. The effect is the same if the user has passed `big_endian` on a big-endian machine, or `little_endian` on a little-endian machine.

## CAST\_FROM\_BINARY\_INTEGER Function

This function returns the RAW binary representation of a BINARY\_INTEGER value.

### Syntax

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (
    n          IN BINARY_INTEGER
    endianness IN PLS_INTEGER DEFAULT BIG_ENDIAN)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(cast_from_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–12 CAST\_FROM\_BINARY\_INTEGER Function Parameters**

Parameter	Description
n	BINARY_INTEGER value.
endianness	A BINARY_INTEGER value indicating the endianness. The function recognizes the defined constants <code>big_endian</code> (1), <code>little_endian</code> (2), and <code>machine_endian</code> (3). The default is <code>big_endian</code> . A setting of <code>machine_endian</code> has the same effect as <code>big_endian</code> on a big endian machine, or the same effect as <code>little_endian</code> on a little endian machine.

### Return Values

The binary representation of the BINARY\_INTEGER value.

## CAST\_FROM\_NUMBER Function

This function returns the RAW binary representation of a NUMBER value.

### Syntax

```
UTL_RAW.CAST_FROM_NUMBER (  
    n IN NUMBER)  
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(cast_from_number, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–13** *CAST\_FROM\_NUMBER Function Parameters*

Parameter	Description
n	NUMBER value

### Return Values

The binary representation of the NUMBER value.

## CAST\_TO\_BINARY\_DOUBLE Function

This function casts the RAW binary representation of a BINARY\_DOUBLE into a BINARY\_DOUBLE.

### Syntax

```
UTL_RAW.CAST_TO_BINARY_DOUBLE (
    r          IN RAW
    endianness IN PLS_INTEGER DEFAULT 1)
RETURN BINARY_DOUBLE;
```

### Pragmas

```
pragma restrict_references(cast_to_binary_double, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–14 CAST\_TO\_BINARY\_DOUBLE Function Parameters**

Parameter	Description
r	Binary representation of a BINARY_DOUBLE
endianness	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

### Return Values

The BINARY\_DOUBLE value.

### Usage Notes

- If the RAW argument is more than 8 bytes, only the first 8 bytes are used and the rest of the bytes are ignored. If the result is  $-0$ ,  $+0$  is returned. If the result is NaN, the value BINARY\_DOUBLE\_NAN is returned.
- If the RAW argument is less than 8 bytes, a VALUE\_ERROR exception is raised.
- An 8-byte binary\_double value maps to the IEEE 754 double-precision format as follows:

```
byte 0: bit 63 ~ bit 56
byte 1: bit 55 ~ bit 48
byte 2: bit 47 ~ bit 40
byte 3: bit 39 ~ bit 32
byte 4: bit 31 ~ bit 24
byte 5: bit 23 ~ bit 16
byte 6: bit 15 ~ bit 8
byte 7: bit 7 ~ bit 0
```

- The parameter endianness describes how the bytes of BINARY\_DOUBLE are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb7 refer to the bytes in raw and db0 ~ db7 refer to the bytes in BINARY\_DOUBLE.

	rb0	rb1	rb2	rb3	rb4	rb5	rb6	rb7
<b>big_endian</b>	db0	db1	db2	db3	db4	db5	db6	db7
<b>little_endian</b>	db7	db6	db5	db4	db3	db2	db1	db0



- In case of machine-endian, the 8 bytes of the RAW argument are copied straight across into the BINARY\_DOUBLE return value. The effect is the same if the user has passed `big_endian` on a big-endian machine, or `little_endian` on a little-endian machine.

## CAST\_TO\_BINARY\_FLOAT Function

This function casts the RAW binary representation of a BINARY\_FLOAT into a BINARY\_FLOAT.

### Syntax

```
UTL_RAW.CAST_TO_BINARY_FLOAT (
    r          IN RAW
    endianness IN PLS_INTEGER DEFAULT 1)
RETURN BINARY_FLOAT;
```

### Pragmas

```
pragma restrict_references(cast_to_binary_float, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–15 CAST\_TO\_BINARY\_FLOAT Function Parameters**

Parameter	Description
r	Binary representation of a BINARY_FLOAT
endianness	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

### Return Values

The BINARY\_FLOAT value.

### Usage Notes

- If the RAW argument is more than 4 bytes, only the first 4 bytes are used and the rest of the bytes are ignored. If the result is -0, +0 is returned. If the result is NaN, the value BINARY\_FLOAT\_NAN is returned.
- If the RAW argument is less than 4 bytes, a VALUE\_ERROR exception is raised.
- A 4-byte binary\_float value maps to the IEEE 754 single-precision format as follows:
  - byte 0: bit 31 ~ bit 24
  - byte 1: bit 23 ~ bit 16
  - byte 2: bit 15 ~ bit 8
  - byte 3: bit 7 ~ bit 0
- The parameter endianness describes how the bytes of BINARY\_FLOAT are mapped to the bytes of RAW. In the following matrix, rb0 ~ rb3 refer to the bytes in RAW and fb0 ~ fb3 refer to the bytes in BINARY\_FLOAT.

	rb0	rb1	rb2	rb3
<b>big_endian</b>	fb0	fb1	fb2	fb3
<b>little_endian</b>	fb3	fb2	fb1	fb0

- In case of machine-endian, the 4 bytes of the RAW argument are copied straight across into the BINARY\_FLOAT return value. The effect is the same if the user has

passed `big_endian` on a big-endian machine, or `little_endian` on a little-endian machine.

## CAST\_TO\_BINARY\_INTEGER Function

This function casts the RAW binary representation of a BINARY\_INTEGER into a BINARY\_INTEGER.

### Syntax

```
UTL_RAW.CAST_TO_BINARY_INTEGER (  
    r          IN RAW  
    endianness IN PLS_INTEGER DEFAULT BIG_ENDIAN)  
RETURN BINARY_INTEGER;
```

### Pragmas

```
pragma restrict_references(cast_to_binary_integer, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–16** *CAST\_TO\_BINARY\_INTEGER Function Parameters*

Parameter	Description
r	Binary representation of a BINARY_INTEGER
endianness	A PLS_INTEGER representing big-endian or little-endian architecture. The default is big-endian.

### Return Values

The BINARY\_INTEGER value

## CAST\_TO\_NUMBER Function

This function casts the RAW binary representation of a NUMBER into a NUMBER.

### Syntax

```
UTL_RAW.CAST_TO_NUMBER (  
    r IN RAW)  
RETURN NUMBER;
```

### Pragmas

```
pragma restrict_references(cast_to_number, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–17** *CAST\_TO\_NUMBER function Parameters*

Parameter	Description
r	Binary representation of a NUMBER

### Return Values

The NUMBER value.

## CAST\_TO\_NVARCHAR2 Function

This function converts a RAW value represented using some number of data bytes into an NVARCHAR2 value with that number of data bytes.

---

---

**Note:** When casting to a NVARCHAR2, the current Globalization Support character set is used for the characters within that NVARCHAR2 value.

---

---

### Syntax

```
UTL_RAW.CAST_TO_NVARCHAR2 (  
    r IN RAW)  
RETURN NVARCHAR2;
```

### Pragmas

```
pragma restrict_references(cast_to_NVARCHAR2, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–18** *CAST\_TO\_NVARCHAR2 Function Parameters*

Parameter	Description
r	RAW (without leading length field) to be changed to a NVARCHAR2

### Return Values

**Table 232–19** *CAST\_TO\_NVARCHAR2 Function Return Values*

Return	Description
NVARCHAR2	Containing having the same data as the input RAW
NULL	If r input parameter was NULL

## CAST\_TO\_RAW Function

This function converts a VARCHAR2 value represented using some number of data bytes into a RAW value with that number of data bytes. The data itself is not modified in any way, but its data type is recast to a RAW data type.

### Syntax

```
UTL_RAW.CAST_TO_RAW (
    c IN VARCHAR2)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(cast_to_raw, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–20 CAST\_TO\_RAW Function Parameters**

Parameter	Description
c	VARCHAR2 to be changed to a RAW

### Return Values

**Table 232–21 CAST\_TO\_RAW Function Return Values**

Return	Description
RAW	Containing the same data as the input VARCHAR2 and equal byte length as the input VARCHAR2 and without a leading length field
NULL	If c input parameter was NULL

## CAST\_TO\_VARCHAR2 Function

This function converts a RAW value represented using some number of data bytes into a VARCHAR2 value with that number of data bytes.

---

---

**Note:** When casting to a VARCHAR2, the current Globalization Support character set is used for the characters within that VARCHAR2.

---

---

### Syntax

```
UTL_RAW.CAST_TO_VARCHAR2 (  
    r IN RAW)  
RETURN VARCHAR2;
```

### Pragmas

```
pragma restrict_references(cast_to_VARCHAR2, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–22** CAST\_TO\_VARCHAR2 Function Parameters

Parameter	Description
r	RAW (without leading length field) to be changed to a VARCHAR2

### Return Values

**Table 232–23** CAST\_TO\_VARCHAR2 Function Return Values

Return	Description
VARCHAR2	Containing having the same data as the input RAW
NULL	If r input parameter was NULL



## COMPARE Function

This function compares two RAW values. If they differ in length, then the shorter is extended on the right according to the optional pad parameter.

### Syntax

```
UTL_RAW.COMPARE (
  r1  IN RAW,
  r2  IN RAW,
  pad IN RAW DEFAULT NULL)
RETURN NUMBER;
```

### Pragmas

```
pragma restrict_references(compare, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–24 COMPARE Function Parameters**

Parameter	Description
r1	1st RAW to be compared, may be NULL or 0 length
r2	2nd RAW to be compared, may be NULL or 0 length
pad	This is an optional parameter. Byte to extend whichever of r1 or r2 is shorter. The default: x'00'

### Return Values

**Table 232–25 COMPARE Function Return Values**

Return	Description
NUMBER	Equals 0 if RAW byte strings are both NULL or identical; or, Equals position (numbered from 1) of the first mismatched byte

## CONCAT Function

This function concatenates up to 12 RAWs into a single RAW. If the concatenated size exceeds 32K, then an error is returned

### Syntax

```
UTL_RAW.CONCAT (  
    r1 IN RAW DEFAULT NULL,  
    r2 IN RAW DEFAULT NULL,  
    r3 IN RAW DEFAULT NULL,  
    r4 IN RAW DEFAULT NULL,  
    r5 IN RAW DEFAULT NULL,  
    r6 IN RAW DEFAULT NULL,  
    r7 IN RAW DEFAULT NULL,  
    r8 IN RAW DEFAULT NULL,  
    r9 IN RAW DEFAULT NULL,  
    r10 IN RAW DEFAULT NULL,  
    r11 IN RAW DEFAULT NULL,  
    r12 IN RAW DEFAULT NULL)  
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(concat, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

r1....r12 are the RAW items to concatenate.

### Return Values

**Table 232–26** *CONCAT Function Return Values*

Return	Description
RAW	Containing the items concatenated

### Exceptions

There is an error if the sum of the lengths of the inputs exceeds the maximum allowable length for a RAW, which is 32767 bytes.

## CONVERT Function

This function converts RAW *r* from character set *from\_charset* to character set *to\_charset* and returns the resulting RAW.

Both *from\_charset* and *to\_charset* must be supported character sets defined to the Oracle server.

### Syntax

```
UTL_RAW.CONVERT (
  r           IN RAW,
  to_charset  IN VARCHAR2,
  from_charset IN VARCHAR2)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(convert, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–27** *CONVERT Function Parameters*

Parameter	Description
<i>r</i>	RAW byte-string to be converted
<i>to_charset</i>	Name of Globalization Support character set to which <i>r</i> is converted
<i>from_charset</i>	Name of Globalization Support character set in which <i>r</i> is supplied

### Return Values

**Table 232–28** *CONVERT Function Return Values*

Return	Description
RAW	Byte string <i>r</i> converted according to the specified character sets.

### Exceptions

**Table 232–29** *CONVERT Function Optional Parameter*

Error	Description
VALUE_ERROR	Either: <ul style="list-style-type: none"> <li>- <i>r</i> missing, NULL, or 0 length</li> <li>- <i>from_charset</i> or <i>to_charset</i> missing, NULL, or 0 length</li> <li>- <i>from_charset</i> or <i>to_charset</i> names invalid or unsupported</li> </ul>

## COPIES Function

This function returns *n* copies of *r* concatenated together.

### Syntax

```
UTL_RAW.COPIES (  
  r IN RAW,  
  n IN NUMBER)  
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(copies, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–30 COPIES Function Parameters**

Parameters	Description
<i>r</i>	RAW to be copied
<i>n</i>	Number of times to copy the RAW (must be positive)

### Return Values

This returns the RAW copied *n* times.

### Exceptions

**Table 232–31 COPIES Function Exceptions**

Error	Description
VALUE_ERROR	Either: - <i>r</i> is missing, NULL or 0 length - <i>n</i> < 1 - Length of result exceeds maximum length of a RAW

## LENGTH Function

This function returns the length in bytes of a RAW *r*.

### Syntax

```
UTL_RAW.LENGTH (  
    r IN RAW)  
RETURN NUMBER;
```

### Pragmas

```
pragma restrict_references(length, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–32** LENGTH Function Parameters

Parameter	Description
<i>r</i>	RAW byte stream to be measured

### Return Values

**Table 232–33** LENGTH Function Return Values

Return	Description
NUMBER	Current length of the RAW

## OVERLAY Function

This function overlays the specified portion of target RAW with `overlay_str` RAW, starting from byte position `pos` of target and proceeding for `len` bytes.

### Syntax

```
UTL_RAW.OVERLAY (
  overlay_str IN RAW,
  target      IN RAW,
  pos         IN BINARY_INTEGER DEFAULT 1,
  len         IN BINARY_INTEGER DEFAULT NULL,
  pad         IN RAW              DEFAULT NULL)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(overlay, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–34 OVERLAY Function Parameters**

Parameters	Description
<code>overlay_str</code>	Byte-string used to overlay target
<code>target</code>	Byte-string which is to be overlaid
<code>pos</code>	Position in target (numbered from 1) to start overlay
<code>len</code>	The number of target bytes to overlay
<code>pad</code>	Pad byte used when overlay <code>len</code> exceeds <code>overlay_str</code> length or <code>pos</code> exceeds target length

### Defaults and Optional Parameters

**Table 232–35 OVERLAY Function Optional Parameters**

Optional Parameter	Description
<code>pos</code>	1
<code>len</code>	To the length of <code>overlay_str</code>
<code>pad</code>	x'00'

### Return Values

**Table 232–36 OVERLAY Function Return Values**

Return	Description
RAW	The target byte_string overlaid as specified.

### Usage Notes

If `overlay_str` has less than `len` bytes, then it is extended to `len` bytes using the `pad` byte. If `overlay_str` exceeds `len` bytes, then the extra bytes in `overlay_str` are ignored. If `len` bytes beginning at position `pos` of target exceeds the length of target, then target is extended to contain the entire length of `overlay_str`.

If `len` is specified, it must be greater than or equal to 0. If `pos` is specified, it must be greater than or equal to 1. If `pos` exceeds the length of `target`, then `target` is padded with `pad` bytes to position `pos`, and `target` is further extended with `overlay_str` bytes.

## Exceptions

**Table 232-37** *OVERLAY Function Exceptions*

<b>Error</b>	<b>Description</b>
VALUE_ERROR	Either: <ul style="list-style-type: none"><li>- <code>Overlay_str</code> is NULL or has 0 length</li><li>- <code>Target</code> is missing or undefined</li><li>- Length of <code>target</code> exceeds maximum length of a RAW</li><li>- <code>len</code> &lt; 0</li><li>- <code>pos</code> &lt; 1</li></ul>

## REVERSE Function

This function reverses a byte sequence in RAW *r* from end to end. For example, x'0102F3' would be reversed to x'F30201', and 'xyz' would be reversed to 'zyx'. The result length is the same as the input RAW length.

### Syntax

```
UTL_RAW.REVERSE (  
    r IN RAW)  
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(reverse, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–38 REVERSE Function Parameters**

Parameter	Description
<i>r</i>	RAW to reverse

### Return Values

**Table 232–39 REVERSE Function Return Values**

Return	Description
RAW	Containing the "reverse" of <i>r</i>

### Exceptions

**Table 232–40 REVERSE Function Exceptions**

Error	Description
VALUE_ERROR	R is NULL or has 0 length



## SUBSTR Function

This function returns `len` bytes, starting at `pos` from RAW `r`.

### Syntax

```
UTL_RAW.SUBSTR (
  r    IN RAW,
  pos  IN BINARY_INTEGER,
  len  IN BINARY_INTEGER DEFAULT NULL)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(substr, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–41 SUBSTR Function Parameters**

Parameter	Description
<code>r</code>	The RAW byte-string from which a portion is extracted
<code>pos</code>	The byte position in <code>r</code> at which to begin extraction
<code>len</code>	The number of bytes from <code>pos</code> to extract from <code>r</code> (optional)

### Defaults and Optional Parameters

**Table 232–42 SUBSTR Function Exceptions**

Optional Parameter	Description
<code>len</code>	Position <code>pos</code> through to the end of <code>r</code>

### Return Values

**Table 232–43 SUBSTR Function Return Values**

Return	Description
portion of <code>r</code>	Beginning at <code>pos</code> for <code>len</code> bytes long
NULL	<code>r</code> input parameter was NULL

### Usage Notes

If `pos` is positive, then SUBSTR counts from the beginning of `r` to find the first byte. If `pos` is negative, then SUBSTR counts backward from the end of the `r`. The value `pos` cannot be 0.

If `len` is omitted, then SUBSTR returns all bytes to the end of `r`. The value `len` cannot be less than 1.

## Exceptions

**Table 232-44** *SUBSTR Function Exceptions*

<b>Error</b>	<b>Description</b>
VALUE_ERROR	VALUE_ERROR is returned if: <ul style="list-style-type: none"><li>pos = 0 or &gt; length of r</li><li>len &lt; 1 or &gt; length of r - (pos-1)</li></ul>

## TRANSLATE Function

This function translates the bytes in the input RAW *r* according to the bytes in the translation RAWs *from\_set* and *to\_set*. If a byte in *r* has a matching byte in *from\_set*, then it is replaced by the byte in the corresponding position in *to\_set*, or deleted.

Bytes in *r*, but undefined in *from\_set*, are copied to the result. Only the first (leftmost) occurrence of a byte in *from\_set* is used. Subsequent duplicates are not scanned and are ignored.

### Syntax

```
UTL_RAW.TRANSLATE (
  r          IN RAW,
  from_set  IN RAW,
  to_set    IN RAW)
RETURN RAW;
```

---

**Note:** Be aware that *to\_set* and *from\_set* are reversed in the calling sequence compared to TRANSLITERATE.

---

### Pragmas

```
pragma restrict_references(translate, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–45 TRANSLATE Function Parameters**

Parameter	Description
<i>r</i>	RAW source byte-string to be translated
<i>from_set</i>	RAW byte-codes to be translated, if present in <i>r</i>
<i>to_set</i>	RAW byte-codes to which corresponding <i>from_str</i> bytes are translated

### Return Values

**Table 232–46 TRANSLATE Function Return Values**

Return	Description
RAW	Translated byte-string

### Usage Notes

- If *to\_set* is shorter than *from\_set*, the extra *from\_set* bytes have no corresponding translation bytes. Bytes from the input RAW that match any such *from\_set* bytes are not translated or included in the result. They are effectively translated to NULL.
- If *to\_set* is longer than *from\_set*, the extra *to\_set* bytes are ignored.
- If a byte value is repeated in *from\_set*, the repeated occurrence is ignored.

---

---

**Note:** Differences from the [TRANSLITERATE Function](#):

- The `from_set` parameter comes before the `to_set` parameter in the calling sequence.
- Bytes from `r` that appear in `from_set` but have no corresponding values in `to_set` are not translated or included in the result.
- The resulting RAW value may be shorter than the input RAW value.

Note that `TRANSLATE` and `TRANSLITERATE` only differ in functionality when `to_set` has fewer bytes than `from_set`.

---

---

## Exceptions

**Table 232–47** *TRANSLATE Function Exceptions*

Error	Description
VALUE_ERROR	Either: <ul style="list-style-type: none"><li>- <code>r</code> is NULL or has 0 length</li><li>- <code>from_set</code> is NULL or has 0 length</li><li>- <code>to_set</code> is NULL or has 0 length</li></ul>

## TRANSLITERATE Function

This function converts the bytes in the input RAW *r* according to the bytes in the transliteration RAWs *from\_set* and *to\_set*. Successive bytes in *r* are looked up in the *from\_set*, and, if not found, copied unaltered to the result RAW. If found, then they are replaced in the result RAW by either corresponding bytes in the *to\_set*, or the *pad* byte when no correspondence exists.

Bytes in *r*, but undefined in *from\_set*, are copied to the result. Only the first (leftmost) occurrence of a byte in *from\_set* is used. Subsequent duplicates are not scanned and are ignored. The result RAW is always the same length as *r*.

### Syntax

```
UTL_RAW.TRANSLITERATE (
  r          IN RAW,
  to_set     IN RAW DEFAULT NULL,
  from_set   IN RAW DEFAULT NULL,
  pad        IN RAW DEFAULT NULL)
RETURN RAW;
```

---

**Note:** Be aware that *to\_set* and *from\_set* are reversed in the calling sequence compared to TRANSLATE.

---

### Pragmas

```
pragma restrict_references(transliterate, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–48** *TRANSLITERATE Function Parameters*

Parameter	Description
<i>r</i>	RAW input byte-string to be converted
<i>to_set</i>	RAW byte-codes to which corresponding <i>from_set</i> bytes are converted (any length)
<i>from_set</i>	RAW byte-codes to be converted, if presenting <i>r</i> (any length)
<i>pad</i>	1 byte used when <i>to_set</i> is shorter than the <i>from_set</i>

### Defaults and Optional Parameters

**Table 232–49** *TRANSLITERATE Function Optional Parameters*

Optional Parameter	Description
<i>to_set</i>	To the NULL string and effectively extended with <i>pad</i> to the length of <i>from_set</i> as necessary
<i>from_set</i>	x'00' through x'fff'
<i>pad</i>	x'00'

## Return Values

**Table 232–50** *TRANSLITERATE Function Return Values*

Return	Description
RAW	Converted byte-string.

## Usage Notes

- If `to_set` is shorter than `from_set`, the extra `from_set` bytes have no corresponding conversion bytes. Bytes from the input `RAW` that match any such `from_set` bytes are converted in the result to the pad byte instead.
- If `to_set` is longer than `from_set`, the extra `to_set` bytes are ignored.
- If a byte value is repeated in `from_set`, the repeated occurrence is ignored.

---

**Note:** Differences from the [TRANSLATE Function](#):

- The `to_set` parameter comes before the `from_set` parameter in the calling sequence.
- Bytes from `r` that appear in `from_set` but have no corresponding values in `to_set` are replaced by pad in the result.
- The resulting `RAW` value always has the same length as the input `RAW` value.

Note that `TRANSLATE` and `TRANSLITERATE` only differ in functionality when `to_set` has fewer bytes than `from_set`.

---

## Exceptions

**Table 232–51** *TRANSLITERATE Function Exceptions*

Error	Description
VALUE_ERROR	R is NULL or has 0 length

## XRANGE Function

This function returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-codes. The specified byte-codes must be single-byte RAW values. If the `start_byte` value is greater than the `end_byte` value, then the succession of resulting bytes begins with `start_byte`, wraps through `x'FF'` back to `x'00'`, then ends at `end_byte`.

### Syntax

```
UTL_RAW.XRANGE (
  start_byte IN RAW DEFAULT NULL,
  end_byte   IN RAW DEFAULT NULL)
RETURN RAW;
```

### Pragmas

```
pragma restrict_references(xrange, WNDS, RNDS, WNPS, RNPS);
```

### Parameters

**Table 232–52 XRANGE Function Parameters**

Parameters	Description
<code>start_byte</code>	Beginning byte-code value of resulting sequence. The default is <code>x'00'</code> .
<code>end_byte</code>	Ending byte-code value of resulting sequence. The default is <code>x'FF'</code> .

### Return Values

**Table 232–53 XRANGE Function Return Values**

Return	Description
RAW	Containing succession of 1-byte hexadecimal encodings





The UTL\_RECOMP package recompiles invalid PL/SQL modules, invalid views, Java classes, indextypes and operators in a database, either sequentially or in parallel.

This chapter contains the following topics:

- [Using UTL\\_RECOMP](#)
  - Overview
  - Operational Notes
  - Examples
- [Summary of UTL\\_RECOMP Subprograms](#)

## Using UTL\_RECOMP

- [Overview](#)
- [Operational Notes](#)
- [Examples](#)

## Overview

This script is particularly useful after a major-version upgrade that typically invalidates all PL/SQL and Java objects. Although invalid objects are recompiled automatically on use, it is useful to run this script prior to operation because this will either eliminate or minimize subsequent latencies due to on-demand automatic recompilation at runtime.

Parallel recompilation can exploit multiple CPUs to reduce the time taken to recompile invalid objects. The degree of parallelism is specified by the first argument to [RECOMP\\_PARALLEL Procedure](#).

In general, a parallelism setting of one thread for each available CPU provides a good initial setting. However, please note that the process of recompiling an invalid object writes a significant amount of data to system tables and is fairly I/O intensive. A slow disk system may be a significant bottleneck and limit speedups available from a higher degree of parallelism.

## Operational Notes

- This package uses the job queue for parallel recompilation.
- This package must be run using SQL\*PLUS.
- You must be connected AS SYSDBA to run this script.
- This package expects the following packages to have been created with VALID status:
  - STANDARD (standard.sql)
  - DBMS\_STANDARD (dbmsstdx.sql)
  - DBMS\_JOB (dbmsjob.sql)
  - DBMS\_RANDOM (dbmsrand.sql)
- There should be no other DDL on the database while running entries in this package. Not following this recommendation may lead to deadlocks.

## Examples

Recompile all objects sequentially:

```
EXECUTE UTL_RECOMP.RECOMP_SERIAL();
```

Recompile objects in schema SCOTT sequentially:

```
EXECUTE UTL_RECOMP.RECOMP_SERIAL('SCOTT');
```

Recompile all objects using 4 parallel threads:

```
EXECUTE UTL_RECOMP.RECOMP_PARALLEL(4);
```

Recompile objects in schema JOE using the number of threads specified in the parameter JOB\_QUEUE\_PROCESSES:

```
EXECUTE UTL_RECOMP.RECOMP_PARALLEL(NULL, 'JOE');
```

## Summary of UTL\_RECOMP Subprograms

**Table 233–1 UTL\_RECOMP Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">RECOMP_PARALLEL Procedure</a> on page 233-7	Recompiles invalid objects in a given schema, or all invalid objects in the database, in parallel
<a href="#">RECOMP_SERIAL Procedure</a> on page 233-8	Recompiles invalid objects in a given schema or all invalid objects in the database

## RECOMP\_PARALLEL Procedure

This procedure uses the information exposed in the `DBA_Dependencies` view to recompile invalid objects in the database, or in a given schema, in parallel.

### Syntax

```
UTL_RECOMP.RECOMP_PARALLEL(  
  threads IN PLS_INTEGER DEFAULT NULL,  
  schema  IN VARCHAR2     DEFAULT NULL,  
  flags   IN PLS_INTEGER DEFAULT 0);
```

### Parameters

**Table 233–2 RECOMP\_PARALLEL Procedure Parameters**

Parameter	Description
threads	The number of recompile threads to run in parallel. If <code>NULL</code> , use the value of <code>'job_queue_processes'</code> .
schema	The schema in which to recompile invalid objects. If <code>NULL</code> , all invalid objects in the database are recompiled.
flags	Flag values are intended for internal testing and diagnosability only.

### Usage Notes

The parallel recompile exploits multiple CPUs to reduce the time taken to recompile invalid objects. However, please note that recompilation writes significant amounts of data to system tables, so the disk system may be a bottleneck and prevent significant speedups.

## RECOMP\_SERIAL Procedure

This procedure recompiles invalid objects in a given schema or all invalid objects in the database.

### Syntax

```
UTL_RECOMP.RECOMP_SERIAL(  
  schema IN VARCHAR2 DEFAULT NULL,  
  flags  IN PLS_INTEGER DEFAULT 0);
```

### Parameters

**Table 233–3** *RECOMP\_SERIAL Procedure Parameters*

Parameter	Description
schema	The schema in which to recompile invalid objects. If NULL, all invalid objects in the database are recompiled.
flags	Flag values are intended for internal testing and diagnosability only.



The UTL\_REF package provides PL/SQL procedures to support reference-based operations. Unlike SQL, UTL\_REF procedures enable you to write generic type methods without knowing the object table name.

This chapter contains the following topics:

- [Using UTL\\_REF](#)
  - Overview
  - Security Model
  - Types
  - Exceptions
- [Summary of UTL\\_REF Subprograms](#)

## Using UTL\_REF

- [Overview](#)
- [Security Model](#)
- [Types](#)
- [Exceptions](#)

## Overview

Oracle supports user-defined composite type or object type. Any instance of an object type is called an object. An object type can be used as the type of a column or as the type of a table.

In an object table, each row of the table stores an object. You can uniquely identify an object in an object table with an object identifier.

A reference is a persistent pointer to an object, and each reference can contain an object identifier. The reference can be an attribute of an object type, or it can be stored in a column of a table. Given a reference, an object can be retrieved.

## Security Model

The procedural option is needed to use this package. This package must be created under `SYS` (connect/as sysdba). Operations provided by this package are performed under the current calling user, not under the package owner `SYS`.

You can use the `UTL_REF` package from stored PL/SQL procedures/packages on the server, as well as from client-side PL/SQL code.

When invoked from PL/SQL procedures/packages on the server, `UTL_REF` verifies that the invoker has the appropriate privileges to access the object pointed to by the `REF`.

---

---

**Note:** This is in contrast to PL/SQL packages/procedures on the server which operate with definer's privileges, where the package owner must have the appropriate privileges to perform the desired operations.

---

---

Thus, if `UTL_REF` is defined under user `SYS`, and user `A` invokes `UTL_REF.SELECT` to select an object from a reference, then user `A` (the invoker) requires the privileges to check.

When invoked from client-side PL/SQL code, `UTL_REF` operates with the privileges of the client session under which the PL/SQL execution is being done.

## Types

An object type is a composite datatype defined by the user or supplied as a library type. You can create the object type `employee_type` using the following syntax:

```
CREATE TYPE employee_type AS OBJECT (  
    name    VARCHAR2(20),  
    id      NUMBER,  
  
    member function GET_ID  
        (name VARCHAR2)  
        RETURN MEMBER);
```

The object type `employee_type` is a user-defined type that contains two attributes, `name` and `id`, and a member function, `GET_ID()`.

You can create an object table using the following SQL syntax:

```
CREATE TABLE employee_table OF employee_type;
```

## Exceptions

Exceptions can be returned during execution of `UTL_REF` functions for various reasons. For example, the following scenarios would result in exceptions:

- The object selected does not exist. This could be because either:
  1. The object has been deleted, or the given reference is dangling (invalid).
  2. The object table was dropped or does not exist.
- The object cannot be modified or locked in a serializable transaction. The object was modified by another transaction after the serializable transaction started.
- You do not have the privilege to select or modify the object. The caller of the `UTL_REF` subprogram must have the proper privilege on the object that is being selected or modified.

**Table 234–1** *UTL\_REF Exceptions*

Exceptions	Description
<code>errnum == 942</code>	Insufficient privileges.
<code>errnum == 1031</code>	Insufficient privileges.
<code>errnum == 8177</code>	Unable to serialize, if in a serializable transaction.
<code>errnum == 60</code>	Deadlock detected.
<code>errnum == 1403</code>	No data found (if the REF is NULL, and so on.).

The `UTL_REF` package does not define any named exceptions. You may define exception handling blocks to catch specific exceptions and to handle them appropriately.

## Summary of UTL\_REF Subprograms

**Table 234–2 UTL\_REF Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">DELETE_OBJECT Procedure</a> on page 234-8	Deletes an object given a reference
<a href="#">LOCK_OBJECT Procedure</a> on page 234-10	Locks an object given a reference
<a href="#">SELECT_OBJECT Procedure</a> on page 234-11	Selects an object given a reference
<a href="#">UPDATE_OBJECT Procedure</a> on page 234-12	Updates an object given a reference

## DELETE\_OBJECT Procedure

This procedure deletes an object given a reference. The semantic of this subprogram is similar to the following SQL statement:

```
DELETE FROM object_table
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

### Syntax

```
UTL_REF.DELETE_OBJECT (
    reference IN REF "<typename>");
```

### Parameters

**Table 234–3 DELETE\_OBJECT Procedure Parameters**

Parameter	Description
reference	Reference of the object to delete.

### Exceptions

May be raised.

### Examples

The following example illustrates usage of the UTL\_REF package to implement this scenario: if an employee of a company changes their address, their manager should be notified.

... declarations of Address\_t and others...

```
CREATE OR REPLACE TYPE Person_t (
    name    VARCHAR2(64),
    gender  CHAR(1),
    address Address_t,
    MEMBER PROCEDURE setAddress(addr IN Address_t)
);

CREATE OR REPLACE TYPE BODY Person_t (
    MEMBER PROCEDURE setAddress(addr IN Address_t) IS
    BEGIN
        address := addr;
    END;
);

CREATE OR REPLACE TYPE Employee_t (
```

Under Person\_t: Simulate implementation of inheritance using a REF to Person\_t and delegation of setAddress to it.

```
    thePerson REF Person_t,
    empno     NUMBER(5),
    deptREF   Department_t,
    mgrREF    Employee_t,
    reminders StringArray_t,
```



```

MEMBER PROCEDURE setAddress(addr IN Address_t),
MEMBER procedure addReminder(reminder VARCHAR2);
);

```

```

CREATE TYPE BODY Employee_t (
MEMBER PROCEDURE setAddress(addr IN Address_t) IS
    myMgr Employee_t;
    meAsPerson Person_t;
BEGIN

```

Update the address by delegating the responsibility to thePerson. Lock the Person object from the reference, and also select it:

```

    UTL_REF.LOCK_OBJECT(thePerson, meAsPerson);
    meAsPerson.setAddress(addr);

```

Delegate to thePerson:

```

    UTL_REF.UPDATE_OBJECT(thePerson, meAsPerson);
    if mgr is NOT NULL THEN

```

Give the manager a reminder:

```

        UTL_REF.LOCK_OBJECT(mgr);
        UTL_REF.SELECT_OBJECT(mgr, myMgr);
        myMgr.addReminder
        ('Update address in the employee directory for' ||
        thePerson.name || ', new address: ' || addr.asString);
        UTL_REF.UPDATE_OBJECT(mgr, myMgr);
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        errnum := SQLCODE;
        errmsg := SUBSTR(SQLERRM, 1, 200);

```

## LOCK\_OBJECT Procedure

This procedure locks an object given a reference. In addition, this procedure lets the program select the locked object. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
  INTO object
  FROM object_table t
  WHERE REF(t) = reference
  FOR UPDATE;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides. It is not necessary to lock an object before updating/deleting it.

### Syntax

```
UTL_REF.LOCK_OBJECT (
  reference IN REF "<typename>");

UTL_REF.LOCK_OBJECT (
  reference IN REF "<typename>",
  object    IN OUT "<typename>");
```

### Parameters

**Table 234–4 LOCK\_OBJECT Procedure Parameters**

Parameter	Description
reference	Reference of the object to lock.
object	The PL/SQL variable that stores the locked object. This variable should be of the same object type as the locked object.

### Exceptions

May be raised.

## SELECT\_OBJECT Procedure

This procedure selects an object given its reference. The selected object is retrieved from the database and its value is put into the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
INTO object
FROM object_table t
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

### Syntax

```
UTL_REF.SELECT_OBJECT (
    reference IN REF "<typename>",
    object    IN OUT "<typename>");
```

### Parameters

**Table 234–5** *SELECT\_OBJECT Procedure Parameters*

Parameter	Description
reference	Reference to the object to select or retrieve.
object	The PL/SQL variable that stores the selected object; this variable should be of the same object type as the referenced object.

### Exceptions

May be raised.

## UPDATE\_OBJECT Procedure

This procedure updates an object given a reference. The referenced object is updated with the value contained in the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
UPDATE object_table t
SET VALUE(t) = object
WHERE REF(t) = reference;
```

Unlike the preceding SQL statement, this subprogram does not require you to specify the object table name where the object resides.

### Syntax

```
UTL_REF.UPDATE_OBJECT (
    reference IN REF "<typename>",
    object    IN    "<typename>");
```

### Parameters

**Table 234–6 UPDATE\_OBJECT Procedure Parameters**

Parameter	Description
reference	Reference of the object to update.
object	The PL/SQL variable that contains the new value of the object. This variable should be of the same object type as the object to update.

### Exceptions

May be raised.

The UTL\_SMTP package is designed for sending electronic mails (e-mails) over Simple Mail Transfer Protocol (SMTP) as specified by RFC821.

**See Also:** *Oracle Database Advanced Application Developer's Guide*

This chapter contains the following topics:

- [Using UTL\\_SMTP](#)
  - Overview
  - Security Model
  - Constants
  - Types
  - Reply Codes
  - Exceptions
  - Rules and Limits
  - Examples
- [Summary of UTL\\_SMTP Subprograms](#)

## Using UTL\_SMTP

- [Overview](#)
- [Security Model](#)
- [Constants](#)
- [Types](#)
- [Reply Codes](#)
- [Exceptions](#)
- [Rules and Limits](#)
- [Examples](#)

## Overview

The UTL\_SMTP protocol consists of a set of commands for an e-mail client to dispatch e-mails to an SMTP server. The UTL\_SMTP package provides interfaces to the SMTP commands. For many of the commands, the package provides both a procedural and a functional interface. The functional form returns the reply from the server for processing by the client. The procedural form checks the reply and raises an exception if the reply indicates a transient (400-range reply code) or permanent error (500-range reply code). Otherwise, it discards the reply.

Note that the original SMTP protocol communicates using 7-bit ASCII. Using UTL\_SMTP, all text data (in other words, those in VARCHAR2) are converted to US7ASCII before it is sent to the server. Some implementations of SMTP servers that support SMTP extension 8BITMIME [RFC1652] support full 8-bit communication between client and server. The body of the DATA command can be transferred in full 8 bits, but the rest of the SMTP command and response must be in 7 bits. When the target SMTP server supports 8BITMIME extension, users of multibyte databases may convert their non-US7ASCII, multibyte VARCHAR2 data to RAW and use the WRITE\_RAW\_DATA subprogram to send multibyte data using 8-bit MIME encoding.

UTL\_SMTP provides for SMTP communication as specified in RFC821, but does not provide an API to format the content of the message according to RFC 822 (for example, setting the subject of an electronic mail). You must format the message appropriately. In addition, UTL\_SMTP does not have the functionality to implement an SMTP server for an e-mail clients to send e-mails using SMTP.

## Security Model

This package is an invoker's rights package. The invoking user must have the `connect` privilege granted in the access control list assigned to the remote network host to which the user must connect.

---

---

**Note:** For more information, see *Managing Fine-Grained Access in PL/SQL Packages and Types* in *Oracle Database Security Guide*

---

---



## Constants

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

The UTL\_SMTP package uses the constants shown in [Table 235-1, "UTL\\_SMTP Constants"](#).

**Table 235-1 UTL\_SMTP Constants**

Name	Type	Value	Description
ALL_SCHEMES	VARCHAR2 (256)	'CRAM-MD5 PLAIN LOGIN'	List of all authentication schemes UTL_SMTP supports, in order of their relative security strength. The subset of the schemes in ALL_SCHEMES (namely, PLAIN and LOGIN) in which cleartext passwords are sent over SMTP must be used only in SMTP connections that are secured by Secure Socket Layer / Transport Layer Security (SSL/TLS).
NON_CLEARTEXT_PASSWORD_SCHEMES	VARCHAR2 (256)	'CRAM-MD5'	List of authentication schemes that UTL_SMTP supports and in which no cleartext passwords are sent over SMTP. They can be used in SMTP connections that are not secured by SSL/TLS. Note that these schemes may still be weak when used in an insecure SMTP connection.

## Types

- [CONNECTION Record Type](#)
- [REPLY, REPLIES Record Types](#)

### CONNECTION Record Type

This is a PL/SQL record type used to represent an SMTP connection.

#### Syntax

```
TYPE connection IS RECORD (
    host          VARCHAR2(255),
    port          PLS_INTEGER,
    tx_timeout    PLS_INTEGER,
    private_tcp_con utl_tcp.connection,
    private_state PLS_INTEGER);
```

#### Fields

**Table 235–2 CONNECTION Record Type Fields**

Field	Description
host	Name of the remote host when connection is established. NULL when no connection is established.
port	Port number of the remote SMTP server connected. NULL when no connection is established.
tx_timeout	Time in seconds that the UTL_SMTP package waits before timing out in a read or write operation in this connection. In read operations, this package times out if no data is available for reading immediately. In write operations, this package times out if the output buffer is full and no data is to be sent into the network without being blocked. 0 indicates not to wait at all. NULL indicates to wait forever.
private_tcp_con	Private, for implementation use only. You should not modify this field.
private_state	Private, for implementation use only. You should not modify this field.

#### Usage Notes

The read-only fields in a connection record are used to return information about the SMTP connection after the connection is successfully made with the [OPEN\\_CONNECTION Functions](#). Changing the values of these fields has no effect on the connection. The fields `private_tcp_con` and `private_state` for implementation use only. You should not modify these fields.

### REPLY, REPLIES Record Types

These are PL/SQL record types used to represent an SMTP reply line. Each SMTP reply line consists of a reply code followed by a text message. While a single reply line is expected for most SMTP commands, some SMTP commands expect multiple reply lines. For those situations, a PL/SQL table of reply records is used to represent multiple reply lines.

## Syntax

```
TYPE reply IS RECORD (  
    code    PLS_INTEGER,  
    text    VARCHAR2(508));
```

```
TYPE replies IS TABLE OF reply INDEX BY BINARY_INTEGER;
```

## Fields

**Table 235-3** *REPLY, REPLIES Record Type Fields*

Field	Description
code	3-digit reply code
text	Text message of the reply

## Reply Codes

The following is a list of the SMTP reply codes.

**Table 235–4 SMTP Reply Codes**

Reply Code	Meaning
211	System status, or system help reply
214	Help message [Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220	<domain> Service ready
221	<domain> Service closing transmission channel
250	Requested mail action okay, completed
251	User not local; forwards to <forward-path>
252	OK, pending messages for node <node> started. Cannot VRFY user (for example, info is not local), but takes message for this user and attempts delivery.
253	OK, <messages> pending messages for node <node> started
354	Start mail input; end with <CRLF . CRLF>
355	Octet-offset is the transaction offset
421	<domain> Service not available, closing transmission channel (This can be a reply to any command if the service knows it must shut down.)
450	Requested mail action not taken: mailbox unavailable [for example, mailbox busy]
451	Requested action terminated: local error in processing
452	Requested action not taken: insufficient system storage
453	You have no mail.
454	TLS not available due to temporary reason. Encryption required for requested authentication mechanism.
458	Unable to queue messages for node <node>
459	Node <node> not allowed: reason
500	Syntax error, command unrecognized (This may include errors such as command line too long.)
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<Machine> does not accept mail.
530	Must issue a STARTTLS command first. Encryption required for requested authentication mechanism.
534	Authentication mechanism is too weak.
538	Encryption required for requested authentication mechanism.

**Table 235-4 (Cont.) SMTP Reply Codes**

<b>Reply Code</b>	<b>Meaning</b>
550	Requested action not taken: mailbox unavailable [for, mailbox not found, no access]
551	User not local; please try <forward-path>
552	Requested mail action terminated: exceeded storage allocation
553	Requested action not taken: mailbox name not allowed [for example, mailbox syntax incorrect]
554	Transaction failed

## Exceptions

The table lists the exceptions that can be raised by the interface of the UTL\_SMTP package. The network error is transferred to a reply code of 421- service not available.

**Table 235–5 UTL\_SMTP Exceptions**

---

INVALID_OPERATION	Raised when an invalid operation is made. In other words, calling API other than the <a href="#">WRITE_DATA Procedure</a> , the <a href="#">WRITE_RAW_DATA Procedure</a> or the <a href="#">CLOSE_DATA Function and Procedure</a> after the <a href="#">OPEN_DATA Function and Procedure</a> is called, or calling <code>WRITE_DATA</code> , <code>WRITE_RAW_DATA</code> or <code>CLOSE_DATA</code> without first calling <code>OPEN_DATA</code> .
TRANSIENT_ERROR	Raised when receiving a reply code in 400 range
PERMANENT_ERROR	Raised when receiving a reply code in 500 range

---

## Rules and Limits

No limitation or range-checking is imposed by the API. However, you must be aware of the following size limitations on various elements of SMTP. Sending data that exceed these limits may result in errors returned by the server.

**Table 235–6 SMTP Size Limitation**

<b>Element</b>	<b>Size Limitation</b>
user	Maximum total length of a user name is 64 characters
domain	Maximum total length of a domain name or number is 64 characters
path	Maximum total length of a reverse-path or forward-path is 256 characters (including the punctuation and element separators)
command line	Maximum total length of a command line including the command word and the <CRLF> is 512 characters
reply line	Maximum total length of a reply line including the reply code and the <CRLF> is 512 characters
text line	Maximum total length of a text line including the <CRLF> is 1000 characters (but not counting the leading dot duplicated for transparency)
recipients buffer	Maximum total number of recipients that must be buffered is 100 recipients

## Examples

The following example illustrates how UTL\_SMTP is used by an application to send e-mail. The application connects to an SMTP server at port 25 and sends a simple text message.

```
DECLARE
  c UTL_SMTP.CONNECTION;

  PROCEDURE send_header(name IN VARCHAR2, header IN VARCHAR2) AS
  BEGIN
    UTL_SMTP.WRITE_DATA(c, name || ': ' || header || UTL_TCP.CRLF);
  END;

BEGIN
  c := UTL_SMTP.OPEN_CONNECTION('smtp-server.acme.com');
  UTL_SMTP.HELO(c, 'foo.com');
  UTL_SMTP.MAIL(c, 'sender@foo.com');
  UTL_SMTP.RCPT(c, 'recipient@foo.com');
  UTL_SMTP.OPEN_DATA(c);
  send_header('From',      '"Sender" <sender@foo.com>');
  send_header('To',       '"Recipient" <recipient@foo.com>');
  send_header('Subject',  'Hello');
  UTL_SMTP.WRITE_DATA(c, UTL_TCP.CRLF || 'Hello, world!');
  UTL_SMTP.CLOSE_DATA(c);
  UTL_SMTP.QUIT(c);
EXCEPTION
  WHEN utl_smtp.transient_error OR utl_smtp.permanent_error THEN
  BEGIN
    UTL_SMTP.QUIT(c);
  EXCEPTION
    WHEN UTL_SMTP.TRANSIENT_ERROR OR UTL_SMTP.PERMANENT_ERROR THEN
      NULL; -- When the SMTP server is down or unavailable, we don't have
            -- a connection to the server. The QUIT call raises an
            -- exception that we can ignore.
  END;
  raise_application_error(-20000,
    'Failed to send mail due to the following error: ' || sqlerrm);
END;
```



---

## Summary of UTL\_SMTP Subprograms

**Table 235–7 UTL\_SMTP Package Subprograms**

Subprogram	Description
<a href="#">AUTH Function and Procedure</a> on page 235-14	Sends the AUTH command to authenticate to the SMTP server
<a href="#">CLOSE_CONNECTION Procedure</a> on page 235-16	Closes the SMTP connection, causing the current SMTP operation to terminate
<a href="#">CLOSE_DATA Function and Procedure</a> on page 235-17	Closes the data session
<a href="#">COMMAND Function and Procedure</a> on page 235-18	Performs a generic SMTP command
<a href="#">COMMAND_REPLIES Function</a> on page 235-19	Performs a generic SMTP command and retrieves multiple reply lines
<a href="#">DATA Function and Procedure</a> on page 235-20	Sends the e-mail body
<a href="#">EHLO Function and Procedure</a> on page 235-21	Performs the initial handshake with SMTP server using the EHLO command
<a href="#">HELO Function and Procedure</a> on page 235-22	Performs the initial handshake with SMTP server using the HELO command
<a href="#">HELP Function</a> on page 235-23	Sends HELP command
<a href="#">MAIL Function and Procedure</a> on page 235-24	Initiates an e-mail transaction with the server, the destination is a mailbox
<a href="#">NOOP Function and Procedure</a> on page 235-25	NULL command
<a href="#">OPEN_CONNECTION Functions</a> on page 235-26	Opens a connection to an SMTP server
<a href="#">OPEN_DATA Function and Procedure</a> on page 235-28	Sends the DATA command
<a href="#">QUIT Function and Procedure</a> on page 235-29	Terminates an SMTP session and disconnects from the server
<a href="#">RCPT Function</a> on page 235-30	Specifies the recipient of an e-mail message
<a href="#">RSET Function and Procedure</a> on page 235-31	Terminates the current e-mail transaction
<a href="#">STARTTLS Function and Procedure</a> on page 235-32	Sends STARTTLS command to secure the SMTP connection using SSL/TLS
<a href="#">VRFY Function</a> on page 235-33	Verifies the validity of a destination e-mail address
<a href="#">WRITE_DATA Procedure</a> on page 235-34	Writes a portion of the e-mail message
<a href="#">WRITE_RAW_DATA Procedure</a> on page 235-35	Writes a portion of the e-mail message with RAW data

## AUTH Function and Procedure

This subprogram sends the AUTH command to authenticate to the SMTP server. The UTL\_SMTP package goes through the user's choices of authentication schemes, skips any that is not supported by the SMTP server and uses the first supported. To determine the schemes the SMTP server supports from its EHLO reply, the user must call the [EHLO Function and Procedure](#). Otherwise, UTL\_SMTP uses the first scheme in the list.

---



---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```

UTL_SMTP.AUTH (
  c          IN OUT NOCOPY connection,
  username   IN          VARCHAR2,
  password   IN          VARCHAR2,
  schemes    IN          VARCHAR2 DEFAULT NON_CLEARTEXT_PASSWORD_SCHEMES)
RETURN reply;

UTL_SMTP.AUTH (
  c          IN OUT NOCOPY connection,
  username   IN          VARCHAR2,
  password   IN          VARCHAR2,
  schemes    IN          VARCHAR2 DEFAULT NON_CLEARTEXT_PASSWORD_SCHEMES);

```

### Parameters

**Table 235–8 AUTH Function and Procedure Parameters**

Parameter	Description
c	SMTP connection
username	Username
password	Password
schemes	Space-separated list of authentication schemes UTL_SMTP is allowed to use in the preferred order. See the ALL_SCHEMES and NON_CLEARTEXT_PASSWORD_SCHEMES constants for suggestions.

### Return Values

**Table 235–9 AUTH Function and Procedure Function Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY</a> , <a href="#">REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

- Currently only PLAIN, LOGIN and CRAM-MD5 authentication schemes are supported by UTL\_SMTP.

- Since the SMTP server may change the authentication schemes it supports after the SMTP connection is secured by SSL/TLS after the `STARTTLS` command (for example, adding `PLAIN` and `LOGIN`), the caller must call the [EHLO Function and Procedure](#) again for UTL\_SMTP to update the list after the [STARTTLS Function and Procedure](#) is called.

## Examples

```
DECLARE
  c utl_smtp.connection;
BEGIN
  c := utl_smtp.open_connection(
    host => 'smtp.example.com',
    port => 25,
    wallet_path => 'file:/oracle/wallets/smtp_wallet',
    wallet_password => 'password',
    secure_connection_before_smtp => FALSE);
  UTL_SMTP.STARTTLS(c);
  UTL_SMTP.AUTH(
    c => c,
    username => 'scott',
    password => 'password'
    schemes => utl_smtp.all_schemes);
END;
```

## CLOSE\_CONNECTION Procedure

This procedure closes the SMTP connection, causing the current SMTP operation to terminate. Use this procedure only to cancel an e-mail in the middle of the data session. To end the SMTP connection properly, use the [QUIT Function and Procedure](#).

### Syntax

```
UTL_SMTP.CLOSE_CONNECTION (  
    c      IN OUT NOCOPY connection);
```

### Parameters

**Table 235–10** *CLOSE\_CONNECTION Procedure Parameters*

Parameter	Description
c	SMTP connection

## CLOSE\_DATA Function and Procedure

This subprogram ends the e-mail message by sending the sequence <CR><LF> . <CR><LF> (a single period at the beginning of a line).

### Syntax

```
UTL_SMTP.CLOSE_DATA (
    c      IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.CLOSE_DATA (
    c      IN OUT NOCOPY connection);
```

### Parameters

**Table 235-11** *CLOSE\_DATA Function and Procedure Parameters*

Parameter	Description
c	SMTP connection

### Return Values

**Table 235-12** *CLOSE\_DATA Function and Procedure Return Values*

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

The calls to `OPEN_DATA`, `WRITE_DATA`, `WRITE_RAW_DATA` and `CLOSE_DATA` must be made in the right order. A program calls `OPEN_DATA` to send the `DATA` command to the SMTP server. After that, it can call `WRITE_DATA` or `WRITE_RAW_DATA` repeatedly to send the actual data. The data is terminated by calling `CLOSE_DATA`. After `OPEN_DATA` is called, the only subprograms that can be called are `WRITE_DATA`, `WRITE_RAW_DATA`, or `CLOSE_DATA`. A call to other subprograms results in an `INVALID_OPERATION` exception being raised.

`CLOSE_DATA` must be called only after `OPEN_CONNECTION`, `HELO` or `EHLO`, `MAIL`, and `RCPT` have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.

Note that there is no function form of `WRITE_DATA` because the SMTP server does not respond until the data-terminator is sent during the call to `CLOSE_DATA`.

## COMMAND Function and Procedure

This subprogram performs a generic SMTP command.

### Syntax

```
UTL_SMTP.COMMAND (
  c      IN OUT NOCOPY  connection,
  cmd    IN              VARCHAR2,
  arg    IN              VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.COMMAND (
  c      IN OUT NOCOPY  connection,
  cmd    IN              VARCHAR2,
  arg    IN              VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 235–13** *COMMAND Function and Procedure Parameters*

Parameter	Description
c	SMTP connection
cmd	SMTP command to send to the server
arg	Optional argument to the SMTP argument. A space is inserted between cmd and arg.

### Return Values

**Table 235–14** *COMMAND Function and Procedure Return Values*

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

This function is used to invoke generic SMTP commands. Use `COMMAND` if only a single reply line is expected. Use `COMMAND_REPLIES` if multiple reply lines are expected.

For `COMMAND`, if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

## COMMAND\_REPLIES Function

This function performs a generic SMTP command and retrieves multiple reply lines.

### Syntax

```
UTL_SMTP.COMMAND_REPLIES (
  c      IN OUT NOCOPY  connection,
  cmd   IN              VARCHAR2,
  arg   IN              VARCHAR2 DEFAULT NULL)
RETURN replies;
```

### Parameters

**Table 235–15** *COMMAND\_REPLIES Function Parameters*

Parameter	Description
c	SMTP connection
cmd	SMTP command to send to the server
arg	Optional argument to the SMTP argument. A space is inserted between cmd and arg.

### Return Values

**Table 235–16** *COMMAND\_REPLIES Function Return Values*

Return Value	Description
replies	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> )

### Usage Notes

This function is used to invoke generic SMTP commands. Use `COMMAND` if only a single reply line is expected. Use `COMMAND_REPLIES` if multiple reply lines are expected.

For `COMMAND`, if multiple reply lines are returned from the SMTP server, it returns the last reply line only.

## DATA Function and Procedure

This subprogram specifies the body of an e-mail message.

### Syntax

```
UTL_SMTP.DATA (
  c      IN OUT NOCOPY connection
  body   IN  VARCHAR2 CHARACTER SET ANY_CS)
RETURN reply;

UTL_SMTP.DATA (
  c      IN OUT NOCOPY connection
  body   IN  VARCHAR2 CHARACTER SET ANY_CS);
```

### Parameters

**Table 235–17 DATA Function and Procedure Parameters**

Parameter	Description
c	SMTP Connection
body	Text of the message to be sent, including headers, in [RFC822] format

### Return Values

**Table 235–18 DATA Function and Procedure Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY</a> , <a href="#">REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The DATA routine terminates the message with a <CR><LF> . <CR><LF> sequence (a single period at the beginning of a line), as required by RFC821. It also translates any sequence of <CR><LF> . <CR><LF> (single period) in body to <CR><LF> . . <CR><LF> (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.

The DATA subprogram must be called only after OPEN\_CONNECTION, HELO or EHLO, MAIL and RCPT have been called. The connection to the SMTP server must be open, and a mail transaction must be active when this routine is called.

The expected response from the server is a message beginning with status code 250. The 354 response received from the initial DATA command is not returned to the caller.



## EHLO Function and Procedure

This subprogram performs the initial handshake with SMTP server using the EHLO command.

### Syntax

```
UTL_SMTP.EHLO (
  c          IN OUT NOCOPY connection,
  domain IN)
RETURN replies;
```

```
UTL_SMTP.EHLO (
  c          IN OUT NOCOPY connection,
  domain IN);
```

### Parameters

**Table 235–19 EHLO Function and Procedure Parameters**

Parameter	Description
c	SMTP connection
domain	Domain name of the local (sending) host. Used for identification purposes.

### Return Values

**Table 235–20 EHLO Function and Procedure Return Values**

Return Value	Description
replies	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ).

### Usage Notes

The EHLO interface is identical to HELO except that it allows the server to return more descriptive information about its configuration. [RFC1869] specifies the format of the information returned, which the PL/SQL application can retrieve using the functional form of this call. For compatibility with HELO, each line of text returned by the server begins with status code 250.

### Related Functions

[HELO Function and Procedure](#)

## HELO Function and Procedure

This subprogram performs the initial handshake with SMTP server using the HELO command.

### Syntax

```
UTL_SMTP.HELO (
  c          IN OUT NOCOPY  connection,
  domain    IN              VARCHAR2)
RETURN reply;

UTL_SMTP.HELO (
  c          IN OUT NOCOPY  connection,
  domain    IN              VARCHAR2);
```

### Parameters

**Table 235–21 HELO Function and Procedure Parameters**

Parameter	Description
c	SMTP connection
domain	Domain name of the local (sending) host. Used for identification purposes.

### Return Values

**Table 235–22 HELO Function and Procedure Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

RFC 821 specifies that the client must identify itself to the server after connecting. This routine performs that identification. The connection must have been opened through a call to [OPEN\\_CONNECTION Functions](#) before calling this routine.

The expected response from the server is a message beginning with status code 250.

### Related Functions

[EHLO Function and Procedure](#)

## HELP Function

This function sends the HELP command.

### Syntax

```
UTL_SMTP.HELP (
  c          IN OUT NOCOPY  connection,
  command   IN              VARCHAR2 DEFAULT NULL)
RETURN replies;
```

### Parameters

**Table 235–23 HELP Function Parameters**

Parameter	Description
c	SMTP connection
command	Command to get the help message

### Return Values

**Table 235–24 HELP Function Return Values**

Return Value	Description
replies	Reply of the command (see <a href="#">REPLY</a> , <a href="#">REPLIES</a> Record Types)

## MAIL Function and Procedure

This subprogram initiates a mail transaction with the server. The destination is a mailbox.

### Syntax

```
UTL_SMTP.MAIL (
  c          IN OUT NOCOPY  connection,
  sender     IN             VARCHAR2,
  parameters IN             VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.MAIL (
  c          IN OUT NOCOPY  connection,
  sender     IN             VARCHAR2,
  parameters IN             VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 235–25 MAIL Function and Procedure Parameters**

Parameter	Description
c	SMTP connection
sender	E-mail address of the user sending the message.
parameters	Additional parameters to mail command as defined in Section 6 of [RFC1869]. It must follow the format of "XXX=XXX (XXX=XXX ...)".

### Return Values

**Table 235–26 MAIL Function and Procedure Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY</a> , <a href="#">REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

This command does not send the message; it simply begins its preparation. It must be followed by calls to `RCPT` and `DATA` to complete the transaction. The connection to the SMTP server must be open and a `HELO` or `EHLO` command must have already been sent.

The expected response from the server is a message beginning with status code 250.

## NOOP Function and Procedure

This subprogram issues the NULL command.

### Syntax

```
UTL_SMTP.NOOP (
  c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.NOOP (
  c IN OUT NOCOPY connection);
```

### Parameter

**Table 235–27 NOOP Function and Procedure Parameters**

Parameter	Description
c	SMTP connection

### Return Values

**Table 235–28 NOOP Function and Procedure Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

- This command has no effect except to elicit a successful reply from the server. It can be issued at any time after the connection to the server has been established with `OPEN_CONNECTION`. The `NOOP` command can be used to verify that the server is still connected and is listening properly.
- This command replies with a single line beginning with status code 250.

## OPEN\_CONNECTION Functions

These functions open a connection to an SMTP server.

---

**Note:** This functionality associated with `wallet_path`, `wallet_password` and `secure_connection_before_smtp` is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

### Syntax

```

UTL_SMTP.OPEN_CONNECTION (
    host                IN  VARCHAR2,
    port                IN  PLS_INTEGER DEFAULT 25,
    c                   OUT connection,
    tx_timeout          IN  PLS_INTEGER DEFAULT NULL,
    wallet_path         IN  VARCHAR2 DEFAULT NULL,
    wallet_password     IN  VARCHAR2 DEFAULT NULL,
    secure_connection_before_smtp IN BOOLEAN DEFAULT FALSE)
RETURN reply;

```

```

UTL_SMTP.OPEN_CONNECTION (
    host                IN  VARCHAR2,
    port                IN  PLS_INTEGER DEFAULT 25,
    tx_timeout          IN  PLS_INTEGER DEFAULT NULL,
    wallet_path         IN  VARCHAR2 DEFAULT NULL,
    wallet_password     IN  VARCHAR2 DEFAULT NULL,
    secure_connection_before_smtp IN BOOLEAN DEFAULT FALSE)
RETURN reply;

```

### Parameters

**Table 235–29 OPEN\_CONNECTION Functions Parameters**

Parameter	Description
<code>host</code>	Name of the SMTP server host
<code>port</code>	Port number on which SMTP server is listening (usually 25)
<code>c</code>	SMTP connection
<code>tx_timeout</code>	Time in seconds that the UTL_SMTP package waits before timing out in a read or write operation for this connection. In read operations, this package times out if no data is available for reading immediately. In write operations, this package times out if the output buffer is full and no data is to be sent into the network without being blocked. 0 indicates not to wait at all. NULL indicates to wait forever.
<code>wallet_path</code>	Directory path that contains the Oracle wallet for SSL/TLS. The format is file: <directory-path>
<code>wallet_password</code>	Password to open the wallet. When the wallet is auto-login enabled, the password can be set to NULL.
<code>secure_connection_before_smtp</code>	If TRUE, a secure connection with SSL/TLS is made before SMTP communication. If FALSE, no connection is made.

## Return Values

**Table 235–30 OPEN\_CONNECTION Functions Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

## Usage Notes

- The expected response from the server is a message beginning with status code 220.
- The version of OPEN\_CONNECTION that returns UTL\_SMTP.CONNECTION record checks the reply code returned by an SMTP server when the connection is first established. It raises an exception when the reply indicates an error. Otherwise, it discards the reply. If you want to examine the reply, invoke the version of OPEN\_CONNECTION that returns REPLY.
- tx\_timeout is intended to govern both the read operations and the write operations. However, an implementation restriction prevents tx\_timeout from governing write operations in the current release.

## Examples

```

DECLARE
  c utl_smtp.connection;
BEGIN
  c := UTL_SMTP.OPEN_CONNECTION(
    host => 'smtp.example.com',
    port => 465,
    wallet_path => 'file:/oracle/wallets/smtp_wallet',
    wallet_password => 'password',
    secure_connection_before_smtp => TRUE);
END;
```

## OPEN\_DATA Function and Procedure

This subprogram sends the DATA command after which you can use WRITE\_DATA and WRITE\_RAW\_DATA to write a portion of the e-mail message.

### Syntax

```
UTL_SMTP.OPEN_DATA (
    c      IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.OPEN_DATA (
    c      IN OUT NOCOPY connection);
```

### Parameters

**Table 235–31 OPEN\_DATA Function and Procedure Parameters**

Parameter	Description
c	SMTP connection
data	Portion of the text of the message to be sent, including headers, in RFC822 format.

### Return Values

**Table 235–32 OPEN\_DATA Function and Procedure Function Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

- The calls to OPEN\_DATA, WRITE\_DATA, WRITE\_RAW\_DATA and CLOSE\_DATA must be made in the right order. A program calls OPEN\_DATA to send the DATA command to the SMTP server. After that, it can call WRITE\_DATA or WRITE\_RAW\_DATA repeatedly to send the actual data. The data is terminated by calling CLOSE\_DATA. After OPEN\_DATA is called, the only subprograms that can be called are WRITE\_DATA, WRITE\_RAW\_DATA, or CLOSE\_DATA. A call to other subprograms results in an INVALID\_OPERATION exception being raised.
- OPEN\_DATA must be called only after OPEN\_CONNECTION, HELO or EHLO, MAIL, and RCPT have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.



## QUIT Function and Procedure

This subprogram terminates an SMTP session and disconnects from the server.

### Syntax

```
UTL_SMTP.QUIT (
    c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.QUIT (
    c IN OUT NOCOPY connection);
```

### Parameter

**Table 235–33** *QUIT Function and Procedure Parameters*

Parameter	Description
c	SMTP connection

### Return Values

**Table 235–34** *QUIT Function and Procedure Function Return Values*

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

The `QUIT` command informs the SMTP server of the client's intent to terminate the session. It then closes the connection established by `OPEN_CONNECTION` which must have been called before executing this command. If a mail transaction is in progress when `QUIT` is issued, it is canceled in the same manner as `RSET`.

The function form of this command returns a single line beginning with the status code 221 on successful termination. In all cases, the connection to the SMTP server is closed. The fields `REMOTE_HOST` and `REMOTE_PORT` of `c` are reset.

### Related Functions

[RSET Function and Procedure](#)

## RCPT Function

This subprogram specifies the recipient of an e-mail message.

### Syntax

```

UTL_SMTP.RCPT (
  c          IN OUT NOCOPY    connection,
  recipient  IN                VARCHAR2,
  parameters IN                VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.RCPT (
  c          IN OUT NOCOPY    connection,
  recipient  IN                VARCHAR2,
  parameters IN                VARCHAR2 DEFAULT NULL);

```

**Table 235–35** *RCPT Function and Procedure Parameters*

Parameter	Description
c	SMTP connection
recipient	E-mail address of the user to which the message is being sent
parameters	Additional parameters to RCPT command as defined in Section 6 of [RFC1869]. It must follow the format of "XXX=XXX (XXX=XXX ...)".

### Return Values

**Table 235–36** *RCPT Function and Procedure Function Return Values*

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

To send a message to multiple recipients, call this routine multiple times. Each invocation schedules delivery to a single e-mail address. The message transaction must have been begun by a prior call to `MAIL`, and the connection to the mail server must have been opened and initialized by prior calls to `OPEN_CONNECTION` and `HELO` or `EHLO` respectively.

The expected response from the server is a message beginning with status code 250 or 251.

## RSET Function and Procedure

This subprogram terminates the current mail transaction.

### Syntax

```
UTL_SMTP.RSET (
  c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.RSET (
  c IN OUT NOCOPY connection);
```

### Parameters

**Table 235–37 RSET Function and Procedure Parameters**

Parameter	Description
c	SMTP connection

### Return Values

**Table 235–38 RSET Function and Procedure Return Values**

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

- This command allows the client to cancel an e-mail message it was in the process of composing. No mail is sent. The client can call RSET at any time after the connection to the SMTP server has been opened by means of OPEN\_CONNECTION until DATA or OPEN\_DATA is called. Once the e-mail data has been sent, it is too late to prevent the e-mail from being sent.
- The server responds to RSET with a message beginning with status code 250.

### Related Functions

[QUIT Function and Procedure](#)

## STARTTLS Function and Procedure

This subprogram sends the `STARTTLS` command to secure the SMTP connection using SSL/TLS. SSL/TLS requires an Oracle wallet which must be specified when the connection was opened by the [OPEN\\_CONNECTION Functions](#).

### Syntax

```
UTL_SMTP.STARTTLS (
  c IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.STARTTLS (
  c IN OUT NOCOPY connection);
```

### Parameters

**Table 235–39 STARTTLS Function and Procedure Parameters**

Parameter	Description
c	SMTP connection

### Return Values

**Table 235–40 STARTTLS Function and Procedure Return Values**

Return Value	Description
reply	SMTP reply

### Usage Notes

The `STARTTLS` command must only be issued on an unencrypted connection and when the SMTP server indicates the support of the command in the reply of the `EHLO` command. The wallet to be used for encryption must have been specified when the initial SMTP connection was opened by the `OPEN_CONNECTION` function.

### Examples

```
DECLARE
  c utl_smtp.connection;
BEGIN
  c := utl_smtp.open_connection(
    host => 'smtp.example.com',
    port => 25,
    wallet_path => 'file:/oracle/wallets/smtp_wallet',
    wallet_password => 'password',
    secure_connection_before_smtp => FALSE);
  utl_smtp.starttls(c);
END
```

## VERFY Function

This function verifies the validity of a destination e-mail address.

### Syntax

```
UTL_SMTP.VERFY (
  c          IN OUT NOCOPY connection
  recipient IN VARCHAR2)
RETURN reply;
```

### Parameters

**Table 235–41** *VERFY Function Parameters*

Parameter	Description
c	SMTP connection
recipient	E-mail address to be verified

### Return Values

**Table 235–42** *VERFY Function Return Values*

Return Value	Description
reply	Reply of the command (see <a href="#">REPLY, REPLIES Record Types</a> ). In cases where there are multiple replies, the last reply is returned.

### Usage Notes

The server attempts to resolve the destination address `recipient`. If successful, it returns the recipient's full name and fully qualified mailbox path. The connection to the server must have already been established by means of `OPEN_CONNECTION` and `HELO` or `EHLO` before making this request.

Successful verification returns one or more lines beginning with status code 250 or 251.

## WRITE\_DATA Procedure

This procedure writes a portion of the e-mail message. A repeat call to `WRITE_DATA` appends data to the e-mail message.

### Syntax

```
UTL_SMTP.WRITE_DATA (
  c      IN OUT NOCOPY connection,
  data   IN VARCHAR2 CHARACTER SET ANY_CS);
```

### Parameters

**Table 235–43** *WRITE\_DATA Procedure Parameters*

Parameter	Description
c	SMTP connection
data	Portion of the text of the message to be sent, including headers, in [RFC822] format

### Usage Notes

- The calls to the [OPEN\\_DATA Function and Procedure](#), [WRITE\\_DATA Procedure](#), [WRITE\\_RAW\\_DATA Procedure](#) and [CLOSE\\_DATA Function and Procedure](#) must be made in the correct order. A program calls `OPEN_DATA` to send the `DATA` command to the SMTP server. After that, it can call `WRITE_DATA` or `WRITE_RAW_DATA` repeatedly to send the actual data. The data is terminated by calling `CLOSE_DATA`. After `OPEN_DATA` is called, the only subprograms that can be called are `WRITE_DATA`, `WRITE_RAW_DATA`, or `CLOSE_DATA`. A call to other subprograms results in an `INVALID_OPERATION` exception being raised.
- The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The `DATA` routine terminates the message with a `<CR><LF> . <CR><LF>` sequence (a single period at the beginning of a line), as required by RFC821. It also translates any sequence of `<CR><LF> . <CR><LF>` (single period) in the body to `<CR><LF> . . <CR><LF>` (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.
- The [OPEN\\_DATA Function and Procedure](#), [WRITE\\_DATA Procedure](#), [WRITE\\_RAW\\_DATA Procedure](#) and [CLOSE\\_DATA Function and Procedure](#) must be called only after [OPEN\\_CONNECTION Functions](#), [HELO Function and Procedure](#), or [EHLO Function and Procedure](#), [MAIL Function and Procedure](#), and [RCPT Function](#) have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.
- Note that there is no function form of the [WRITE\\_DATA Procedure](#) because the SMTP server does not respond until the data-terminator is sent during the call to [CLOSE\\_DATA Function and Procedure](#).
- Text (`VARCHAR2`) data sent using `WRITE_DATA` is converted to US7ASCII before it is sent. If the text contains multibyte characters, each multibyte character in the text that cannot be converted to US7ASCII is replaced by a '?' character. If 8BITMIME extension is negotiated with the SMTP server using the `EHLO` subprogram, multibyte `VARCHAR2` data can be sent by first converting the text to RAW using the `UTL_RAW` package, and then sending the RAW data using `WRITE_RAW_DATA`.

## WRITE\_RAW\_DATA Procedure

This procedure writes a portion of the e-mail message. A repeat call to `WRITE_RAW_DATA` appends data to the e-mail message.

### Syntax

```
UTL_SMTP.WRITE_RAW_DATA (
  c      IN OUT NOCOPY connection
  data  IN RAW);
```

### Parameters

**Table 235–44** *WRITE\_RAW\_DATA Procedure Parameters*

Parameter	Description
c	SMTP connection
data	Portion of the text of the message to be sent, including headers, in [RFC822] format

### Usage Notes

- The calls to the [OPEN\\_DATA Function and Procedure](#), [WRITE\\_DATA Procedure](#), [WRITE\\_RAW\\_DATA Procedure](#) and [CLOSE\\_DATA Function and Procedure](#) must be made in the correct order. A program calls `OPEN_DATA` to send the `DATA` command to the SMTP server. After that, it can call `WRITE_DATA` or `WRITE_RAW_DATA` repeatedly to send the actual data. The data is terminated by calling `CLOSE_DATA`. After `OPEN_DATA` is called, the only subprograms that can be called are `WRITE_DATA`, `WRITE_RAW_DATA`, or `CLOSE_DATA`. A call to other subprograms results in an `INVALID_OPERATION` exception being raised.
- The application must ensure that the contents of the body parameter conform to the MIME(RFC822) specification. The `DATA` routine terminates the message with a `<CR><LF> . <CR><LF>` sequence (a single period at the beginning of a line), as required by RFC821. It also translates any sequence of `<CR><LF> . <CR><LF>` (single period) in the body to `<CR><LF> . . <CR><LF>` (double period). This conversion provides the transparency as described in Section 4.5.2 of RFC821.
- The [OPEN\\_DATA Function and Procedure](#), [WRITE\\_DATA Procedure](#), [WRITE\\_RAW\\_DATA Procedure](#) and [CLOSE\\_DATA Function and Procedure](#) must be called only after [OPEN\\_CONNECTION Functions](#), [HELO Function and Procedure](#), or [EHLO Function and Procedure](#), [MAIL Function and Procedure](#), and [RCPT Function](#) have been called. The connection to the SMTP server must be open and a mail transaction must be active when this routine is called.
- Note that there is no function form of the [WRITE\\_DATA Procedure](#) because the SMTP server does not respond until the data-terminator is sent during the call to [CLOSE\\_DATA Function and Procedure](#).





The UTL\_SPADV package, one of a set of Oracle Streams packages, provides subprograms to collect and analyze statistics for the Oracle Streams components in a distributed database environment. This package uses the Oracle Streams Performance Advisor to gather statistics.

This chapter contains the following topic:

- [Using UTL\\_SPADV](#)
  - Overview
  - Security Model
  - Operational Notes
- [Summary of UTL\\_SPADV Subprograms](#)

**See Also:** *Oracle Streams Concepts and Administration* for more information about this package and the Oracle Streams Performance Advisor

## Using UTL\_SPADV

- [Overview](#)
- [Security Model](#)
- [Operational Notes](#)

## Overview

This package enables you to collect and analyze statistics about the performance or Oracle Streams components. You can either collect statistics on demand or you can create a monitoring job that continually monitors Oracle Streams performance.

When this package is used on an Oracle Database 11g Release 2 (11.2) database, it can monitor Oracle Database 10g Release 2 (10.2) and later databases. It cannot monitor databases before release 10.2.

**See Also:** *Oracle Streams Concepts and Administration*

## Security Model

Security on this package can be controlled in either of the following ways:

- Granting EXECUTE on this package to selected users or roles.
- Granting EXECUTE\_CATALOG\_ROLE to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted EXECUTE privilege on the package directly. It cannot be granted through a role.

To ensure that the user who runs the subprograms in this package has the necessary privileges, configure an Oracle Streams administrator and connect as the Oracle Streams administrator when using this package.

**See Also:** *Oracle Streams Replication Administrator's Guide* for information about configuring an Oracle Streams administrator

## Operational Notes

To use this package, you must connect to an Oracle database as an Oracle Streams administrator and run the `utlspadv.sql` script in the `rdbms/admin` directory in `ORACLE_HOME`.

The `utlspadv.sql` script creates the following tables:

- [STREAMS\\$\\_PA\\_COMPONENT](#) Table
- [STREAMS\\$\\_PA\\_COMPONENT\\_LINK](#) Table
- [STREAMS\\$\\_PA\\_COMPONENT\\_PROP](#) Table
- [STREAMS\\$\\_PA\\_COMPONENT\\_STAT](#) Table
- [STREAMS\\$\\_PA\\_CONTROL](#) Table
- [STREAMS\\$\\_PA\\_DATABASE](#) Table
- [STREAMS\\$\\_PA\\_DATABASE\\_PROP](#) Table
- [STREAMS\\$\\_PA\\_MONITORING](#) Table
- [STREAMS\\$\\_PA\\_PATH\\_BOTTLENECK](#) Table
- [STREAMS\\$\\_PA\\_PATH\\_STAT](#) Table
- [STREAMS\\$\\_PA\\_SHOW\\_COMP\\_STAT](#) Table
- [STREAMS\\$\\_PA\\_SHOW\\_PATH\\_STAT](#) Table

The Oracle Streams Performance Advisor populates these tables when it is run.

**See Also:** *Oracle Streams Concepts and Administration*

### STREAMS\$\_PA\_COMPONENT Table

The `STREAMS$_PA_COMPONENT` table displays information about the Oracle Streams components at each database.

**Table 236–1** *STREAMS\$\_PA\_COMPONENT Table*

Column	Datatype	NULL	Description
<code>COMPONENT_ID</code>	NUMBER	NOT NULL	Identification number assigned to the component by the Oracle Streams Performance Advisor
<code>COMPONENT_NAME</code>	VARCHAR2 (194)		Name of the component
<code>COMPONENT_DB</code>	VARCHAR2 (128)		Name of the database that contains the component

**Table 236–1 (Cont.) STREAMS\$\_PA\_COMPONENT Table**

Column	Datatype	NULL	Description
COMPONENT_TYPE	VARCHAR2 (20)		Type of the component The following types are possible: <ul style="list-style-type: none"> <li>▪ CAPTURE for a capture process</li> <li>▪ PROPAGATION SENDER for a propagation sender</li> <li>▪ PROPAGATION RECEIVER for a propagation receiver</li> <li>▪ APPLY for an apply process</li> <li>▪ QUEUE for a queue</li> </ul>
COMPONENT_CHANGED_TIME	DATE		Time when the component was last changed

**STREAMS\$\_PA\_COMPONENT\_LINK Table**

The STREAMS\$\_PA\_COMPONENT\_LINK table displays information about how information flows between Oracle Streams components.

**Table 236–2 STREAMS\$\_PA\_COMPONENT\_LINK Table**

Column	Datatype	NULL	Description
PATH_ID	NUMBER	NOT NULL	Identification number assigned to the path by the Oracle Streams Performance Advisor
PATH_KEY	VARCHAR2 (4000)		Unique key assigned to the path by the Oracle Streams Performance Advisor
SOURCE_COMPONENT_ID	NUMBER	NOT NULL	Source component ID for the path The path starts with this component.
DESTINATION_COMPONENT_ID	NUMBER	NOT NULL	Destination component ID for the path The path ends with this component.
POSITION	NUMBER		Position of the component in the path

**STREAMS\$\_PA\_COMPONENT\_PROP Table**

The STREAMS\$\_PA\_COMPONENT\_PROP table displays information about capture processes and apply processes necessary for analysis by the Streams Performance Advisor.

**Table 236–3** *STREAMS\$\_PA\_COMPONENT\_PROP* Table

Column	Datatype	NULL	Description
COMPONENT_ID	NUMBER	NOT NULL	Identification number assigned to the component by the Oracle Streams Performance Advisor
PROP_NAME	VARCHAR2 (30)		<p>Property name</p> <p>For a capture process, the component properties include the following:</p> <ul style="list-style-type: none"> <li>▪ SOURCE_DATABASE - The source database for the changes captured by the capture process</li> <li>▪ PARALLELISM - The setting for the parallelism capture process parameter</li> <li>▪ OPTIMIZATION_MODE - Indicates whether the capture process uses combined capture and apply (greater than zero) or does not use combined capture and apply (0)</li> </ul> <p>For an apply process, the component properties include the following:</p> <ul style="list-style-type: none"> <li>▪ SOURCE_DATABASE - The source database for the messages applied by the apply process</li> <li>▪ PARALLELISM - The setting for the parallelism apply process parameter</li> <li>▪ APPLY_CAPTURED - Indicates whether the apply process applies captured messages (YES) persistent messages (NO)</li> <li>▪ MESSAGE_DELIVERY_MODE - Either buffered or persistent</li> </ul>
PROP_VALUE	VARCHAR2 (30)		Property value

**STREAMS\$\_PA\_COMPONENT\_STAT** Table

The *STREAMS\$\_PA\_COMPONENT\_STAT* table displays performance statistics and session statistics about each Oracle Streams component.

**Table 236–4** *STREAMS\$\_PA\_COMPONENT\_STAT* Table

Column	Datatype	NULL	Description
ADVISOR_RUN_ID	NUMBER		Identification number of the Oracle Streams Performance Advisor run

**Table 236–4 (Cont.) STREAMS\$\_PA\_COMPONENT\_STAT Table**

Column	Datatype	NULL	Description
ADVISOR_RUN_TIME	DATE		Time when the Oracle Streams Performance Advisor was run for the advisor run ID
COMPONENT_ID	NUMBER		Identification number assigned to the component by the Oracle Streams Performance Advisor
STATISTIC_TIME	DATE		Time when the statistic was recorded
STATISTIC_NAME	VARCHAR2 (64)		Name of the statistic
STATISTIC_VALUE	NUMBER		Value recorded for the statistic
STATISTIC_UNIT	VARCHAR2 (64)		Unit of measurement for the statistic
SUB_COMPONENT_TYPE	VARCHAR2 (64)		Type of the subcomponent Only capture processes and apply processes have subcomponents. The following capture process subcomponent types are possible: <ul style="list-style-type: none"> <li>■ LOGMINER READER for a builder server of a capture process</li> <li>■ LOGMINER PREPARER for a preparer server of a capture process</li> <li>■ LOGMINER BUILDER for a reader server of a capture process</li> <li>■ CAPTURE SESSION for a capture process session</li> </ul> The following apply process subcomponent types are possible: <ul style="list-style-type: none"> <li>■ PROPAGATION SENDER+RECEIVER for sending LCRs from a capture process directly to an apply process in a combined capture and apply configuration in which both the capture process and apply process run on a single database</li> <li>■ APPLY READER for a reader server of an apply process</li> <li>■ APPLY COORDINATOR for a coordinator process of an apply process</li> <li>■ APPLY SERVER for a reader server of an apply process</li> </ul>
SESSION_ID	NUMBER		Identification number of the session for the component. Query the V\$SESSION view for information about the session.
SESSION_SERIAL#	NUMBER		Session serial number of the session for the component. Query the V\$SESSION view for information about the session.



**STREAMS\$\_PA\_CONTROL Table**

The STREAMS\$\_PA\_CONTROL table displays the parameters set for the COLLECT\_STATS procedure in this package. The parameters control the monitoring behavior.

**Table 236–5 STREAMS\$\_PA\_CONTROL Table**

Column	Datatype	NULL	Description
ADVISOR_RUN_ID	NUMBER		Identification number of the Oracle Streams Performance Advisor run
ADVISOR_RUN_TIME	DATE		Time when the Oracle Streams Performance Advisor was last run
PARAM_NAME	VARCHAR2 (30)		The name of the parameter
PARAM_VALUE	VARCHAR2 (4000)		The value set for the parameter
PARAM_UNIT	VARCHAR2 (30)		The unit of the parameter

**STREAMS\$\_PA\_DATABASE Table**

The STREAMS\$\_PA\_DATABASE table displays information about each database that contains Oracle Streams components.

**Table 236–6 STREAMS\$\_PA\_DATABASE Table**

Column	Datatype	NULL	Description
GLOBAL_NAME	VARCHAR2 (128)	NOT NULL	Global name of the database analyzed by the Oracle Streams Performance Advisor
LAST_QUERIED	DATE		The time when the Performance Advisor successfully collected information from a database in its last run
ERROR_NUMBER	NUMBER		The error number of the error encountered when the database was last queried
ERROR_MESSAGE	VARCHAR2 (4000)		The error message of the error encountered when the database was last queried

**STREAMS\$\_PA\_DATABASE\_PROP Table**

The STREAMS\$\_PA\_DATABASE\_PROP table displays Oracle Streams database property information necessary for analysis by the Streams Performance Advisor.

**Table 236–7 STREAMS\$\_PA\_DATABASE\_PROP Table**

Column	Datatype	NULL	Description
GLOBAL_NAME	VARCHAR2 (128)	NOT NULL	Global name of the database analyzed by the Oracle Streams Performance Advisor

**Table 236–7 (Cont.) STREAMS\$\_PA\_DATABASE\_PROP Table**

Column	Datatype	NULL	Description
PROP_NAME	VARCHAR2 (30)		Property name  The database properties include the following: <ul style="list-style-type: none"> <li>▪ VERSION</li> <li>▪ COMPATIBILITY</li> <li>▪ MANAGEMENT_PACK_ACCESS</li> <li>▪ DB_UNIQUE_NAME</li> </ul>
PROP_VALUE	VARCHAR2 (30)		Property value

**STREAMS\$\_PA\_MONITORING Table**

The STREAMS\$\_PA\_MONITORING table displays information about each monitoring job running in a database.

**Table 236–8 STREAMS\$\_PA\_MONITORING Table**

Column	Datatype	NULL	Description
JOB_NAME	VARCHAR2 (30)	NOT NULL	Name of the monitoring job
CLIENT_NAME	VARCHAR2 (30)		Name of the client that submitted the job  <b>See Also:</b> " <a href="#">Full Monitoring Job Names</a> " on page 236-25
QUERY_USER_NAME	VARCHAR2 (30)		User granted privileges to view the monitoring results
SHOW_STATS_TABLE	VARCHAR2 (30)		Name of the table used by the SHOW_STATS procedure to display statistics
STARTED_TIME	TIMESTAMP		Time the monitoring job started
STOPPED_TIME	TIMESTAMP		Time the monitoring job last stopped
ALTERED_TIME	TIMESTAMP		Time the monitoring job was last altered
STATE	VARCHAR2 (30)		State of the monitoring job, either ENABLED or STOPPED

**STREAMS\$\_PA\_PATH\_BOTTLENECK Table**

The STREAMS\$\_PA\_PATH\_BOTTLENECK table displays information about Oracle Streams components that might be slowing down the flow of messages.

**Table 236–9 STREAMS\$\_PA\_PATH\_BOTTLENECK Table**

Column	Datatype	NULL	Description
ADVISOR_RUN_ID	NUMBER		Identification number of the Oracle Streams Performance Advisor run
ADVISOR_RUN_TIME	DATE		Time when the Oracle Streams Performance Advisor was last run
ADVISOR_RUN_REASON	VARCHAR2 (4000)		Reason for the bottleneck
PATH_ID	NUMBER		Identification number assigned to the path by the Oracle Streams Performance Advisor

**Table 236–9 (Cont.) STREAMS\$\_PA\_PATH\_BOTTLENECK Table**

Column	Datatype	NULL	Description
PATH_KEY	VARCHAR2 (4000)		Unique key assigned to the path by the Oracle Streams Performance Advisor
COMPONENT_ID	NUMBER		Identification number assigned to the component by the Oracle Streams Performance Advisor
TOP_SESSION_ID	NUMBER		Session ID of the top component. Query the V\$SESSION view for information about the session.
TOP_SESSION_SERIAL#	NUMBER		Session serial number of the top component. Query the V\$SESSION view for information about the session.
ACTION_NAME	VARCHAR2 (32)		Action name for the top session
BOTTLENECK_IDENTIFIED	VARCHAR2 (30)		Whether a bottleneck was identified

**STREAMS\$\_PA\_PATH\_STAT Table**

The STREAMS\$\_PA\_PATH\_STAT table displays performance statistics about each stream path.

**Table 236–10 STREAMS\$\_PA\_PATH\_STAT Table**

Column	Datatype	NULL	Description
ADVISOR_RUN_ID	NUMBER		Identification number of the Oracle Streams Performance Advisor run
ADVISOR_RUN_TIME	DATE		Time when the Oracle Streams Performance Advisor was run for the advisor run ID
PATH_ID	NUMBER		Identification number assigned to the path by the Oracle Streams Performance Advisor
PATH_KEY	VARCHAR2 (4000)		Unique key assigned to the path by the Oracle Streams Performance Advisor
STATISTIC_TIME	DATE		Time when the statistic was recorded
STATISTIC_NAME	VARCHAR2 (64)		Name of the statistic
STATISTIC_VALUE	NUMBER		Value recorded for the statistic
STATISTIC_UNIT	VARCHAR2 (64)		Unit of measurement for the statistic

**STREAMS\$\_PA\_SHOW\_COMP\_STAT Table**

The STREAMS\$\_PA\_SHOW\_COMP\_STAT table displays statistics for Oracle Streams components.

**Table 236–11 STREAMS\$\_PA\_SHOW\_COMP\_STAT Table**

Column	Datatype	NULL	Description
ADVISOR_RUN_ID	NUMBER		Identification number of the Oracle Streams Performance Advisor run

**Table 236–11 (Cont.) STREAMS\$\_PA\_SHOW\_COMP\_STAT Table**

Column	Datatype	NULL	Description
ADVISOR_RUN_TIME	DATE		Time when the Oracle Streams Performance Advisor was last run
PATH_ID	NUMBER		Identification number assigned to the path by the Oracle Streams Performance Advisor
POSITION	NUMBER		Position of the component in the path
COMPONENT_ID	NUMBER		Identification number assigned to the component by the Oracle Streams Performance Advisor
COMPONENT_NAME	VARCHAR2 (194)		Name of the component
COMPONENT_TYPE	VARCHAR2 (30)		Type of the component The following types are possible: <ul style="list-style-type: none"> <li>■ CAPTURE for a capture process</li> <li>■ PROPAGATION SENDER for a propagation sender</li> <li>■ PROPAGATION RECEIVER for a propagation receiver</li> <li>■ APPLY for an apply process</li> <li>■ QUEUE for a queue</li> </ul>
SUB_COMPONENT_TYPE	VARCHAR2 (30)		Type of the subcomponent Only capture processes and apply processes have subcomponents. The following capture process subcomponent types are possible: <ul style="list-style-type: none"> <li>■ LOGMINER READER for a builder server of a capture process</li> <li>■ LOGMINER PREPARER for a preparer server of a capture process</li> <li>■ LOGMINER BUILDER for a reader server of a capture process</li> <li>■ CAPTURE SESSION for a capture process session</li> </ul> The following apply process subcomponent types are possible: <ul style="list-style-type: none"> <li>■ PROPAGATION SENDER+RECEIVER for sending LCRs from a capture process directly to an apply process in a combined capture and apply configuration in which both the capture process and apply process run on a single database</li> <li>■ APPLY READER for a reader server of an apply process</li> <li>■ APPLY COORDINATOR for a coordinator process of an apply process</li> <li>■ APPLY SERVER for a reader server of an apply process</li> </ul>

**Table 236–11 (Cont.) STREAMS\$\_PA\_SHOW\_COMP\_STAT Table**

Column	Datatype	NULL	Description
SESSION_ID	NUMBER		Identification number of the session for the component. Query the V\$SESSION view for information about the session.
SESSION_SERIAL#	NUMBER		Session serial number of the session for the component. Query the V\$SESSION view for information about the session.
STATISTIC_ALIAS	VARCHAR2 (30)		Name of the statistic
STATISTIC_NAME	VARCHAR2 (128)		Name of the statistic
STATISTIC_VALUE	NUMBER		Value recorded for the statistic
STATISTIC_UNIT	VARCHAR2 (128)		Unit of measurement for the statistic

**STREAMS\$\_PA\_SHOW\_PATH\_STAT Table**

The STREAMS\$\_PA\_SHOW\_PATH\_STAT table displays statistics for the stream paths in an Oracle Streams configuration. A monitoring job uses this table as the default table for the statistics collected for stream paths.

**Table 236–12 STREAMS\$\_PA\_SHOW\_PATH\_STAT Table**

Column	Datatype	NULL	Description
PATH_ID	NUMBER		Identification number assigned to the path by the Oracle Streams Performance Advisor
ADVISOR_RUN_ID	NUMBER		Identification number of the Oracle Streams Performance Advisor run
ADVISOR_RUN_TIME	DATE		Time when the Oracle Streams Performance Advisor was last run
SETTING	VARCHAR2 (2000)		Setting for the Oracle Streams Performance Advisor Run
STATISTICS	VARCHAR2 (4000)		Component-level statistics
SESSION_STATISTICS	VARCHAR2 (4000)		Session-level statistics
OPTIMIZATION	NUMBER		Whether the path uses the combined capture and apply optimization  0 (zero) means that the path does not use the combined capture and apply optimization.  1 means that the path uses the combined capture and apply optimization.

## Summary of UTL\_SPADV Subprograms

**Table 236–13 DBMS\_STREAMS Package Subprograms**

Subprogram	Description
<a href="#">ALTER_MONITORING Procedure</a> on page 236-15	Alters the monitoring job submitted by the current user.
<a href="#">COLLECT_STATS Procedure</a> on page 236-17	Uses the Oracle Streams Performance Advisor to gather statistics about the Oracle Streams components and subcomponents in a distributed database environment.
<a href="#">IS_MONITORING Function</a> on page 236-19	Checks whether a monitoring job is currently running.
<a href="#">SHOW_STATS Procedure</a> on page 236-20	Generates output that includes the statistics gathered by the COLLECT_STATS procedure.
<a href="#">START_MONITORING Procedure</a> on page 236-23	Starts a monitoring job.
<a href="#">STOP_MONITORING Procedure</a> on page 236-26	Stops a monitoring job.

## ALTER\_MONITORING Procedure

This procedure alters the monitoring job submitted by the current user.

### Syntax

```
UTL_SPADV.ALTER_MONITORING(
    interval                IN NUMBER DEFAULT NULL,
    top_event_threshold     IN NUMBER DEFAULT NULL,
    bottleneck_idle_threshold IN NUMBER DEFAULT NULL,
    bottleneck_flowctrl_threshold IN NUMBER DEFAULT NULL,
    retention_time          IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 236–14 ALTER\_MONITORING Procedure Parameters**

Parameter	Description
interval	The amount of time, in seconds, between each Performance Advisor run. The maximum is 3600 seconds.  If NULL, then the current value is not changed.
top_event_threshold	A percentage that determines whether a top wait event statistic is collected.  The percentage for a wait event must be greater than the value specified in this parameter for the procedure to collect the wait event statistic. For example, if 15 is specified, then only wait events with a value larger than 15% are collected.  If NULL, then the current value is not changed.
bottleneck_idle_threshold	A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its IDLE percentage.  The IDLE percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are idle 50% of the time or less are eligible for bottleneck analysis.  If NULL, then the current value is not changed.
bottleneck_flowctrl_threshold	A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its FLOW CONTROL percentage.  The FLOW CONTROL percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are paused for flow control 50% of the time or less are eligible for bottleneck analysis.  If NULL, then the current value is not changed.
retention_time	The number of hours to retain monitoring results.  If NULL, then the current value is not changed.

## Exceptions

**Table 236–15** *ALTER\_MONITORING Procedure Exceptions*

<b>Exception</b>	<b>Description</b>
ORA-20113	no active monitoring job found



## COLLECT\_STATS Procedure

This procedure uses the Oracle Streams Performance Advisor to gather statistics about the Oracle Streams components and subcomponents in a distributed database environment.

---



---

**Note:** This procedure commits.

---



---

**See Also:** *Oracle Streams Concepts and Administration* for more information about the Oracle Streams Performance Advisor

### Syntax

```
UTL_SPADV.COLLECT_STATS (
    interval                IN NUMBER    DEFAULT 60,
    num_runs                IN NUMBER    DEFAULT 10,
    comp_stat_table        IN VARCHAR2   DEFAULT 'STREAMS$_ADVISOR_COMP_STAT',
    path_stat_table        IN VARCHAR2   DEFAULT 'STREAMS$_ADVISOR_PATH_STAT',
    top_event_threshold    IN NUMBER    DEFAULT 15,
    bottleneck_idle_threshold IN NUMBER  DEFAULT 50,
    bottleneck_flowctrl_threshold IN NUMBER DEFAULT 50);
```

### Parameters

**Table 236–16 COLLECT\_STATS Procedure Parameters**

Parameter	Description
interval	The amount of time, in seconds, between each Performance Advisor run. The maximum is 3600 seconds.
num_runs	The number of times that the Oracle Streams Performance Advisor is run by the procedure.
comp_stat_table	The name of the table that stores the statistics collected for Oracle Streams components and subcomponents. Specify the table name as <i>[schema_name.]object_name</i> . If the schema is not specified, then the current user is the default.  The procedure creates the specified table if it does not exist.  Oracle recommends that you use the default table STREAMS\$_ADVISOR_COMP_STAT.  See " <a href="#">Usage Notes</a> " on page 236-18 for more information about this parameter.
path_stat_table	The name of the table that stores the statistics collected for stream paths. Specify the table name as <i>[schema_name.]object_name</i> . If the schema is not specified, then the current user is the default.  The procedure creates the specified table if it does not exist.  Oracle recommends that you use the default table STREAMS\$_ADVISOR_PATH_STAT.  See " <a href="#">Usage Notes</a> " on page 236-18 for more information about this parameter.

**Table 236–16 (Cont.) COLLECT\_STATS Procedure Parameters**

Parameter	Description
<code>top_event_threshold</code>	<p>A percentage that determines whether a top wait event statistic is collected.</p> <p>The percentage for a wait event must be greater than the value specified in this parameter for the procedure to collect the wait event statistic. For example, if 15 is specified, then only wait events with a value larger than 15% are collected.</p>
<code>bottleneck_idle_threshold</code>	<p>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its IDLE percentage.</p> <p>The IDLE percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are idle 50% of the time or less are eligible for bottleneck analysis.</p>
<code>bottleneck_flowctrl_threshold</code>	<p>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its FLOW CONTROL percentage.</p> <p>The FLOW CONTROL percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are paused for flow control 50% of the time or less are eligible for bottleneck analysis.</p>

## Usage Notes

The table specified in the `path_stat_table` parameter stores stream path statistics. This table also concatenates the component and subcomponent statistics stored in the table specified in the `comp_stat_table` parameter. The `SHOW_STATS` procedure in this package shows only the statistics stored in the table specified in the `path_stat_table` parameter.

## IS\_MONITORING Function

This function checks whether a monitoring job is currently running. This function either returns `TRUE` if a monitoring job is currently running or `FALSE` if a monitoring job is not currently running.

A monitoring job is submitted using the `START_MONITORING` procedure.

**See Also:** "[START\\_MONITORING Procedure](#)" on page 236-23

### Syntax

```
UTL_SPADV.IS_MONITORING(  
    job_name      IN VARCHAR2  DEFAULT 'STREAMS$_MONITORING_JOB',  
    client_name   IN VARCHAR2  DEFAULT NULL)  
RETURN BOOLEAN;
```

### Parameters

**Table 236–17** *IS\_MONITORING Function Parameters*

Parameter	Description
<code>job_name</code>	The name of the job for which to check.
<code>client_name</code>	The name of the client that submitted the job.

## SHOW\_STATS Procedure

This procedure generates output that includes the statistics gathered by the `COLLECT_STATS` and `START_MONITORING` procedures.

The output is formatted so that it can be imported into a spreadsheet for analysis.

---



---

**Note:** This procedure does not commit.

---



---

### See Also:

- ["COLLECT\\_STATS Procedure"](#) on page 236-17
- ["START\\_MONITORING Procedure"](#) on page 236-23
- *Oracle Streams Concepts and Administration* for more information about the Oracle Streams Performance Advisor

## Syntax

```
UTL_SPADV.SHOW_STATS (
  path_stat_table IN VARCHAR2  DEFAULT 'STREAMS$_ADVISOR_PATH_STAT',
  path_id         IN NUMBER    DEFAULT NULL,
  bgn_run_id      IN NUMBER    DEFAULT -1,
  end_run_id      IN NUMBER    DEFAULT -10,
  show_path_id    IN BOOLEAN   DEFAULT TRUE,
  show_run_id     IN BOOLEAN   DEFAULT TRUE,
  show_run_time   IN BOOLEAN   DEFAULT TRUE,
  show_optimization IN BOOLEAN DEFAULT TRUE,
  show_setting    IN BOOLEAN   DEFAULT FALSE,
  show_stat       IN BOOLEAN   DEFAULT TRUE,
  show_sess       IN BOOLEAN   DEFAULT FALSE,
  show_legend     IN BOOLEAN   DEFAULT TRUE);
```

## Parameters

**Table 236–18** *SHOW\_STATS Procedure Parameters*

Parameter	Description
<code>path_stat_table</code>	<p>The name of the table that contains the stream path statistics. Specify the table name as <code>[schema_name.]object_name</code>. If the schema is not specified, then the current user is the default.</p> <p>When you gather statistics using the <code>COLLECT_STATS</code> procedure, this table is specified in the <code>path_stat_table</code> parameter in the <code>COLLECT_STATS</code> procedure. The default table is <code>STREAMS\$_ADVISOR_PATH_STAT</code>.</p> <p>When you gather statistics using the <code>START_MONITORING</code> procedure, you can determine the name for this table by querying the <code>SHOW_STATS_TABLE</code> column in the <code>STREAMS\$_PA_MONITORING</code> view. The default table for a monitoring job is <code>STREAMS\$_PA_SHOW_PATH_STAT</code>.</p>
<code>path_id</code>	<p>A stream path ID.</p> <p>If non-NULL, then the procedure shows output for the specified stream path only.</p> <p>If NULL, then the procedure shows output for all active stream paths.</p>

**Table 236–18 (Cont.) SHOW\_STATS Procedure Parameters**

Parameter	Description
bgn_run_id	The first Oracle Streams Performance Advisor run ID to show in the range of runs. See "Usage Notes" on page 236-22 for more information about this parameter.
end_run_id	The last Oracle Streams Performance Advisor run ID to show in the range of runs. See "Usage Notes" on page 236-22 for more information about this parameter.
show_path_id	If TRUE, then the path ID for each stream path is included in the output. If FALSE, then the path ID for each stream path is not included in the output.
show_run_id	If TRUE, then the Oracle Streams Performance Advisor run ID is included in the output. If FALSE, then the Oracle Streams Performance Advisor run ID is not included in the output.
show_run_time	If TRUE, then the Oracle Streams Performance Advisor run time is included in the output. If FALSE, then the Oracle Streams Performance Advisor run time is not included in the output.
show_optimization	If TRUE, then path output includes information pertaining to the combined capture and apply optimization. If FALSE, then path output does not include information pertaining to the combined capture and apply optimization.
show_setting	If TRUE, then the settings for the threshold parameters are included in the output. The threshold parameters are the <code>top_event_threshold</code> , <code>bottleneck_idle_threshold</code> , and <code>bottleneck_flowctrl_threshold</code> parameters in the <code>COLLECT_STATS</code> procedure. If FALSE, then the settings for the threshold parameters are not included in the output.
show_stat	If TRUE, then the component-level and subcomponent-level statistics are included in the output. These components include capture processes, queues, propagation senders, propagation receivers, and apply processes. The subcomponents are the subcomponents for capture processes and apply processes. If FALSE, then the component-level and subcomponent-level statistics are not included in the output.
show_sess	If TRUE, then the session-level statistics are included in the output. Session-level statistics include <code>IDLE</code> , <code>FLOW CONTROL</code> , and <code>EVENT</code> statistics. If FALSE, then the session-level statistics are not included in the output.
show_legend	If TRUE, then the legend is included in the output. The legend describes the abbreviations used in the output. If FALSE, then the legend is not included in the output.

## Usage Notes

Use the `bgn_run_id` and `end_run_id` together to specify the range of Oracle Streams Performance Advisor runs to display. Positive numbers show statistics from an earlier run forward. Negative numbers show statistics from a later run backward.

For example, if `bgn_run_id` is set to 1 and `end_run_id` is set to 10, then the procedure shows statistics for the first ten Oracle Streams Performance Advisor runs.

However, if `bgn_run_id` is set to -1 and `end_run_id` is set to -10, then the procedure shows statistics for the last ten Oracle Streams Performance Advisor runs.

**See Also:** *Oracle Streams Concepts and Administration* for information about the combined capture and apply optimization

## START\_MONITORING Procedure

This procedure starts a monitoring job.

This procedure runs the COLLECT\_STATS procedure to gather statistics about the Oracle Streams components and subcomponents in a distributed database environment.

---



---

**Note:** This procedure commits.

---



---

### See Also:

- ["COLLECT\\_STATS Procedure"](#) on page 236-17
- *Oracle Streams Concepts and Administration* for more information about the Oracle Streams Performance Advisor

## Syntax

```
UTL_SPADV.START_MONITORING(
  job_name          IN VARCHAR2  DEFAULT 'STREAMS$_MONITORING_JOB',
  client_name       IN VARCHAR2  DEFAULT NULL,
  query_user_name   IN VARCHAR2  DEFAULT NULL,
  interval          IN NUMBER    DEFAULT 60,
  top_event_threshold IN NUMBER   DEFAULT 15,
  bottleneck_idle_threshold IN NUMBER DEFAULT 50,
  bottleneck_flowctrl_threshold IN NUMBER DEFAULT 50,
  retention_time    IN NUMBER    DEFAULT 24);
```

## Parameters

**Table 236–19 START\_MONITORING Procedure Parameters**

Parameter	Description
job_name	The name of the monitoring job to create.
client_name	The name of the client.
query_user_name	The user who will query the result tables. This procedure grants privileges to the specified user to enable the user to query the result tables.
interval	The amount of time, in seconds, between each Performance Advisor run. The maximum is 3600 seconds. The specified interval is used for the interval parameter in the COLLECT_STATS procedure.
top_event_threshold	A percentage that determines whether a top wait event statistic is collected. The percentage for a wait event must be greater than the value specified in this parameter for the procedure to collect the wait event statistic. For example, if 15 is specified, then only wait events with a value larger than 15% are collected.

**Table 236–19 (Cont.) START\_MONITORING Procedure Parameters**

Parameter	Description
bottleneck_idle_threshold	<p>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its IDLE percentage.</p> <p>The IDLE percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are idle 50% of the time or less are eligible for bottleneck analysis.</p>
bottleneck_flowctrl_threshold	<p>A percentage that determines whether an Oracle Streams component session is eligible for bottleneck analysis based on its FLOW CONTROL percentage.</p> <p>The FLOW CONTROL percentage must be less than or equal to the value specified in this parameter for the Oracle Streams component session to be eligible for bottleneck analysis. For example, if 50 is specified, then only components that are paused for flow control 50% of the time or less are eligible for bottleneck analysis.</p>
retention_time	The number of hours to retain monitoring results.

## Exceptions

**Table 236–20 START\_MONITORING Procedure Exceptions**

Exception	Description
ORA-20111	<p>cannot start monitoring due to active EM monitoring job</p> <p>Stop the Oracle Enterprise Manager (EM) monitoring job, and run the START_MONITORING procedure again.</p>
ORA-20112	<p>cannot start monitoring due to active Streams monitoring job</p> <p>Stop the Streams monitoring job, and run the START_MONITORING procedure again.</p>

## Usage Notes

The following are usage notes for the START\_MONITORING procedure:

- [Requirements for the User Running the Procedure](#)
- [Full Monitoring Job Names](#)
- [Restrictions on Monitoring Jobs](#)

### Requirements for the User Running the Procedure

The user who runs the START\_MONITORING procedure must meet the following requirements:

- The user must have access to a database link to each database that contains Oracle Streams components.
- The user must have been granted privileges using the DBMS\_STREAMS\_AUTH.GRANT\_ADMIN\_PRIVILEGE procedure, and each database link must connect to a user at the remote database that has been granted privileges using the DBMS\_STREAMS\_AUTH.GRANT\_ADMIN\_PRIVILEGE procedure.



### Full Monitoring Job Names

When you submit a monitoring job, the client name and job name are concatenated to form the full monitoring job name. You specify the client name using the `client_name` parameter and the job name using the `job_name` parameter when you run the `START_MONITORING` procedure. The client name for a monitoring job submitted by Oracle Enterprise Manager is always EM.

The following table show examples of full monitoring job names:

Setting for client_name Parameter	Setting for job_name parameter	Full Monitoring Job Name
NULL	STREAMS\$_MONITORING_JOB	STREAMS\$_MONITORING_JOB
EM	STREAMS\$_MONITORING_JOB	EMSTREAMS\$_MONITORING_JOB
strm	STREAMS\$_MONITORING_JOB	strmSTREAMS\$_MONITORING_JOB
strm	mjob1	strmmjob1

### Restrictions on Monitoring Jobs

The following restrictions apply to monitoring jobs:

- The limit for the length of the full monitoring job name is 30 bytes.
- Two monitoring jobs cannot have the same full monitoring job name, even if the monitoring jobs were submitted by different schemas. The name check is not case-sensitive. For example, `strmSTREAMS$_MONITORING_JOB` and `STRMSTREAMS$_MONITORING_JOB` are considered to be the same name.
- Oracle Enterprise Manager can have at most one monitoring job for each database.
- Each schema can have at most one monitoring job.

## STOP\_MONITORING Procedure

This procedure stops a monitoring job that was submitted by the current user.

### Syntax

```
UTL_SPADV.STOP_MONITORING(  
    purge IN BOOLEAN DEFAULT FALSE);
```

### Parameters

**Table 236–21 STOP\_MONITORING Procedure Parameters**

Parameter	Description
purge	If TRUE, then the procedure purges information about the monitoring job from the result tables.  If FALSE, then the procedure retains information about the monitoring job in the result tables.

With the UTL\_TCP package and its procedures and functions, PL/SQL applications can communicate with external TCP/IP-based servers using TCP/IP. Because many Internet application protocols are based on TCP/IP, this package is useful to PL/SQL applications that use Internet protocols and e-mail.

This chapter contains the following topics:

- [Using UTL\\_TCP](#)
  - Overview
  - Security Model
  - Types
  - Exceptions
  - Rules and Limits
  - Examples
- [Summary of UTL\\_TCP Subprograms](#)

## Using UTL\_TCP

- [Overview](#)
- [Security Model](#)
- [Types](#)
- [Exceptions](#)
- [Rules and Limits](#)
- [Examples](#)

## Overview

The `UTL_TCP` package provides TCP/IP client-side access functionality in PL/SQL.

## Security Model

This package is an invoker's rights package and the invoking user needs the connect privilege granted in the access control list assigned to the remote network host to which he wants to connect.

---

---

**Note:** For more information about managing fine-grained access, see Oracle Database Security Guide

---

---

## Types

- [CONNECTION Type](#)
- [CRLF](#)

### CONNECTION Type

This is a PL/SQL record type used to represent a TCP/IP connection.

#### Syntax

```
TYPE connection IS RECORD (
  remote_host    VARCHAR2(255) ,
  remote_port    PLS_INTEGER,
  local_host     VARCHAR2(255) ,
  local_port     PLS_INTEGER,
  charset        VARCHAR2(30) ,
  newline        VARCHAR2(2) ,
  tx_timeout     PLS_INTEGER,
  private_sd     PLS_INTEGER);
```

#### Fields

**Table 237–1 Connection Record Type Fields**

Field	Description
remote_host	Name of the remote host when connection is established. NULL when no connection is established.
remote_port	Port number of the remote host connected. NULL when no connection is established.
local_host	Name of the local host used to establish the connection. NULL when no connection is established.
local_port	Port number of the local host used to establish the connection. NULL when no connection is established.
charset	The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database are converted to and from the on-the-wire character set as they are sent and received on the network.
newline	Newline character sequence. This newline character sequence is appended to the text line sent by <code>WRITE_LINE</code> API.
tx_timeout	Time in seconds that the UTL_TCP package waits before giving up in a read or write operation in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

#### Usage Notes

The fields in a connection record are used to return information about the connection, which is often made using `OPEN_CONNECTION`. Changing the values of those fields

has no effect on the connection. The fields `private_XXXX` are for implementation use only. You should not modify the values.

In the current release of the `UTL_TCP` package, the parameters `local_host` and `local_port` are ignored when `open_connection` makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The `local_host` and `local_port` fields are not set in the connection record returned by the function.

Time out on write operations is not supported in the current release of the `UTL_TCP` package.

## CRLF

The character sequence carriage-return line-feed. It is the newline sequence commonly used by many communication standards.

### Syntax

```
CRLF CONSTANT VARCHAR2(2 CHAR);
```

### Usage Notes

This package variable defines the newline character sequence commonly used in many Internet protocols. This is the default value of the newline character sequence for `WRITE_LINE`, specified when a connection is opened. While such protocols use `<CR><LF>` to denote a new line, some implementations may choose to use just line-feed to denote a new line. In such cases, users can specify a different newline character sequence when a connection is opened.



## Exceptions

The exceptions raised by the TCP/IP package are listed in [Table 237-2](#).

**Table 237-2 TCP/IP Exceptions**

<b>Exception</b>	<b>Description</b>
BUFFER_TOO_SMALL	Buffer is too small for input that requires look-ahead
END_OF_INPUT	Raised when no more data is available to read from the connection
NETWORK_ERROR	Generic network error
BAD_ARGUMENT	Bad argument passed in an API call (for example, a negative buffer size)
TRANSFER_TIMEOUT	No data is read and a read time out occurred
PARTIAL_MULTIBYTE_CHAR	No complete character is read and a partial multibyte character is found at the end of the input

## Rules and Limits

The interface provided in the package only allows connections to be initiated by the PL/SQL program. It does not allow the PL/SQL program to accept connections initiated outside the program.

## Examples

The following code example illustrates how the TCP/IP package can be used to retrieve a Web page over HTTP. It connects to a Web server listening at port 80 (standard port for HTTP) and requests the root document.

```

DECLARE
  c utl_tcp.connection; -- TCP/IP connection to the Web server
  ret_val pls_integer;
BEGIN
  c := utl_tcp.open_connection(remote_host => 'www.acme.com',
                              remote_port => 80,
                              charset => 'US7ASCII'); -- open connection
  ret_val := utl_tcp.write_line(c, 'GET / HTTP/1.0'); -- send HTTP request
  ret_val := utl_tcp.write_line(c);
  BEGIN
    LOOP
      dbms_output.put_line(utl_tcp.get_line(c, TRUE)); -- read result
    END LOOP;
  EXCEPTION
    WHEN utl_tcp.end_of_input THEN
      NULL; -- end of input
  END;
  utl_tcp.close_connection(c);
END;

```

The following code example illustrates how the TCP/IP package can be used by an application to send e-mail (also known as email from PL/SQL). The application connects to an SMTP server at port 25 and sends a simple text message.

```

PROCEDURE send_mail (sender IN VARCHAR2,
                    recipient IN VARCHAR2,
                    message IN VARCHAR2)
IS
  mailhost VARCHAR2(30) := 'mailhost.mydomain.com';
  smtp_error EXCEPTION;
  mail_conn utl_tcp.connection;
  PROCEDURE smtp_command(command IN VARCHAR2,
                        ok IN VARCHAR2 DEFAULT '250')
  IS
    response varchar2(3);
    len pls_integer;
  BEGIN
    len := utl_tcp.write_line(mail_conn, command);
    response := substr(utl_tcp.get_line(mail_conn), 1, 3);
    IF (response <> ok) THEN
      RAISE smtp_error;
    END IF;
  END;
END;

BEGIN
  mail_conn := utl_tcp.open_connection(remote_host => mailhost,
                                      remote_port => 25,
                                      charset => 'US7ASCII');

  smtp_command('HELO ' || mailhost);
  smtp_command('MAIL FROM: ' || sender);
  smtp_command('RCPT TO: ' || recipient);
  smtp_command('DATA', '354');
  smtp_command(message);

```

```
smtp_command('QUIT', '221');
utl_tcp.close_connection(mail_conn);
EXCEPTION
  WHEN OTHERS THEN
    -- Handle the error
END;
```

---

## Summary of UTL\_TCP Subprograms

**Table 237-3 UTL\_TCP Package Subprograms**

Subprogram	Description
<a href="#">AVAILABLE Function</a> on page 237-12	Determines the number of bytes available for reading from a TCP/IP connection
<a href="#">CLOSE_ALL_CONNECTIONS Procedure</a> on page 237-14	Closes all open TCP/IP connections
<a href="#">CLOSE_CONNECTION Procedure</a> on page 237-15	Closes an open TCP/IP connection
<a href="#">FLUSH Procedure</a> on page 237-16	Transmits immediately to the server all data in the output buffer, if a buffer is used
<a href="#">GET_LINE Function</a> on page 237-17	Returns the line of data read
<a href="#">GET_LINE_NCHAR Function</a> on page 237-18	Returns the line of data read in NCHAR form
<a href="#">GET_RAW Function</a> on page 237-19	Return the data read instead of the amount of data read
<a href="#">GET_TEXT Function</a> on page 237-20	Returns the text data read
<a href="#">GET_TEXT_NCHAR Function</a> on page 237-21	Returns the text data read in NCHAR form
<a href="#">OPEN_CONNECTION Function</a> on page 237-22	Opens a TCP/IP connection to a specified service
<a href="#">READ_LINE Function</a> on page 237-25	Receives a text line from a service on an open connection
<a href="#">READ_RAW Function</a> on page 237-27	Receives binary data from a service on an open connection
<a href="#">READ_TEXT Function</a> on page 237-28	Receives text data from a service on an open connection
<a href="#">SECURE_CONNECTION Procedure</a> on page 237-30	Secures a TCP/IP connection using SSL/TLS
<a href="#">WRITE_LINE Function</a> on page 237-31	Transmits a text line to a service on an open connection
<a href="#">WRITE_RAW Function</a> on page 237-32	Transmits a binary message to a service on an open connection
<a href="#">WRITE_TEXT Function</a> on page 237-33	Transmits a text message to a service on an open connection

---

## AVAILABLE Function

This function determines the number of bytes available for reading from a TCP/IP connection. It is the number of bytes that can be read immediately without blocking. Determines if data is ready to be read from the connection.

### Syntax

```
UTL_TCP.AVAILABLE (
  c          IN OUT NOCOPY connection,
  timeout   IN PLS_INTEGER DEFAULT 0)
RETURN PLS_INTEGER;
```

### Parameters

**Table 237–4 AVAILABLE Function Parameters**

Parameter	Description
c	TCP connection to determine the amount of data that is available to be read
timeout	Time in seconds to wait before giving up and reporting that no data is available. Zero (0) indicates not to wait at all. NULL indicates to wait forever.

### Return Values

The number of bytes available for reading without blocking

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. Users may use this API to determine if data is available to be read before calling the read API so that the program are not blocked because data is not ready to be read from the input.

The number of bytes available for reading returned by this function may be less than what is actually available. On some platforms, this function may only return 1, to indicate that some data is available. If you are concerned about the portability of your application, then assume that this function returns a positive value when data is available for reading, and 0 when no data is available. This function returns a positive value when all the data at a particular connection has been read and the next read result in the `END_OF_INPUT` exception.

The following example illustrates using this function in a portable manner:

```
DECLARE
  c    utl_tcp.connection
  data VARCHAR2(256);
  len  PLS_INTEGER;
BEGIN
  c := utl_tcp.open_connection(...);
  LOOP
    IF (utl_tcp.available(c) > 0) THEN
      len := utl_tcp.read_text(c, data, 256);
    ELSE
      ---do some other things
      . . .
    END IF
```

```
    END LOOP;  
END;
```

## **CLOSE\_ALL\_CONNECTIONS Procedure**

This procedure closes all open TCP/IP connections.

### **Syntax**

```
UTL_TCP.CLOSE_ALL_CONNECTIONS;
```

### **Usage Notes**

This call is provided to close all connections before a PL/SQL program ends to avoid dangling connections.



## CLOSE\_CONNECTION Procedure

This procedure closes an open TCP/IP connection.

### Syntax

```
UTL_TCP.CLOSE_CONNECTION (  
    c IN OUT NOCOPY connection);
```

### Parameters

**Table 237-5** *CLOSE\_CONNECTION Procedure Parameters*

Parameter	Description
c	TCP connection to close

### Usage Notes

Connection must have been opened by a previous call to OPEN\_CONNECTION. The fields `remote_host`, `remote_port`, `local_host`, `local_port` and `charset` of `c` are reset after the connection is closed.

An open connection must be closed explicitly. An open connection remains open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.

## FLUSH Procedure

This procedure transfers immediately to the server all data in the output buffer, if a buffer is used.

### Syntax

```
UTL_TCP.FLUSH (  
    c IN OUT NOCOPY connection);
```

### Parameters

**Table 237–6** *FLUSH Procedure Parameters*

Parameter	Description
c	TCP connection to which to send data

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`.

## GET\_LINE Function

This function returns the line of data read.

### Syntax

```
UTL_TCP.GET_LINE (
  c          IN OUT NOCOPY connection,
  remove_crlf IN          BOOLEAN DEFAULT FALSE,
  peek      IN          BOOLEAN DEFAULT FALSE)
RETURN VARCHAR2;
```

### Parameters

**Table 237-7 GET\_LINE Function Parameters**

Parameter	Description
c	TCP connection from which to receive data
remove_crlf	If TRUE, then one or more trailing CRLF characters are removed from the received message.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.

### Return Values

The text line read

### Usage Notes

- The connection must have already been opened through a call to OPEN\_CONNECTION.
- See READ\_LINE for the read time out, character set conversion, buffer size, and multibyte character issues.

## GET\_LINE\_NCHAR Function

This function returns the line of data read in NCHAR form.

### Syntax

```
UTL_TCP.GET_LINE_NCHAR (
  c          IN OUT NOCOPY connection,
  remove_crlf IN          BOOLEAN DEFAULT FALSE,
  peek      IN          BOOLEAN DEFAULT FALSE)
RETURN NVARCHAR2;
```

### Parameters

**Table 237–8** GET\_LINE\_NCHAR Function Parameters

Parameter	Description
c	TCP connection from which to receive data
remove_crlf	If TRUE, then one or more trailing CRLF characters are removed from the received message.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.

### Return Values

The text line read

### Usage Notes

- The connection must have already been opened through a call to OPEN\_CONNECTION.
- See READ\_LINE for the read time out, character set conversion, buffer size, and multibyte character issues.

## GET\_RAW Function

This function returns the data read instead of the amount of data read.

### Syntax

```
UTL_TCP.GET_RAW (
  c      IN OUT NOCOPY connection,
  len   IN          PLS_INTEGER DEFAULT 1,
  peek  IN          BOOLEAN      DEFAULT FALSE)
RETURN RAW;
```

### Parameters

**Table 237–9 GET\_RAW Function Parameters**

Parameter	Description
c	TCP connection from which to receive data
len	The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.
remove_crlf	If TRUE, then one or more trailing CRLF characters are removed from the received message.

### Return Values

The binary data read

### Usage Notes

The connection must have already been opened through a call to OPEN\_CONNECTION.

For all the get\_\* APIs described in this section, see the corresponding READ\_\* API for the read time out issue. For GET\_TEXT and GET\_LINE, see the corresponding READ\_\* API for character set conversion, buffer size, and multibyte character issues.

## GET\_TEXT Function

This function returns the text data read.

### Syntax

```
UTL_TCP.GET_TEXT (
  c      IN OUT NOCOPY connection,
  len   IN          PLS_INTEGER DEFAULT 1,
  peek  IN          BOOLEAN     DEFAULT FALSE)
RETURN VARCHAR2;
```

### Parameters

**Table 237–10** GET\_TEXT Function Parameters

Parameter	Description
c	TCP connection from which to receive data
len	Number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.
remove_crlf	If TRUE, then one or more trailing CRLF characters are removed from the received message.

### Return Values

The text data read

### Usage Notes

The connection must have already been opened through a call to OPEN\_CONNECTION.

For all the get\_\* APIs described in this section, see the corresponding read\_\* API for the read time out issue. For GET\_TEXT and GET\_LINE, see the corresponding READ\_\* API for character set conversion, buffer size, and multibyte character issues.

## GET\_TEXT\_NCHAR Function

This function returns the text data read in NCHAR form.

### Syntax

```
UTL_TCP.GET_TEXT_NCHAR (
  c      IN OUT NOCOPY connection,
  len   IN          PLS_INTEGER DEFAULT 1,
  peek  IN          BOOLEAN      DEFAULT FALSE)
RETURN NVARCHAR2;
```

### Parameters

**Table 237-11** GET\_TEXT\_NCHAR Function Parameters

Parameter	Description
c	TCP connection from which to receive data
len	The number of bytes (or characters for VARCHAR2) of data to receive. Default is 1.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.
remove_crlf	If TRUE, then one or more trailing CRLF characters are removed from the received message.

### Return Values

The text data read

### Usage Notes

The connection must have already been opened through a call to OPEN\_CONNECTION.

For all the get\_\* APIs described in this section, see the corresponding read\_\* API for the read time out issue. For GET\_TEXT and GET\_LINE, see the corresponding READ\_\* API for character set conversion, buffer size, and multibyte character issues.

## OPEN\_CONNECTION Function

This function opens a TCP/IP connection to a specified service.

---



---

**Note:** The functionality associated with `wallet_path` and `wallet_password` is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

### Syntax

```
UTL_TCP.OPEN_CONNECTION (
    remote_host      IN VARCHAR2,
    remote_port      IN PLS_INTEGER,
    local_host       IN VARCHAR2 DEFAULT NULL,
    local_port       IN PLS_INTEGER DEFAULT NULL,
    in_buffer_size   IN PLS_INTEGER DEFAULT NULL,
    out_buffer_size  IN PLS_INTEGER DEFAULT NULL,
    charset          IN VARCHAR2 DEFAULT NULL,
    newline          IN VARCHAR2 DEFAULT CRLF,
    tx_timeout       IN PLS_INTEGER DEFAULT NULL,
    wallet_path      IN VARCHAR2 DEFAULT NULL,
    wallet_password  IN VARCHAR2 DEFAULT NULL,
    RETURN connection;
```

### Parameters

**Table 237–12 OPEN\_CONNECTION Function Parameters**

Parameter	Description
<code>remote_host</code>	Name of the host providing the service. When <code>remote_host</code> is NULL, it connects to the local host.
<code>remote_port</code>	Port number on which the service is listening for connections
<code>local_host</code>	Name of the host providing the service. NULL means does not care.
<code>local_port</code>	Port number on which the service is listening for connections. NULL means don't care.
<code>in_buffer_size</code>	The size of input buffer. The use of an input buffer can speed up execution performance in receiving data from the server. The appropriate size of the buffer depends on the flow of data between the client and the server, and the traffic/latency on the network. A zero value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the input buffer is 32767 bytes.
<code>out_buffer_size</code>	The size of output buffer. The use of an output buffer can speed up execution performance in sending data to the server. The appropriate size of buffer depends on the flow of data between the client and the server, and the network condition. A zero value means no buffer should be used. A NULL value means the caller does not care if a buffer is used or not. The maximum size of the output buffer is 32767 bytes.



**Table 237–12 (Cont.) OPEN\_CONNECTION Function Parameters**

Parameter	Description
charset	The on-the-wire character set. Since text messages in the database may be encoded in a character set that is different from the one expected on the wire (that is, the character set specified by the communication protocol, or the one stipulated by the other end of the communication), text messages in the database are converted to and from the on-the-wire character set as they are sent and received on the network using <code>READ_TEXT</code> , <code>READ_LINE</code> , <code>WRITE_TEXT</code> and <code>WRITE_LINE</code> . Set this parameter to <code>NULL</code> when no conversion is needed.
newline	Newline character sequence. This newline character sequence is appended to the text line sent by <code>WRITE_LINE</code> API.
tx_timeout	Time in seconds that the <code>UTL_TCP</code> package should wait before giving up in a read or write operations in this connection. In read operations, this package gives up if no data is available for reading immediately. In write operations, this package gives up if the output buffer is full and no data is to be sent in the network without being blocked. Zero (0) indicates not to wait at all. <code>NULL</code> indicates to wait forever.
wallet_path	Directory path that contains the Oracle wallet for SSL/TLS. The format is <code>file:directory-path</code>
wallet_password	Password to open the wallet. When the wallet is auto-login enabled, the password may be set to <code>NULL</code> .

## Return Values

A connection to the targeted TCP/IP service

## Usage Notes

- Note that connections opened by this `UTL_TCP` package can remain open and be passed from one database call to another in a shared server configuration. However, the connection must be closed explicitly. The connection remains open when the PL/SQL record variable that stores the connection goes out-of-scope in the PL/SQL program. Failing to close unwanted connections may result in unnecessary tying up of local and remote system resources.
- In the current release of the `UTL_TCP` package, the parameters `local_host` and `local_port` are ignored when `open_connection` makes a TCP/IP connection. It does not attempt to use the specified local host and port number when the connection is made. The `local_host` and `local_port` fields is not set in the connection record returned by the function.
- `tx_timeout` is intended to govern both the read operations and the write operations. However, an implementation restriction prevents `tx_timeout` from governing write operations in the current release.

## Examples

```
DECLARE
  c UTL_TCP.CONNECTION;
BEGIN
  c := UTL_TCP.OPEN_CONNECTION(
    host      => 'www.example.com',
    port      => 443,
    wallet_path => 'file:/oracle/wallets/smtp_wallet',
    wallet_password => '*****');
```

```
    UTL_TCP.SECURE_CONNECTION (c => c);  
END;
```

## READ\_LINE Function

This function receives a text line from a service on an open connection. A line is terminated by a line-feed, a carriage-return or a carriage-return followed by a line-feed.

### Syntax

```
UTL_TCP.READ_LINE (
  c          IN OUT NOCOPY connection,
  data      IN OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
  peek      IN          BOOLEAN DEFAULT FALSE)
RETURN PLS_INTEGER;
```

### Parameters

**Table 237–13 READ\_LINE Function Parameters**

Parameter	Description
c	TCP connection from which to receive data
data	Data received.
remove_crlf	If TRUE, then one or more trailing CRLF characters are removed from the received message.
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to TRUE and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.

### Return Values

The number of characters of data received

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. This function does not return until the end-of-line have been reached, or the end of input has been reached. Text messages is converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

If transfer time out is set when the connection is opened, then this function waits for each data packet to be ready to read until time out occurs. If it occurs, then this function stops reading and returns all the data read successfully. If no data is read successfully, then the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, then this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, then the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `READ_RAW` function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of

the character have not arrived and read time out occurs, then the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

## READ\_RAW Function

This function receives binary data from a service on an open connection.

### Syntax

```
UTL_TCP.READ_RAW (
  c      IN OUT NOCOPY connection,
  data   IN OUT NOCOPY RAW,
  len    IN          PLS_INTEGER DEFAULT 1,
  peek   IN          BOOLEAN      DEFAULT FALSE)
RETURN PLS_INTEGER;
```

### Parameters

**Table 237–14 READ\_RAW Function Parameters**

Parameter	Description
c	TCP connection from which to receive data
data (IN OUT COPY)	Data received
len	Number of bytes of data to receive
peek	Normally, you want to read the data and remove it from the input queue, that is, consume it. In some situations, you may just want to look ahead at the data, that is, peek at it, without removing it from the input queue, so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to <code>TRUE</code> and set up an input buffer before the connection is opened. The amount of data you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.

### Return Values

The number of bytes of data received

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. This function does not return until the specified number of bytes have been read, or the end of input has been reached.

If transfer time out is set when the connection is opened, then this function waits for each data packet to be ready to read until time out occurs. If it occurs, then this function stops reading and returns all the data read successfully. If no data is read successfully, then the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

## READ\_TEXT Function

This function receives text data from a service on an open connection.

### Syntax

```
UTL_TCP.READ_TEXT (
  c      IN OUT NOCOPY connection,
  data   IN OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
  len    IN          PLS_INTEGER DEFAULT 1,
  peek   IN          BOOLEAN      DEFAULT FALSE)
RETURN PLS_INTEGER;
```

### Parameters

**Table 237–15 READ\_TEXT Function Parameters**

Parameter	Description
c	TCP connection from which to receive data
data	Data received
len	Number of characters of data to receive
peek	Normally, users want to read the data and remove it from the input queue, that is, consume it. In some situations, users may just want to look ahead at the data without removing it from the input queue so that it is still available for reading (or even peeking) in the next call. To keep the data in the input queue, set this flag to <code>TRUE</code> and an input buffer must be set up when the connection is opened. The amount of data that you can peek at (that is, read but keep in the input queue) must be less than the size of input buffer.

### Return Values

The number of characters of data received

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. This function does not return until the specified number of characters has been read, or the end of input has been reached. Text messages is converted from the on-the-wire character set, specified when the connection was opened, to the database character set before they are returned to the caller.

Unless explicitly overridden, the size of a `VARCHAR2` buffer is specified in terms of bytes, while the parameter `len` refers to the maximum number of characters to be read. When the database character set is multibyte, where a single character may consist of more than 1 byte, you should ensure that the buffer can hold the maximum of characters. In general, the size of the `VARCHAR2` buffer should equal the number of characters to be read, multiplied by the maximum number of bytes of a character of the database character set.

If transfer time out is set when the connection is opened, then this function waits for each data packet to be ready to read until time out occurs. If it occurs, then this function stops reading and returns all the data read successfully. If no data is read successfully, then the `transfer_timeout` exception is raised. The exception can be handled and the read operation can be retried later.

If a partial multibyte character is found at the end of input, then this function stops reading and returns all the complete multibyte characters read successfully. If no complete character is read successfully, then the `partial_multibyte_char` exception is raised. The exception can be handled and the bytes of that partial multibyte character can be read as binary by the `READ_RAW` function. If a partial multibyte character is seen in the middle of the input because the remaining bytes of the character have not arrived and read time out occurs, then the `transfer_timeout` exception is raised instead. The exception can be handled and the read operation can be retried later.

## SECURE\_CONNECTION Procedure

This procedure secures a TCP/IP connection using SSL/TLS. SSL/TLS requires an Oracle wallet which must be specified when the connection was opened by the [OPEN\\_CONNECTION Function](#).

### Syntax

```
UTL_TCP.SECURE_CONNECTION (  
    c      IN OUT NOCOPY connection);
```

### Parameters

**Table 237–16** SECURE\_CONNECTION Procedure Parameters

Parameter	Description
c	TCP connection from which to receive data



## WRITE\_LINE Function

This function transmits a text line to a service on an open connection. The `newline` character sequence is appended to the message before it is transmitted.

### Syntax

```
UTL_TCP.WRITE_LINE (
  c      IN OUT NOCOPY connection,
  data IN          VARCHAR2 DEFAULT NULL CHARACTER SET ANY_CS)
RETURN PLS_INTEGER;
```

### Parameters

**Table 237–17** *WRITE\_LINE Function Parameters*

Parameter	Description
<code>c</code>	TCP connection to which to send data
<code>data</code>	Buffer containing the data to be sent

### Return Values

The actual number of characters of data transmitted

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. Text messages are converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.

## WRITE\_RAW Function

This function transmits a binary message to a service on an open connection. The function does not return until the specified number of bytes have been written.

### Syntax

```
UTL_TCP.WRITE_RAW (  
  c      IN OUT NOCOPY connection,  
  data  IN          RAW,  
  len   IN          PLS_INTEGER DEFAULT NULL)  
RETURN PLS_INTEGER;
```

### Parameters

**Table 237–18** WRITE\_RAW Function Parameters

Parameter	Description
c	TCP connection to which to send data
data	Buffer containing the data to be sent
len	The number of bytes of data to transmit. When len is NULL, the whole length of data is written.

### Return Values

The number of bytes of data transmitted

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`.

## WRITE\_TEXT Function

This function transmits a text message to a service on an open connection.

### Syntax

```
UTL_TCP.WRITE_TEXT (
  c      IN OUT NOCOPY connection,
  data IN          VARCHAR2 CHARACTER SET ANY_CS,
  len IN          PLS_INTEGER DEFAULT NULL)
RETURN num_chars PLS_INTEGER;
```

### Parameters

**Table 237–19** *WRITE\_TEXT Function Parameters*

Parameter	Description
c	TCP connection to which to send data
data	Buffer containing the data to be sent
len	The number of characters of data to transmit. When len is NULL, the whole length of data is written. The actual amount of data written may be less because of network condition.

### Return Values

The actual number of characters of data transmitted

### Usage Notes

The connection must have already been opened through a call to `OPEN_CONNECTION`. Text messages are converted to the on-the-wire character set, specified when the connection was opened, before they are transmitted on the wire.



The UTL\_URL package has two functions: ESCAPE and UNESCAPE.

**See Also:** [Chapter 224, "UTL\\_HTTP"](#)

This chapter contains the following topics:

- [Using UTL\\_URL](#)
  - Overview
  - Exceptions
  - Examples
- [Summary of UTL\\_URL Subprograms](#)

## Using UTL\_URL

- [Overview](#)
- [Exceptions](#)
- [Examples](#)

## Overview

A Uniform Resource Locator (URL) is a string that identifies a Web resource, such as a page or a picture. Use a URL to access such resources by way of the HyperText Transfer Protocol (HTTP). For example, the URL for Oracle's Web site is:

```
http://www.oracle.com
```

Normally, a URL contains English alphabetic characters, digits, and punctuation symbols. These characters are known as the *unreserved characters*. Any other characters in URLs, including multibyte characters or binary octet codes, must be escaped to be accurately processed by Web browsers or Web servers. Some punctuation characters, such as dollar sign (\$), question mark (?), colon (:), and equals sign (=), are reserved as delimiters in a URL. They are known as the *reserved characters*. To literally process these characters, instead of treating them as delimiters, they must be escaped.

The unreserved characters are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (\_), period (.), exclamation point (!), tilde (~), asterisk (\*), accent ('), left parenthesis ( ( ), right parenthesis ( ) )

The reserved characters are:

- Semi-colon (;) slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign (\$), percentage sign (%), and comma (,)

The UTL\_URL package has two functions that provide escape and unescape mechanisms for URL characters. Use the escape function to escape a URL before the URL is used to fetch a Web page by way of the UTL\_HTTP package. Use the unescape function to unescape an escaped URL before information is extracted from the URL.

For more information, refer to the Request For Comments (RFC) document RFC2396. Note that this URL escape and unescape mechanism is different from the x-www-form-urlencoded encoding mechanism described in the HTML specification:

```
http://www.w3.org/TR/html
```

## Exceptions

[Table 238–1](#) lists the exceptions that can be raised when the `UTL_URL` package API is invoked.

**Table 238–1** *UTL\_URL Exceptions*

<b>Exception</b>	<b>Error Code</b>	<b>Reason</b>
<code>BAD_URL</code>	29262	The URL contains badly formed escape code sequences
<code>BAD_FIXED_WIDTH_CHARSET</code>	29274	Fixed-width multibyte character set is not allowed as a URL character set.



## Examples

You can implement the `x-www-form-urlencoded` encoding using the `UTL_URL.ESCAPE` function as follows:

```
CREATE OR REPLACE FUNCTION form_url_encode (
    data    IN VARCHAR2,
    charset IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
    RETURN utl_url.escape(data, TRUE, charset); -- note use of TRUE
END;
```

For decoding data encoded with the `form-URL-encode` scheme, the following function implements the decoding scheme:

```
CREATE OR REPLACE FUNCTION form_url_decode(
    data    IN VARCHAR2,
    charset IN VARCHAR2) RETURN VARCHAR2 AS
BEGIN
    RETURN utl_url.unescape(
        replace(data, '+', ' '),
        charset);
END;
```

## Summary of UTL\_URL Subprograms

---

**Table 238–2 UTL\_URL Package Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">ESCAPE Function</a> on page 238-7	Returns a URL with illegal characters (and optionally reserved characters) escaped using the %2-digit-hex-code format
<a href="#">UNESCAPE Function</a> on page 238-9	Unescapes the escape character sequences to their original forms in a URL. Convert the %xx escape character sequences to the original characters

## ESCAPE Function

This function returns a URL with illegal characters (and optionally reserved characters) escaped using the %2-digit-hex-code format.

### Syntax

```
UTL_URL.ESCAPE (
  url                IN VARCHAR2 CHARACTER SET ANY_CS,
  escape_reserved_chars IN BOOLEAN DEFAULT FALSE,
  url_charset        IN VARCHAR2 DEFAULT utl_http.body_charset)
RETURN VARCHAR2;
```

### Parameters

**Table 238–3 ESCAPE Function Parameters**

Parameter	Description
url	The original URL
escape_reserved_chars	Indicates whether the URL reserved characters should be escaped. If set to TRUE, both the reserved and illegal URL characters are escaped. Otherwise, only the illegal URL characters are escaped. The default value is FALSE.
url_charset	When escaping a character (single-byte or multibyte), determine the target character set that character should be converted to before the character is escaped in %hex-code format. If url_charset is NULL, the database charset is assumed and no character set conversion will occur. The default value is the current default body character set of the UTL_HTTP package, whose default value is ISO-8859-1. The character set can be named in Internet Assigned Numbers Authority (IANA) or in the Oracle naming convention.

### Usage Notes

Use this function to escape URLs that contain illegal characters as defined in the URL specification RFC 2396. The legal characters in URLs are:

- A through Z, a through z, and 0 through 9
- Hyphen (-), underscore (\_), period (.), exclamation point (!), tilde (~), asterisk (\*), accent ('), left parenthesis ( ( ), right parenthesis ( ) )

The reserved characters consist of:

- Semi-colon (;) slash (/), question mark (?), colon (:), at sign (@), ampersand (&), equals sign (=), plus sign (+), dollar sign (\$), and comma (,)

Many of the reserved characters are used as delimiters in the URL. You should escape characters beyond those listed here by using escape\_url. Also, to use the reserved characters in the name-value pairs of the query string of a URL, those characters must be escaped separately. An escape\_url cannot recognize the need to escape those characters because once inside a URL, those characters become indistinguishable from the actual delimiters. For example, to pass a name-value pair \$logon=scott/tiger into the query string of a URL, escape the \$ and / separately as %24logon=scott%2Ftiger and use it in the URL.

Normally, you will escape the entire URL, which contains the reserved characters (delimiters) that should not be escaped. For example:

```
utl_url.escape('http://www.acme.com/a url with space.html')
```

**Returns:**

```
http://foo.com/a%20url%20with%20space.html
```

In other situations, you may want to send a query string with a value that contains reserved characters. In that case, escape only the value fully (with `escape_reserved_chars` set to `TRUE`) and then concatenate it with the rest of the URL. For example:

```
url := 'http://www.acme.com/search?check=' || utl_url.escape  
('Is the use of the "$" sign okay?', TRUE);
```

This expression escapes the question mark (?), dollar sign (\$), and space characters in 'Is the use of the "\$" sign okay?' but not the ? after search in the URL that denotes the use of a query string.

The Web server that you intend to fetch Web pages from may use a character set that is different from that of your database. In that case, specify the `url_charset` as the Web server character set so that the characters that need to be escaped are escaped in the target character set. For example, a user of an EBCDIC database who wants to access an ASCII Web server should escape the URL using `US7ASCII` so that a space is escaped as `%20` (hex code of a space in ASCII) instead of `%40` (hex code of a space in EBCDIC).

This function does not validate a URL for the proper URL format.

## UNESCAPE Function

This function unescapes the escape character sequences to its original form in a URL, to convert the %XX escape character sequences to the original characters.

### Syntax

```
UTL_URL.UNESCAPE (
  url          IN VARCHAR2 CHARACTER SET ANY_CS,
  url_charset  IN VARCHAR2 DEFAULT utl_http.body_charset)
RETURN VARCHAR2;
```

### Parameters

**Table 238–4 UNESCAPE Function Parameters**

Parameter	Description
url	The URL to unescape
url_charset	After a character is unescaped, the character is assumed to be in the source_charset character set and it will be converted from the source_charset to the database character set before the URL is returned. If source_charset is NULL, the database charset is assumed and no character set conversion occurred. The default value is the current default body character set of the UTL_HTTP package, whose default value is "ISO-8859-1". The character set can be named in Internet Assigned Numbers Authority (IANA) or Oracle naming convention.

### Usage Notes

The Web server that you receive the URL from may use a character set that is different from that of your database. In that case, specify the url\_charset as the Web server character set so that the characters that need to be unescaped are unescaped in the source character set. For example, a user of an EBCDIC database who receives a URL from an ASCII Web server should unescape the URL using US7ASCII so that %20 is unescaped as a space (0x20 is the hex code of a space in ASCII) instead of a ? (because 0x20 is not a valid character in EBCDIC).

This function does not validate a URL for the proper URL format.



---

---

## WPG\_DOCLOAD

The WPG\_DOCLOAD package provides an interface to download files, BLOBs and BFILES.

**See Also:** For more information about implementation of this package:

- *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*
- *Oracle Fusion Middleware User's Guide for mod\_plsql*

The chapter contains the following topics:

- [Using WPG\\_DOCLOAD](#)
  - Constants
- [Summary of WPG\\_DOCLOAD Subprograms](#)

## Using WPG\_DOCLOAD

- [Constants](#)



## Constants

- [NAME\\_COL\\_LEN](#)
- [MIMET\\_COL\\_LEN](#)
- [MAX\\_DOCTABLE\\_NAME\\_LEN](#)

### NAME\_COL\_LEN

The `NAME` column in your document table must be the same as the value of `name_col_len`.

```
name_col_len CONSTANT pls_integer := 64;
```

### MIMET\_COL\_LEN

The `MIME_TYPE` column in your document table must be the same as the value of `mimet_col_len`.

```
mimet_col_len CONSTANT pls_integer := 48;
```

### MAX\_DOCTABLE\_NAME\_LEN

The name length of your document table must be less than `max_doctable_name_len`.

```
max_doctable_name_len CONSTANT pls_integer := 256;
```

## Summary of WPG\_DOCLOAD Subprograms

**Table 239–1 WPG\_DOCLOAD Package Subprograms**

Subprogram	Description
<a href="#">DOWNLOAD_FILE Procedures</a> on page 239-5	Downloads files, BLOBS and BFILES

## DOWNLOAD\_FILE Procedures

There are three versions of this procedure:

- The first version downloads files and is invoked from within a document download procedure to signal the PL/SQL Gateway that `p_filename` is to be downloaded from the document table to the client's browser.
- The second version can be called from within any procedure to signal the PL/SQL Gateway that `p_blob` is to be downloaded to the client's browser.
- The third version can be called from within any procedure to signal the PL/SQL Gateway that `p_bfile` is to be downloaded to the client's browser.

### Syntax

```
WPG_DOCLOAD.DOWNLOAD_FILE(
  p_filename      IN          VARCHAR2,
  p_bcaching      IN          BOOLEAN DEFAULT TRUE);
```

```
WPG_DOCLOAD.DOWNLOAD_FILE(
  p_blob          IN OUT NOCOPY BLOB);
```

```
WPG_DOCLOAD.DOWNLOAD_FILE(
  p_bfile         IN OUT          BFILE);
```

### Parameters

**Table 239–2** *DOWNLOAD\_FILE Procedure Parameters*

Parameter	Description
<code>p_filename</code>	The file to download from the document table.
<code>p_blob</code>	The BLOB to download.
<code>p_bfile</code>	The BFILE to download (see <a href="#">Usage Notes</a> ).
<code>p_bcaching</code>	Whether browser caching is enabled (see <a href="#">Usage Notes</a> ).

### Usage Notes

- Normally, a document will be downloaded to the browser unless the browser sends an 'If-Modified-Since' header to the gateway indicating that it has the requested document in its cache. In that case, the gateway will determine if the browser's cached copy is up to date, and if it is, it will send an HTTP 304 status message to the browser indicating that the browser should display the cached copy. However, because a document URL and a document do not necessarily have a one-to-one relationship in the PL/SQL Web Gateway, in some cases it may be undesirable to have the cached copy of a document displayed. In those cases, the `p_bcaching` parameter should be set to `FALSE` to indicate to the gateway to ignore the 'If-Modified-Since' header, and download the document.
- `p_bfile` and `p_blob` are declared as `IN OUT` because the locator is initially opened to check for file accessibility and existence. The open operation can only be performed if the locator is writable and readable.



---

---

## ANYDATA TYPE

An `ANYDATA TYPE` contains an instance of a given type, plus a description of the type. In this sense, an `ANYDATA` is self-describing. An `ANYDATA` can be persistently stored in the database.

This chapter contains the following topics:

- [Using ANYDATA TYPE](#)
  - Restrictions
  - Operational Notes
- [Summary of ANYDATA Subprograms](#)

## Using ANYDATA TYPE

- [Restrictions](#)
- [Operational Notes](#)

## Restrictions

Persistent storage of ANYDATA instances whose type contains embedded LOBs other than BFILES is not currently supported.

## Operational Notes

- [Construction](#)
- [Access](#)

### Construction

There are 2 ways to construct an ANYDATA. The CONVERT\* calls enable construction of the ANYDATA in its entirety with a single call. They serve as explicit CAST functions from any type in the Oracle ORDBMS to ANYDATA.

```
STATIC FUNCTION ConvertBDouble(dbl IN BINARY_DOUBLE) return ANYDATA,  
STATIC FUNCTION ConvertBfile(b IN BFILE) RETURN ANYDATA,  
STATIC FUNCTION ConvertBFloat(fl IN BINARY_FLOAT) return ANYDATA,  
STATIC FUNCTION ConvertBlob(b IN BLOB) RETURN ANYDATA,  
STATIC FUNCTION ConvertChar(c IN CHAR) RETURN ANYDATA,  
STATIC FUNCTION ConvertClob(c IN CLOB) RETURN ANYDATA,  
STATIC FUNCTION ConvertCollection(col IN "collection_type") RETURN ANYDATA,  
STATIC FUNCTION ConvertDate(dat IN DATE) RETURN ANYDATA,  
STATIC FUNCTION ConvertIntervalDS(inv IN INTERVAL DAY TO SECOND) return ANYDATA,  
STATIC FUNCTION ConvertIntervalYM(inv IN INTERVAL YEAR TO MONTH) return ANYDATA,  
STATIC FUNCTION ConvertNchar(nc IN NCHAR) return ANYDATA,  
STATIC FUNCTION ConvertNClob(nc IN NCLOB) return ANYDATA,  
STATIC FUNCTION ConvertNumber(num IN NUMBER) RETURN ANYDATA,  
STATIC FUNCTION ConvertNVarchar2(nc IN NVARCHAR2) return ANYDATA,  
STATIC FUNCTION ConvertObject(obj IN "<object_type>") RETURN ANYDATA,  
STATIC FUNCTION ConvertRaw(r IN RAW) RETURN ANYDATA,  
STATIC FUNCTION ConvertRef(rf IN REF "<object_type>") RETURN ANYDATA,  
STATIC FUNCTION ConvertTimestamp(ts IN TIMESTAMP) return ANYDATA,  
STATIC FUNCTION ConvertTimestampTZ(ts IN TIMESTAMP WITH TIMEZONE) return ANYDATA,  
STATIC FUNCTION ConvertTimestampLTZ(ts IN TIMESTAMP WITH LOCAL TIMEZONE) return  
ANYDATA,  
STATIC FUNCTION ConvertURowid(rid IN UROWID) return ANYDATA,  
STATIC FUNCTION ConvertVarchar(c IN VARCHAR) RETURN ANYDATA,  
STATIC FUNCTION ConvertVarchar2(c IN VARCHAR2) RETURN ANYDATA,
```

The second way to construct an ANYDATA is a piece by piece approach. The [BEGINCREATE Static Procedure](#) call begins the construction process and [ENDCREATE Member Procedure](#) call finishes the construction process. In between these two calls, the individual attributes of an object type or the elements of a collection can be set using SET\* calls. For piece by piece access of the attributes of objects and elements of collections, the [PIECEWISE Member Procedure](#) should be invoked prior to GET\* calls.

Note: The ANYDATA has to be constructed or accessed sequentially starting from its first attribute (or collection element). The BEGINCREATE call automatically begins the construction in a piece-wise mode. There is no need to call PIECEWISE immediately after BEGINCREATE. ENDCREATE should be called to finish the construction process (before which any access calls can be made).

### Access

Access functions are available based on SQL. These functions do not throw exceptions on type-mismatch. Instead, they return NULL if the type of the ANYDATA does not correspond to the type of access. If you wish to use only ANYDATA functions of the appropriate types returned in a query, you should use a WHERE clause which uses GETTYPENAME and choose the type you are interested in (say "SYS.NUMBER"). Each of



these functions returns the value of a specified datatype inside a SYS . ANYDATA wrapper.

```
MEMBER FUNCTION AccessBDouble(self IN ANYDATA) return BINARY_DOUBLE
    DETERMINISTIC,
MEMBER FUNCTION AccessBfile(self IN ANYDATA) return BFILE,
MEMBER FUNCTION AccessBFloat(self IN ANYDATA) return BINARY_FLOAT
    DETERMINISTIC,
MEMBER FUNCTION AccessBlob(self IN ANYDATA) return BLOB,
MEMBER FUNCTION AccessChar(self IN ANYDATA) return CHAR,
MEMBER FUNCTION AccessClob(self IN ANYDATA) return CLOB,
MEMBER FUNCTION AccessDate(self IN ANYDATA) return DATE,
MEMBER FUNCTION AccessIntervalYM(self IN ANYDATA) return INTERVAL YEAR TO MONTH,
MEMBER FUNCTION AccessIntervalDS(self IN ANYDATA) return INTERVAL DAY TO SECOND,
MEMBER FUNCTION AccessNchar(self IN ANYDATA) return NCHAR,
MEMBER FUNCTION AccessNClob(self IN ANYDATA) return NCLOB
MEMBER FUNCTION AccessNumber(self IN ANYDATA) return NUMBER,
MEMBER FUNCTION AccessNVarchar2(self IN ANYDATA) return NVARCHAR2,
MEMBER FUNCTION AccessRaw(self IN ANYDATA) return RAW,
MEMBER FUNCTION AccessTimestamp(self IN ANYDATA) return TIMESTAMP,
MEMBER FUNCTION AccessTimestampLTZ(self IN ANYDATA) return TIMESTAMP WITH LOCAL
    TIMEZONE,
MEMBER FUNCTION AccessTimestampTZ(self IN ANYDATA) return TIMESTAMP WITH
    TIMEZONE,
MEMBER FUNCTION AccessUrowid(self IN ANYDATA) return UROWID DETERMINISTIC
MEMBER FUNCTION AccessVarchar(self IN ANYDATA) return VARCHAR,
MEMBER FUNCTION AccessVarchar2(self IN ANYDATA) return VARCHAR2,
```

## Summary of ANYDATA Subprograms

**Table 240–1** ANYDATA Type Subprograms

Subprogram	Description
<a href="#">BEGINCREATE</a> Static Procedure on page 240-7	Begins creation process on a new ANYDATA
<a href="#">ENDCREATE</a> Member Procedure on page 240-8	Ends creation of an ANYDATA
<a href="#">GET*</a> Member Functions on page 240-9	Gets the current data value (which should be of appropriate type)
<a href="#">GETTYPE</a> Member Function on page 240-12	Gets the Type of the ANYDATA
<a href="#">GETTYPENAME</a> Member Function on page 240-13	Get the fully qualified type name for the ANYDATA
<a href="#">PIECEWISE</a> Member Procedure on page 240-14	Sets the MODE of access of the current data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT)
<a href="#">SET*</a> Member Procedures on page 240-15	Sets the current data value.

## BEGINCREATE Static Procedure

This procedure begins the creation process on a new ANYDATA.

### Syntax

```
STATIC PROCEDURE BeginCreate(
  dtype          IN OUT NOCOPY AnyType,
  adata          OUT NOCOPY ANYDATA);
```

### Parameters

**Table 240–2 BEGINCREATE Procedure Parameters**

Parameter	Description
dtype	The type of the ANYDATA. (Should correspond to OCI_TYPECODE_OBJECT or a Collection typecode.)
adata	ANYDATA being constructed.

### Exception

DBMS\_TYPES.INVALID\_PARAMETERS: dtype is invalid (not fully constructed, and similar deficits.)

### Usage Notes

There is no need to call `PIECEWISE` immediately after this call. The construction process begins in a piece-wise manner automatically.

## ENDCREATE Member Procedure

This procedure ends creation of an ANYDATA. Other creation functions cannot be called after this call.

### Syntax

```
MEMBER PROCEDURE EndCreate(  
    self          IN OUT NOCOPY ANYDATA);
```

### Parameters

**Table 240–3** *ENDCREATE Procedure Parameter*

Parameter	Description
self	An ANYDATA.

## GET\* Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the MODE with which we are accessing (depending on whether we have invoked the `PIECEWISE` call).

If `PIECEWISE` has NOT been called, we are accessing the `ANYDATA` in its entirety and the type of the data value should match the type of the `ANYDATA`.

If `PIECEWISE` has been called, we are accessing the `ANYDATA` piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

### Syntax

```
MEMBER FUNCTION GetBDouble(
    self      IN ANYDATA,
    dbl       OUT NOCOPY BINARY_DOUBLE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBfile(
    self      IN ANYDATA,
    b         OUT NOCOPY BFILE)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GetBFloat(
    self      IN ANYDATA,
    fl       OUT NOCOPY BINARY_FLOAT)
RETURN PLS_INTEGER;

MEMBER FUNCTION GetBlob(
    self      IN ANYDATA,
    b         OUT NOCOPY BLOB)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GetChar(
    self      IN ANYDATA,
    c         OUT NOCOPY CHAR)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GetClob(
    self      IN ANYDATA,
    c         OUT NOCOPY CLOB)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GetCollection(
    self      IN ANYDATA,
    col       OUT NOCOPY "<collection_type>")
RETURN      PLS_INTEGER;

MEMBER FUNCTION GetDate(
    self      IN ANYDATA,
    dat       OUT NOCOPY DATE)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GetIntervals(
    self      IN ANYDATA,
    inv       OUT NOCOPY INTERVAL DAY TO SECOND)
RETURN      PLS_INTEGER;
```

```
MEMBER FUNCTION GetIntervalYM(  
    self          IN ANYDATA,  
    inv          OUT NOCOPY INTERVAL YEAR TO MONTH)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetNchar(  
    self          IN ANYDATA,  
    nc           OUT NOCOPY NCHAR)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetNClob(  
    self          IN ANYDATA,  
    nc           OUT NOCOPY NCLOB)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetNumber(  
    self          IN ANYDATA,  
    num          OUT NOCOPY NUMBER)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetNvarchar2(  
    self          IN ANYDATA,  
    nc           OUT NOCOPY NVARCHAR2)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetObject(  
    self          IN ANYDATA,  
    obj          OUT NOCOPY "<object_type>")  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetRaw(  
    self          IN ANYDATA,  
    r            OUT NOCOPY RAW)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetRef(  
    self          IN ANYDATA,  
    rf           OUT NOCOPY REF "<object_type>")  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GetTimestamp(  
    self          IN ANYDATA,  
    ts           OUT NOCOPY TIMESTAMP)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GetTimestampTZ(  
    self          IN ANYDATA,  
    ts           OUT NOCOPY TIMESTAMP WITH TIME ZONE)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GetTimestampLTZ(  
    self          IN ANYDATA,  
    ts           OUT NOCOPY TIMESTAMP WITH LOCAL TIME ZONE)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GetVarchar(  
    self          IN ANYDATA,  
    c            OUT NOCOPY VARCHAR)  
RETURN          PLS_INTEGER;
```

```
MEMBER FUNCTION GetVarchar2(
  self      IN ANYDATA,
  c         OUT NOCOPY VARCHAR2)
RETURN     PLS_INTEGER;
```

## Parameters

**Table 240–4** *GET\* Function Parameter*

Parameter	Description
self	An ANYDATA.
num	The number to be obtained.

## Return Values

DBMS\_TYPES.SUCCESS or DBMS\_TYPES.NO\_DATA

The return value is relevant only if `PIECEWISE` has been already called (for a collection). In such a case, `DBMS_TYPES.NO_DATA` signifies the end of the collection when all elements have been accessed.

## Exceptions

`DBMS_TYPES.TYPE_MISMATCH`: When the expected type is different from the passed in type.

`DBMS_TYPES.INVALID_PARAMETERS`: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

`DBMS_TYPES.INCORRECT_USAGE`: Incorrect usage.

## GETTYPE Member Function

This function gets the typecode of the ANYDATA.

### Syntax

```
MEMBER FUNCTION GETTYPE (  
    self          IN ANYDATA,  
    typ           OUT NOCOPY AnyType)  
RETURN          PLS_INTEGER;
```

### Parameters

**Table 240–5** *GETTYPE Function Parameter*

Parameter	Description
self	An ANYDATA.
typ	The AnyType corresponding to the ANYDATA. May be NULL if it does not represent a user-defined type.

### Return Values

The typecode corresponding to the type of the ANYDATA.



## GETYPENAME Member Function

This function gets the fully qualified type name for the ANYDATA.

If the ANYDATA is based on a built-in type, this function will return NUMBER and other relevant information.

If it is based on a user defined type, this function will return *schema\_name.type\_name*, for example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

### Syntax

```
MEMBER FUNCTION GETYPENAME (  
    self          IN ANYDATA)  
RETURN          VARCHAR2;
```

### Parameters

**Table 240–6** GETYPENAME Function Parameter

Parameter	Description
self	An ANYDATA.

### Return Values

Type name of the ANYDATA.

## PIECEWISE Member Procedure

This procedure sets the **MODE** of access of the current data value to be an attribute at a time (if the data value is of `TYPECODE_OBJECT`).

It sets the **MODE** of access of the data value to be a collection element at a time (if the data value is of collection type). Once this call has been made, subsequent calls to `SET*` and `GET*` will sequentially obtain individual attributes or collection elements.

### Syntax

```
MEMBER PROCEDURE PIECEWISE(  
    self          IN OUT NOCOPY ANYDATA);
```

### Parameters

**Table 240–7** *PIECEWISE Procedure Parameters*

Parameter	Description
<code>self</code>	The current data value.

### Exceptions

- `DBMS_TYPES.INVALID_PARAMETERS`
- `DBMS_TYPES.INCORRECT_USAGE`: On incorrect usage.

### Usage Notes

The current data value must be of an `OBJECT` or `COLLECTION` type before this call can be made.

Piece-wise construction and access of nested attributes that are of object or collection types is not supported.

## SET\* Member Procedures

Sets the current data value.

This is a list of procedures that should be called depending on the type of the current data value. The type of the data value should be the type of the attribute at the current position during the piece-wise construction process.

### Syntax

```
MEMBER PROCEDURE SETBDOUBLE(
    self      IN OUT NOCOPY ANYDATA,
    dbl       IN BINARY_DOUBLE,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBFILE(
    self      IN OUT NOCOPY ANYDATA,
    b         IN BFILE,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBFLOAT(
    self      IN OUT NOCOPY ANYDATA,
    fl       IN          BINARY_FLOAT,
    last_elem IN          boolean DEFAULT FALSE);

MEMBER PROCEDURE SETBLOB(
    self      IN OUT NOCOPY ANYDATA,
    b         IN BLOB,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCHAR(
    self      IN OUT NOCOPY ANYDATA,
    c         IN CHAR,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCLOB(
    self      IN OUT NOCOPY ANYDATA,
    c         IN CLOB,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETCOLLECTION(
    self      IN OUT NOCOPY ANYDATA,
    col       IN "<collection_type>",
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETDATE(
    self      IN OUT NOCOPY ANYDATA,
    dat       IN DATE,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALDS(
    self      IN OUT NOCOPY ANYDATA,
    inv       IN INTERVAL DAY TO SECOND,
    last_elem IN boolean DEFAULT FALSE);

MEMBER PROCEDURE SETINTERVALYM(
    self      IN OUT NOCOPY ANYDATA,
    inv       IN INTERVAL YEAR TO MONTH,
    last_elem IN boolean DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETNCHAR(  
    self          IN OUT NOCOPY ANYDATA,  
    nc            IN NCHAR,  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETNCLOB(  
    self          IN OUT NOCOPY ANYDATA,  
    nc            IN NClob,  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETNUMBER(  
    self          IN OUT NOCOPY ANYDATA,  
    num          IN NUMBER,  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETNVARCHAR2(  
    self          IN OUT NOCOPY ANYDATA,  
    nc            IN NVarchar2,  
    last_elem    IN boolean DEFAULT FALSE),  
  
MEMBER PROCEDURE SETOBJECT(  
    self          IN OUT NOCOPY ANYDATA,  
    obj          IN "<object_type>",  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETRAW(  
    self          IN OUT NOCOPY ANYDATA,  
    r            IN RAW,  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETREF(  
    self          IN OUT NOCOPY ANYDATA,  
    rf           IN REF "<object_type>",  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETTIMESTAMP(  
    self          IN OUT NOCOPY ANYDATA,  
    ts           IN TIMESTAMP,  
    last_elem    IN BOOLEAN DEFAULT FALSE);  
  
MEMBER PROCEDURE SETTIMESTAMPPTZ(self IN OUT NOCOPY ANYDATA,  
    ts           IN TIMESTAMP WITH TIME ZONE,  
    last_elem    IN BOOLEAN DEFAULT FALSE);  
  
MEMBER PROCEDURE SETTIMESTAMPPLTZ(  
    self IN OUT NOCOPY ANYDATA,  
    ts IN TIMESTAMP WITH LOCAL TIME ZONE,  
    last_elem IN boolean DEFAULT FALSE),  
  
MEMBER PROCEDURE SETVARCHAR(  
    self          IN OUT NOCOPY ANYDATA,  
    c            IN VARCHAR,  
    last_elem    IN boolean DEFAULT FALSE);  
  
MEMBER PROCEDURE SETVARCHAR2(  
    self          IN OUT NOCOPY ANYDATA,  
    c            IN VARCHAR2,  
    last_elem    IN boolean DEFAULT FALSE);
```

## Parameters

**Table 240–8 SET\* Procedure Parameters**

Parameter	Description
<code>self</code>	An ANYDATA.
<code>num</code>	The number, and associated information, that is to be set.
<code>last_elem</code>	Relevant only if ANYDATA represents a collection. Set to <code>TRUE</code> if it is the last element of the collection, <code>FALSE</code> otherwise.

## Exceptions

- `DBMS_TYPES.INVALID_PARAMETERS`: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).
- `DBMS_TYPES.INCORRECT_USAGE`: Incorrect usage.
- `DBMS_TYPES.TYPE_MISMATCH`: When the expected type is different from the passed in type.

## Usage Notes

When `BEGINCREATE` is called, construction has already begun in a piece-wise fashion. Subsequent calls to `SET*` will set the successive attribute values.

If the ANYDATA is a standalone collection, the `SET*` call will set the successive collection elements.



---

---

## ANYDATASET TYPE

An ANYDATASET TYPE contains a description of a given type plus a set of data instances of that type. An ANYDATASET can be persistently stored in the database if desired, or it can be used as interface parameters to communicate self-descriptive sets of data, all of which belong to a certain type.

This chapter contains the following topics:

- [Construction](#)
- [Summary of ANYDATASET TYPE Subprograms](#)

## Construction

---

The ANYDATASET needs to be constructed value by value, sequentially.

For each data instance (of the type of the ANYDATASET), the ADDINSTANCE function must be invoked. This adds a new data instance to the ANYDATASET. Subsequently, SET\* can be called to set each value in its entirety.

The MODE of construction/access can be changed to attribute/collection element wise by making calls to PIECEWISE.

- If the type of the ANYDATASET is TYPECODE\_OBJECT, individual attributes will be set with subsequent SET\* calls. Likewise on access.
- If the type of the current data value is a collection type individual collection elements will be set with subsequent SET\* calls. Likewise on access. This call is very similar to ANYDATA.PIECEWISE call defined for the type ANYDATA.

Note that there is no support for piece-wise construction and access of nested (not top level) attributes that are of object types or collection types.

ENDCREATE should be called to finish the construction process (before which no access calls can be made).



---

## Summary of ANYDATASET TYPE Subprograms

**Table 241–1 ANYDATASET Type Subprograms**

Subprogram	Description
<a href="#">ADDINSTANCE Member Procedure</a> on page 241-4	Adds a new data instance to an ANYDATASET.
<a href="#">BEGINCREATE Static Procedure</a> on page 241-5	Creates a new ANYDATASET which can be used to create a set of data values of the given ANYTYPE.
<a href="#">ENDCREATE Member Procedure</a> on page 241-6	Ends Creation of a ANYDATASET. Other creation functions cannot be called after this call.
<a href="#">GET* Member Functions</a> on page 241-7	Gets the current data value (which should be of appropriate type).
<a href="#">GETCOUNT Member Function</a> on page 241-10	Gets the number of data instances in an ANYDATASET.
<a href="#">GETINSTANCE Member Function</a> on page 241-11	Gets the next instance in an ANYDATASET.
<a href="#">GETTYPE Member Function</a> on page 241-12	Gets the ANYTYPE describing the type of the data instances in an ANYDATASET. current data value (which should be of appropriate type).
<a href="#">GETTYPENAME Member Function</a> on page 241-13	Gets the AnyType describing the type of the data instances in an ANYDATASET.
<a href="#">PIECEWISE Member Procedure</a> on page 241-14	Sets the MODE of construction, access of the data value to be an attribute at a time (if the data value is of TYPECODE_OBJECT).
<a href="#">SET* Member Procedures</a> on page 241-15	Sets the current data value.

## ADDINSTANCE Member Procedure

This procedure adds a new data instance to an ANYDATASET.

### Syntax

```
MEMBER PROCEDURE AddInstance(  
    self          IN OUT NOCOPY ANYDATASET);
```

### Parameters

**Table 241–2** ADDINSTANCE Procedure Parameter

Parameter	Description
self	The ANYDATASET being constructed.

### Exceptions

DBMS\_TYPES.invalid\_parameters: Invalid parameters.  
DBMS\_TYPES.incorrect\_usage: On incorrect usage.

### Usage Notes

The data instances have to be added sequentially. The previous data instance must be fully constructed (or set to NULL) before a new one can be added.

This call DOES NOT automatically set the mode of construction to be piece-wise. The user has to explicitly call PIECEWISE if a piece-wise construction of the instance is intended.

## BEGINCREATE Static Procedure

This procedure creates a new ANYDATASET which can be used to create a set of data values of the given ANYTYPE .

### Syntax

```
STATIC PROCEDURE BeginCreate(
  typecode      IN PLS_INTEGER,
  rtype         IN OUT NOCOPY AnyType,
  aset          OUT NOCOPY ANYDATASET);
```

### Parameters

**Table 241–3 BEGINCREATE Procedure Parameter**

Parameter	Description
typecode	The typecode for the type of the ANYDATASET.
dtype	The type of the data values. This parameter is a must for user-defined types like TYPECODE_OBJECT, Collection typecodes, and similar others.
aset	The ANYDATASET being constructed.

### Exceptions

DBMS\_TYPES.invalid\_parameters: dtype is invalid (not fully constructed, and like errors.)

## ENDCREATE Member Procedure

This procedure ends Creation of a `ANYDATASET`. Other creation functions cannot be called after this call.

### Syntax

```
MEMBER PROCEDURE ENDCREATE(  
    self          IN OUT NOCOPY ANYDATASET);
```

### Parameters

**Table 241–4** *ENDCREATE Procedure Parameter*

Parameter	Description
<code>self</code>	The <code>ANYDATASET</code> being constructed.

## GET\* Member Functions

These functions get the current data value (which should be of appropriate type).

The type of the current data value depends on the `MODE` with which you are accessing it (depending on how we have invoked the `PIECEWISE` call). If `PIECEWISE` has not been called, we are accessing the instance in its entirety and the type of the data value should match the type of the `ANYDATASET`.

If `PIECEWISE` has been called, we are accessing the instance piece-wise. The type of the data value should match the type of the attribute (or collection element) at the current position.

### Syntax

```
MEMBER FUNCTION GETBDOUBLE(
    self      IN ANYDATASET,
    dbl       OUT NOCOPY BINARY_DOUBLE)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETBFLOAT(
    self      IN ANYDATASET,
    fl        OUT NOCOPY BINARY_FLOAT)
RETURN PLS_INTEGER;

MEMBER FUNCTION GETBFILE(
    self      IN ANYDATASET,
    b         OUT NOCOPY BFILE)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GETBLOB(
    self      IN ANYDATASET,
    b         OUT NOCOPY BLOB)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GETCHAR(
    self      IN ANYDATASET,
    c         OUT NOCOPY CHAR)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GETCLOB(
    self      IN ANYDATASET,
    c         OUT NOCOPY CLOB)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GETCOLLECTION(
    self      IN ANYDATASET,
    col       OUT NOCOPY "<collection_type>")
RETURN      PLS_INTEGER;

MEMBER FUNCTION GETDATE(
    self      IN ANYDATASET,
    dat       OUT NOCOPY DATE)
RETURN      PLS_INTEGER;

MEMBER FUNCTION GETINTERVALDS(
    self      IN ANYDATASET,
    inv       IN OUT NOCOPY INTERVAL DAY TO SECOND)
RETURN PLS_INTEGER;
```

```
MEMBER FUNCTION GETINTERVALYM(  
    self          IN ANYDATASET,  
    inv IN OUT NOCOPY INTERVAL YEAR TO MONTH)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GETNCHAR(  
    self          IN ANYDATASET,  
    nc           OUT NOCOPY NCHAR)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GETNCLOB(  
    self          IN ANYDATASET,  
    nc           OUT NOCOPY NCLOB)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GETNUMBER(  
    self          IN ANYDATASET,  
    num          OUT NOCOPY NUMBER)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GETNVARCHAR2(  
    self          IN ANYDATASET,  
    nc           OUT NOCOPY NVARCHAR2)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GETOBJECT(  
    self          IN ANYDATASET,  
    obj          OUT NOCOPY "<object_type>")  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GETRAW(  
    self          IN ANYDATASET,  
    r            OUT NOCOPY RAW)  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GETREF(  
    self          IN ANYDATASET,  
    rf           OUT NOCOPY REF "<object_type>")  
RETURN          PLS_INTEGER;  
  
MEMBER FUNCTION GETTIMESTAMP(  
    self          IN ANYDATASET,  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GETTIMESTAMPPLTZ(  
    self          IN ANYDATASET,  
    ts           OUT NOCOPY TIMESTAMP WITH LOCAL TIME ZONE)  
RETURN PLS_INTEGER;  
  
MEMBER FUNCTION GETTIMESTAMPPTZ(  
    self          IN ANYDATASET,  
    ts           OUT NOCOPY TIMESTAMP WITH TIME ZONE)  
RETURN PLS_INTEGER,  
  
MEMBER FUNCTION GETUROWID(  
    self          IN ANYDATASET,  
    rid          OUT NOCOPY UROWID)  
RETURN PLS_INTEGER
```

```
MEMBER FUNCTION GETVARCHAR (
  self      IN ANYDATASET,
  c         OUT NOCOPY VARCHAR)
RETURN     PLS_INTEGER;
```

```
MEMBER FUNCTION GETVARCHAR2 (
  self      IN ANYDATASET,
  c         OUT NOCOPY VARCHAR2)
RETURN     PLS_INTEGER;
```

## Parameters

**Table 241–5** *GET\* Function Parameters*

Parameter	Description
self	The ANYDATASET being accessed.
num	The number, and associated information., that is to be obtained.

## Return Values

DBMS\_TYPES . SUCCESS or DBMS\_TYPES . NO\_DATA

The return value is relevant only if `PIECEWISE` has been already called (for a collection). In such a case, `DBMS_TYPES.NO_DATA` signifies the end of the collection when all elements have been accessed.

## Exceptions

DBMS\_TYPES . INVALID\_PARAMETERS: Invalid Parameters (if it is not appropriate to add a number at this point in the creation process).

DBMS\_TYPES . INCORRECT\_USAGE: Incorrect usage

DBMS\_TYPES . TYPE\_MISMATCH: When the expected type is different from the passed in type.

## GETCOUNT Member Function

This function gets the number of data instances in an ANYDATASET.

### Syntax

```
MEMBER FUNCTION GetCount (  
    self          IN ANYDATASET)  
    RETURN        PLS_INTEGER;
```

### Parameter

**Table 241–6** *GETCOUNT Function Parameter*

Parameter	Description
self	The ANYDATASET being accessed.

### Return Values

The number of data instances.



## GETINSTANCE Member Function

This function gets the next instance in an ANYDATASET. Only sequential access to the instances in an ANYDATASET is allowed. After this function has been called, the GET\* functions can be invoked on the ANYDATASET to access the current instance. If PIECEWISE is called before doing the GET\* calls, the individual attributes (or collection elements) can be accessed.

It is an error to invoke this function before the ANYDATASET is fully created.

### Syntax

```
MEMBER FUNCTION GETINSTANCE(  
    self          IN OUT NOCOPY ANYDATASET)  
    RETURN        PLS_INTEGER;
```

### Parameters

**Table 241–7** *GETINSTANCE Function Parameter*

Parameter	Description
self	The ANYDATASET being accessed.

### Return Values

DBMS\_TYPES.SUCCESS or DBMS\_TYPES.NO\_DATA

DBMS\_TYPES.NO\_DATA signifies the end of the ANYDATASET (all instances have been accessed).

### Usage Notes

This function should be called even before accessing the first instance.

## GETTYPE Member Function

Gets the AnyType describing the type of the data instances in an ANYDATASET.

### Syntax

```
MEMBER FUNCTION GETTYPE (  
    self          IN ANYDATASET,  
    typ           OUT NOCOPY AnyType)  
RETURN          PLS_INTEGER;
```

### Parameters

**Table 241–8** *GETTYPE Function Parameter*

Parameter	Description
self	The ANYDATASET.
typ	The ANYTYPE corresponding to the AnyData. May be NULL if it does not represent a user-defined function.

### Return Values

The typecode corresponding to the type of the ANYDATA.

## GETYPENAME Member Function

This procedure gets the fully qualified type name for the ANYDATASET.

If the ANYDATASET is based on a built-in, this function will return NUMBER and associated information.

If it is based on a user defined type, this function will return *schema\_name.type\_name*. for example, SCOTT.FOO.

If it is based on a transient anonymous type, this function will return NULL.

### Syntax

```
MEMBER FUNCTION GETYPENAME(  
    self          IN ANYDATASET)  
    RETURN        VARCHAR2;
```

### Parameter

**Table 241–9** GETYPENAME Function Parameter

Parameter	Description
self	The ANYDATASET being constructed.

### Return Values

Type name of the ANYDATASET.

## PIECEWISE Member Procedure

This procedure sets the `MODE` of construction, access of the data value to be an attribute at a time (if the data value is of `TYPECODE_OBJECT`).

It sets the `MODE` of construction, access of the data value to be a collection element at a time (if the data value is of a collection `TYPE`). Once this call has been made, subsequent `SET*` and `GET*` calls will sequentially obtain individual attributes or collection elements.

### Syntax

```
MEMBER PROCEDURE PIECEWISE(  
    self          IN OUT NOCOPY ANYDATASET);
```

### Parameters

**Table 241–10** *PIECEWISE Procedure Parameter*

Parameter	Description
<code>self</code>	The ANYDATASET being constructed.

### Exceptions

`DBMS_TYPES.INVALID_PARAMETERS`: Invalid parameters.

`DBMS_TYPES.INCORRECT_USAGE`: On incorrect usage.

### Usage Notes

The current data value must be of an object or collection type before this call can be made. There is no support for piece-wise construction or access of embedded object type attributes or nested collections.

## SET\* Member Procedures

This procedure sets the current data value.

The type of the current data value depends on the `MODE` with which we are constructing (depending on how we have invoked the `PIECEWISE` call). The type of the current data should be the type of the `ANYDATASET` if `PIECEWISE` has NOT been called. The type should be the type of the attribute at the current position if `PIECEWISE` has been called.

### Syntax

```
MEMBER PROCEDURE SETBDOUBLE(
  self          IN OUT NOCOPY ANYDATASET,
  dbl           IN BINARY_DOUBLE,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETBFLOAT(
  self          IN OUT NOCOPY ANYDATASET,
  fl           IN BINARY_FLOAT,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETBFILE(
  self          IN OUT NOCOPY ANYDATASET,
  b            IN BFILE,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETBLOB(
  self          IN OUT NOCOPY ANYDATASET,
  b            IN BLOB,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETCHAR(
  self          IN OUT NOCOPY ANYDATASET,
  c            IN CHAR,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETCLOB(
  self          IN OUT NOCOPY ANYDATASET,
  c            IN CLOB,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETCOLLECTION(
  self          IN OUT NOCOPY ANYDATASET,
  col          IN "<collection_type>",
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETDATE(
  self          IN OUT NOCOPY ANYDATASET,
  dat          IN DATE,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETINTERVALDS(
  self          IN OUT NOCOPY ANYDATASET,
  inv          IN INTERVAL DAY TO SECOND,
  last_elem     IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETINTERVALYM(
  self          IN OUT NOCOPY ANYDATASET,
```

```
        inv          IN INTERVAL YEAR TO MONTH,
        last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNCHAR(
    self          IN OUT NOCOPY ANYDATASET,
    nc            IN NCHAR,
    last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNCLOB(

s
    self          IN OUT NOCOPY ANYDATASET,
    nc            IN NClob,
    last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNUMBER(
    self          IN OUT NOCOPY ANYDATASET,
    num          IN NUMBER,
    last_elem    BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETNVARCHAR2(
    self          IN OUT NOCOPY ANYDATASET,
    nc            IN NVarchar2,
    last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETOBJECT(
    self          IN OUT NOCOPY ANYDATASET,
    obj          IN "<object_type>",
    last_elem    BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETRAW(
    self          IN OUT NOCOPY ANYDATASET,
    r            IN RAW,
    last_elem    BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETREF(
    self          IN OUT NOCOPY ANYDATASET,
    rf          IN REF "<object_type>",
    last_elem    BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMP(
    self          IN OUT NOCOPY ANYDATASET,
    ts          IN TIMESTAMP,
    last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPPLTZ(
    self          IN OUT NOCOPY ANYDATASET,
    ts          IN TIMESTAMP WITH LOCAL TIME ZONE,
    last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETTIMESTAMPPTZ(
    self          IN OUT NOCOPY ANYDATASET,
    ts          IN TIMESTAMP WITH TIME ZONE,
    last_elem    IN BOOLEAN DEFAULT FALSE);

MEMBER PROCEDURE SETUROWID(
    self          IN OUT NOCOPY ANYDATASET,
    rid          IN UROWID,
    last_elem    IN BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETVARCHAR(
  self          IN OUT NOCOPY ANYDATASET,
  c             IN VARCHAR,
  last_elem    BOOLEAN DEFAULT FALSE);
```

```
MEMBER PROCEDURE SETVARCHAR2(
  self          IN OUT NOCOPY ANYDATASET,
  c             IN VARCHAR2,
  last_elem    BOOLEAN DEFAULT FALSE);
```

## Parameters

**Table 241–11** *SET\* Procedure Parameters*

Parameter	Description
self	The ANYDATASET being accessed.
num	The number, and associated information, that is to be set.
last_elem	Relevant only if <code>PIECEWISE</code> has been already called (for a collection). Set to <code>TRUE</code> if it is the last element of the collection, <code>FALSE</code> otherwise.

## Exceptions

- `DBMS_TYPES.INVALID_PARAMETERS`: Invalid parameters (if it is not appropriate to add a number at this point in the creation process).
- `DBMS_TYPES.INCORRECT_USAGE`: Incorrect usage.
- `DBMS_TYPES.TYPE_MISMATCH`: When the expected type is different from the passed in type.





---

---

## ANYTYPE TYPE

An `ANYTYPE TYPE` can contain a type description of any persistent SQL type, named or unnamed, including object types and collection types. It can also be used to construct new transient type descriptions.

New persistent types can only be created using the `CREATE TYPE` statement. Only new transient types can be constructed using the `ANYTYPE` interfaces.

This chapter discusses the following:

- [Summary of ANYTYPE Subprograms](#)

---

## Summary of ANYTYPE Subprograms

**Table 242–1 ANYTYPE Type Subprograms**

Subprogram	Description
<a href="#">BEGINCREATE Static Procedure</a> on page 242-3	Creates a new instance of ANYTYPE which can be used to create a transient type description.
<a href="#">SETINFO Member Procedure</a> on page 242-4	Sets any additional information required for constructing a COLLECTION or builtin type.
<a href="#">ADDATTR Member Procedure</a> on page 242-6	Adds an attribute to an ANYTYPE (of typecode DBMS_TYPES.TYPECODE_OBJECT).
<a href="#">ENDCREATE Member Procedure</a> on page 242-7	Ends creation of a transient ANYTYPE. Other creation functions cannot be called after this call.
<a href="#">GETPERSISTENT Static Function</a> on page 242-8	Returns an ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.
<a href="#">GETINFO Member Function</a> on page 242-9	Gets the type information for the ANYTYPE.
<a href="#">GETATTRELEMINFO Member Function</a> on page 242-10	Gets the type information for an attribute of the type (if it is of TYPECODE_OBJECT). Gets the type information for a collection's element type if the <i>self</i> parameter is of a collection type.

## BEGINCREATE Static Procedure

This procedure creates a new instance of ANYTYPE which can be used to create a transient type description.

### Syntax

```
STATIC PROCEDURE BEGINCREATE (
  typecode      IN          PLS_INTEGER,
  atype         OUT NOCOPY ANYTYPE);
```

### Parameters

**Table 242–2** *BEGINCREATE Procedure Parameters*

Parameter	Description
typecode	Use a constant from DBMS_TYPES package. Typecodes for user-defined type: <ul style="list-style-type: none"> <li>▪ DBMS_TYPES.TYPECODE_OBJECT</li> <li>▪ DBMS_TYPES.TYPECODE_VARRAY or</li> <li>▪ DBMS_TYPES.TYPECODE_TABLE</li> </ul> Typecodes for builtin types: <ul style="list-style-type: none"> <li>▪ DBMS_TYPES.TYPECODE_NUMBER, and similar types.</li> </ul>
atype	ANYTYPE for a transient type

## SETINFO Member Procedure

This procedure sets any additional information required for constructing a `COLLECTION` or builtin type.

### Syntax

```
MEMBER PROCEDURE SETINFO (
  self          IN OUT NOCOPY ANYTYPE,
  prec          IN PLS_INTEGER,
  scale        IN PLS_INTEGER,
  len          IN PLS_INTEGER,
  csid         IN PLS_INTEGER,
  csfrm        IN PLS_INTEGER,
  atype        IN ANYTYPE DEFAULT NULL,
  elem_tc      IN PLS_INTEGER DEFAULT NULL,
  elem_count   IN PLS_INTEGER DEFAULT 0);
```

### Parameters

**Table 242–3 SETINFO Procedure Parameters**

Parameter	Description
<code>self</code>	The transient <code>ANYTYPE</code> that is being constructed.
<code>prec</code>	Optional. Required if typecode represents a <code>NUMBER</code> . Give precision and scale. Ignored otherwise.
<code>scale</code>	Optional. Required if typecode represents a <code>NUMBER</code> . Give precision and scale. Ignored otherwise.
<code>len</code>	Optional. Required if typecode represents a <code>RAW</code> , <code>CHAR</code> , <code>VARCHAR</code> , or <code>VARCHAR2</code> type. Gives length.
<code>csid</code>	Required if typecode represents types requiring character information such as <code>CHAR</code> , <code>VARCHAR</code> , or <code>VARCHAR2</code> .
<code>csfrm</code>	Required if typecode represents types requiring character information such as <code>CHAR</code> , <code>VARCHAR</code> , or <code>VARCHAR2</code> .
<code>atype</code>	Optional. Required if collection element typecode is a user-defined type such as <code>TYPECODE_OBJECT</code> , and similar others. It is also required for a built-in type that needs user-defined type information such as <code>TYPECODE_REF</code> . This parameter is not needed otherwise.

The Following Parameters Are Required For Collection Types

**Table 242–4 SETINFO Procedure Parameters - Collection Types**

Parameter	Description
<code>elem_tc</code>	Must be of the collection element's typecode (from <code>DBMS_TYPES</code> package).
<code>elem_count</code>	Pass 0 for <code>elem_count</code> if the <code>self</code> represents a nested table ( <code>TYPECODE_TABLE</code> ). Otherwise pass the collection count if <code>self</code> represents a <code>VARRAY</code> .

### Exceptions

- `DBMS_TYPES.INVALID_PARAMETER`: Invalid Parameters (typecode, typeinfo)

- `DBMS_TYPES.INCORRECT_USAGE`: Incorrect usage (cannot call after calling `ENDCREATE`, and similar actions.)

### Usage Notes

It is an error to call this function on an `ANYTYPE` that represents a persistent user defined type.

## ADDATTR Member Procedure

This procedure adds an attribute to an ANYTYPE (of typecode DBMS\_TYPES.TYPECODE\_OBJECT).

### Syntax

```
MEMBER PROCEDURE ADDATTR (
    self          IN OUT NOCOPY ANYTYPE,
    aname         IN VARCHAR2,
    typecode      IN PLS_INTEGER,
    prec          IN PLS_INTEGER,
    scale         IN PLS_INTEGER,
    len           IN PLS_INTEGER,
    csid          IN PLS_INTEGER,
    csfrm         IN PLS_INTEGER,
    attr_type     IN ANYTYPE DEFAULT NULL);
```

### Parameters

**Table 242–5 ADDATTR Procedure Parameters**

Parameter	Description
self	The transient ANYTYPE that is being constructed. Must be of type DBMS_TYPES.TYPECODE_OBJECT.
aname	Optional. Attribute's name. Could be NULL.
typecode	Attribute's typecode. Can be built-in or user-defined typecode (from DBMS_TYPES package).
prec	Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
scale	Optional. Required if typecode represents a NUMBER. Give precision and scale. Ignored otherwise.
len	Optional. Required if typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Give length.
csid	Optional. Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, or VARCHAR2.
csfrm	Optional. Required if typecode represents a type requiring character information, such as CHAR, VARCHAR, or VARCHAR2.
attr_type	Optional. ANYTYPE corresponding to a user-defined type. This parameter is required if the attribute is a user defined type.

### Exceptions

- DBMS\_TYPES.INVALID\_PARAMETERS: Invalid Parameters (typecode, typeinfo)
- DBMS\_TYPES.INCORRECT\_USAGE: Incorrect usage (cannot call after calling EndCreate, and similar actions.)

## ENDCREATE Member Procedure

This procedure ends creation of a transient ANYTYPE. Other creation functions cannot be called after this call.

### Syntax

```
MEMBER PROCEDURE ENDCREATE(  
    self          IN OUT NOCOPY ANYTYPE);
```

### Parameter

**Table 242–6** *ENDCREATE Procedure Parameter*

Parameter	Description
self	The transient ANYTYPE that is being constructed.

## GETPERSISTENT Static Function

This procedure returns an ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.

### Syntax

```
STATIC FUNCTION GETPERSISTENT(  
    schema_name    IN VARCHAR2,  
    type_name      IN VARCHAR2,  
    version        IN VARCHAR2 DEFAULT NULL)  
RETURN            ANYTYPE;
```

### Parameters

**Table 242–7** GETPERSISTENT Function Parameters

Parameter	Description
schema_name	Schema name of the type.
type_name	Type name.
version	Type version.

### Return Values

An ANYTYPE corresponding to a persistent type created earlier using the CREATE TYPE SQL statement.



## GETINFO Member Function

This function gets the type information for the ANYTYPE.

### Syntax

```
MEMBER FUNCTION GETINFO (
  self      IN ANYTYPE,
  prec      OUT PLS_INTEGER,
  scale     OUT PLS_INTEGER,
  len       OUT PLS_INTEGER,
  csid      OUT PLS_INTEGER,
  csfrm     OUT PLS_INTEGER,
  schema_name OUT VARCHAR2,
  type_name OUT VARCHAR2,
  version   OUT varchar2,
  count     OUT PLS_INTEGER)
RETURN     PLS_INTEGER;
```

### Parameters

**Table 242–8** GETINFO Function Parameters

Parameter	Description
self	The ANYTYPE.
prec	If typecode represents a number. Gives precision and scale. Ignored otherwise.
scale	If typecode represents a number. Gives precision and scale. Ignored otherwise.
len	If typecode represents a RAW, CHAR, VARCHAR, or VARCHAR2 type. Gives length.
csid	If typecode represents a type requiring character information such as: CHAR, VARCHAR, or VARCHAR2.
csid	If typecode represents a type requiring character information such as: CHAR, VARCHAR, or VARCHAR2.
schema_name	Type's schema (if persistent).
type_name	Type's typename.
version	Type's version.
count	If <i>self</i> is a VARRAY, this gives the VARRAY count. If <i>self</i> is of TYPECODE_OBJECT, this gives the number of attributes.

### Return Values

The typecode of *self*.

### Exceptions

- DBMS\_TYPES.INVALID\_PARAMETERS: Invalid Parameters (position is beyond bounds or the ANYTYPE is not properly Constructed).

## GETATTRELEMINFO Member Function

This function gets the type information for an attribute of the type (if it is of `TYPECODE_OBJECT`). Gets the type information for a collection's element type if the *self* parameter is of a collection type.

### Syntax

```
MEMBER FUNCTION GETATTRELEMINFO (
    self          IN ANYTYPE,
    pos           IN PLS_INTEGER,
    prec         OUT PLS_INTEGER,
    scale        OUT PLS_INTEGER,
    len          OUT PLS_INTEGER,
    csid         OUT PLS_INTEGER,
    csfrm        OUT PLS_INTEGER,
    attr_elt_type OUT ANYTYPE
    aname        OUT VARRCHAR2)
RETURN         PLS_INTEGER;
```

### Parameters

**Table 242–9** GETATTRELEMINFO Function Parameters

Parameter	Description
<code>self</code>	The <code>ANYTYPE</code> .
<code>pos</code>	If <code>self</code> is of <code>TYPECODE_OBJECT</code> , this gives the attribute position (starting at 1). It is ignored otherwise.
<code>prec</code>	If attribute/collection element typecode represents a <code>NUMBER</code> . Gives precision and scale. Ignored otherwise.
<code>scale</code>	If attribute/collection element typecode represents a <code>NUMBER</code> . Gives precision and scale. Ignored otherwise.
<code>len</code>	If typecode represents a <code>RAW</code> , <code>CHAR</code> , <code>VARCHAR</code> , or <code>VARCHAR2</code> type. Gives length.
<code>csid</code> , <code>csfrm</code>	If typecode represents a type requiring character information such as: <code>CHAR</code> , <code>VARCHAR</code> , or <code>VARCHAR2</code> . Gives character set ID, character set form.
<code>attr_elt_type</code>	If attribute/collection element typecode represents a user-defined type, this returns the <code>ANYTYPE</code> corresponding to it. User can subsequently describe the <i>attr_elt_type</i> .
<code>aname</code>	Attribute name (if it is an attribute of an object type, <code>NULL</code> otherwise).

### Return Values

The typecode of the attribute or collection element.

### Exceptions

`DBMS_TYPES.INVALID_PARAMETERS`: Invalid Parameters (position is beyond bounds or the `ANYTYPE` is not properly constructed).

---

---

## Oracle Streams AQ TYPEs

This chapter describes the types used with Oracle Streams Advanced Queuing (AQ) packages for PL/SQL, `DBMS_AQ`, and `DBMS_AQADM`.

**See Also:** *Oracle Streams Advanced Queuing User's Guide* for information about using Oracle Streams AQ.

This chapter contains the following topics:

- [Summary of Types](#)

---

## Summary of Types

- [AQ\\$\\_AGENT](#) Type
- [AQ\\$\\_AGENT\\_LIST\\_T](#) Type
- [AQ\\$\\_DESCRIPTOR](#) Type
- [AQ\\$\\_NTFN\\_DESCRIPTOR](#) Type
- [AQ\\$\\_POST\\_INFO](#) Type
- [AQ\\$\\_POST\\_INFO\\_LIST](#) Type
- [AQ\\$\\_PURGE\\_OPTIONS\\_T](#) Type
- [AQ\\$\\_RECIPIENT\\_LIST\\_T](#) Type
- [AQ\\$\\_REG\\_INFO](#) Type
- [AQ\\$\\_REG\\_INFO\\_LIST](#) Type
- [AQ\\$\\_SUBSCRIBER\\_LIST\\_T](#) Type
- [DEQUEUE\\_OPTIONS\\_T](#) Type
- [ENQUEUE\\_OPTIONS\\_T](#) Type
- [SYS.MSG\\_PROP\\_T](#) Type
- [MESSAGE\\_PROPERTIES\\_T](#) Type
- [MESSAGE\\_PROPERTIES\\_ARRAY\\_T](#) Type
- [MSGID\\_ARRAY\\_T](#) Type

## AQ\$\_AGENT Type

This type identifies a producer or a consumer of a message.

### Syntax

```
TYPE SYS.AQ$_AGENT IS OBJECT (
  name      VARCHAR2(30),
  address   VARCHAR2(1024),
  protocol  NUMBER DEFAULT 0);
```

### Attributes

**Table 243–1 AQ\$\_AGENT Attributes**

Attribute	Description
name	Name of a producer or consumer of a message. The name must follow object name guidelines in the <i>Oracle Database SQL Language Reference</i> with regard to reserved characters.
address	Protocol-specific address of the recipient. If the protocol is 0, then the address is of the form <code>[schema.]queue[@dblink]</code> . For example, a queue named <code>emp_messages</code> in the HR queue at the site <code>db1.net</code> has the address: <code>hr.emp_messages@db1.net</code>
protocol	Protocol to interpret the address and propagate the message. Protocols 1-127 are reserved for internal use. If the protocol number is in the range 128 - 255, the address of the recipient is not interpreted by Oracle Streams AQ.

## AQ\$\_AGENT\_LIST\_T Type

This type identifies the list of agents for which DBMS\_AQ.LISTEN listens.

**See Also:** ["AQ\\$\\_AGENT Type"](#) on page 243-3

### Syntax

```
TYPE SYS.AQ$_AGENT_LIST_T IS TABLE OF SYS.AQ$_AGENT  
INDEX BY BINARY_INTEGER;
```

## AQ\$\_DESCRIPTOR Type

This type specifies the Oracle Streams AQ descriptor received by the AQ PL/SQL callbacks upon notification.

**See Also:** ["MESSAGE\\_PROPERTIES\\_T Type"](#) on page 243-24

### Syntax

```
TYPE SYS.AQ$_DESCRIPTOR IS OBJECT (
  queue_name      VARCHAR2(61),
  consumer_name   VARCHAR2(30),
  msg_id          RAW(16),
  msg_prop        MSG_PROP_T,
  gen_desc        AQ$_NTFN_DESCRIPTOR,
  msgid_array     SYS.AQ$_NTFN_MSGID_ARRAY,
  ntfnRecdInGrp  NUMBER);
```

### Attributes

**Table 243–2 AQ\$\_DESCRIPTOR Attributes**

Attribute	Description
queue_name	Name of the queue in which the message was enqueued which resulted in the notification
consumer_name	Name of the consumer for the multiconsumer queue
msg_id	Identification number of the message
msg_prop	Message properties specified by the MSG_PROP_T type
gen_desc	Indicates the timeout specifications
msgid_array	Group notification message ID list
ntfnRecdInGrp	Notifications received in group

## AQ\$\_NTFN\_DESCRIPTOR Type

This type is for storing a generic notification descriptor regarding PL/SQL notification flags.

### Syntax

```
TYPE SYS.AQ$_NTFN_DESCRIPTOR IS OBJECT(  
    ntfn_flags NUMBER)
```

### Attributes

**Table 243–3 AQ\$\_NTFN\_DESCRIPTOR Attributes**

Attribute	Description
ntfn_flags	Set to 1 if the notifications are already removed after a stipulated timeout. Set 2 to denote grouping. Default is 0.



## AQ\$\_NTFN\_MSGID\_ARRAY Type

This type is for storing grouping notification data for AQ namespace, value  $2^{30}$  which is the max varray size.

### Syntax

```
TYPE SYS.AQ$_NTFN_MSGID_ARRAY  
AS VARRAY(1073741824) OF RAW(16);
```

## AQ\$\_POST\_INFO Type

Specifies anonymous subscriptions to which you want to post messages.

### Syntax

```
TYPE SYS.AQ$_POST_INFO IS OBJECT (  
    name          VARCHAR2(128),  
    namespace     NUMBER,  
    payload       RAW(2000) DEFAULT NULL);
```

### Attributes

**Table 243–4 AQ\$\_POST\_INFO Attributes**

Attribute	Description
name	Name of the anonymous subscription to which you want to post
namespace	To receive notifications from other applications through <code>DBMS_AQ.POST</code> or <code>OCISubscriptionPost()</code> , the namespace must be <code>DBMS_AQ.NAMESPACE_ANONYMOUS</code>
payload	The payload to be posted to the anonymous subscription

## AQ\$\_POST\_INFO\_LIST Type

Identifies the list of anonymous subscriptions to which you want to post messages.

**See Also:** [AQ\\$\\_POST\\_INFO Type](#) on page 243-8

### Syntax

```
TYPE SYS.AQ$_POST_INFO_LIST AS VARRAY(1024) OF SYS.AQ$_POST_INFO;
```

## AQ\$\_PURGE\_OPTIONS\_T Type

This type specifies the options available for purging a queue table.

**See Also:** [PURGE\\_QUEUE\\_TABLE Procedure](#) on page 23-42.

### Syntax

```
TYPE AQ$_PURGE_OPTIONS_T IS RECORD (
    block          BOOLEAN          DEFAULT FALSE
    delivery_mode  PLS_INTEGER     DEFAULT PERSISTENT);
```

**Table 243–5** AQ\$\_PURGE\_OPTIONS\_T Type Attributes

Attribute	Description
block	<p>TRUE/FALSE.</p> <ul style="list-style-type: none"> <li>▪ If <code>block</code> is <code>TRUE</code>, then an exclusive lock on all the queues in the queue table is held while purging the queue table. This will cause concurrent enqueueers and dequeuers to block while the queue table is purged. The purge call always succeeds if <code>block</code> is <code>TRUE</code>.</li> <li>▪ The default for <code>block</code> is <code>FALSE</code>. This will not block enqueueers and dequeuers, but it can cause the purge to fail with an error during high concurrency times.</li> </ul>
delivery_mode	Kind of messages to purge, either <code>DBMS_AQ.BUFFERED</code> or <code>DBMS_AQ.PERSISTENT</code>

## AQ\$\_RECIPIENT\_LIST\_T Type

Identifies the list of agents that receive the message. This type can be used only when the queue is enabled for multiple dequeues.

**See Also:** ["AQ\\$\\_AGENT Type"](#) on page 243-3

### Syntax

```
TYPE SYS.AQ$_RECIPIENT_LIST_T IS TABLE OF SYS.AQ$_AGENT  
INDEX BY BINARY_INTEGER;
```

## AQ\$\_REG\_INFO Type

This type identifies a producer or a consumer of a message.

### Syntax

```
TYPE SYS.AQ$_REG_INFO IS OBJECT (
    name                VARCHAR2(128),
    namespace           NUMBER,
    callback            VARCHAR2(4000),
    context             RAW(2000) DEFAULT NULL,
    qosflags           NUMBER,
    timeout            NUMBER,
    ntfn_grouping_class NUMBER,
    ntfn_grouping_value NUMBER DEFAULT 600,
    ntfn_grouping_type NUMBER,
    ntfn_grouping_start_time  TIMESTAMP WITH TIME ZONE,
    ntfn_grouping_repeat_count NUMBER);
```

### Attributes

**Table 243–6 AQ\$\_REG\_INFO Type Attributes**

Attribute	Description
name	Specifies the name of the subscription. The subscription name is of the form <i>schema.queue</i> if the registration is for a single consumer queue or <i>schema.queue:consumer_name</i> if the registration is for a multiconsumer queues.
namespace	Specifies the namespace of the subscription. To receive notification from Oracle Streams AQ queues, the namespace must be <code>DBMS_AQ.NAMESPACE_AQ</code> . To receive notifications from other applications through <code>DBMS_AQ.POST</code> or <code>OCISubscriptionPost()</code> , the namespace must be <code>DBMS_AQ.NAMESPACE_ANONYMOUS</code> .
callback	Specifies the action to be performed on message notification. For HTTP notifications, use <code>http://www.company.com:8080</code> . For e-mail notifications, use <code>mailto://xyz@company.com</code> . For raw message payload for the <code>PLSQLCALLBACK</code> procedure, use <code>plsql://schema.procedure?PR=0</code> . For user-defined type message payload converted to XML for the <code>PLSQLCALLBACK</code> procedure, use <code>plsql://schema.procedure?PR=1</code> .
context	Specifies the context that is to be passed to the callback function
qosflags	Can be set to one or more of the following values to specify the notification quality of service: <ul style="list-style-type: none"> <li>▪ <code>NTFN_QOS_RELIABLE</code>- This value specifies that reliable notification is required. Reliable notifications persist across instance and database restarts.</li> <li>▪ <code>NTFN_QOS_PAYLOAD</code> - This value specifies that payload delivery is required. It is supported only for client notification and only for RAW queues.</li> <li>▪ <code>NTFN_QOS_PURGE_ON_NTFN</code> - This value specifies that the registration is to be purged automatically when the first notification is delivered to this registration location.</li> </ul>

**Table 243–6 (Cont.) AQ\$\_REG\_INFO Type Attributes**

Attribute	Description
<code>ntfn_grouping_class</code>	Currently, only the following flag can be set to specify criterion for grouping. The default value will be 0. If <code>ntfn_grouping_class</code> is 0, all other notification grouping attributes must be 0. <ul style="list-style-type: none"> <li>NTFN_GROUPING_CLASS_TIME - Notifications grouped by time, that is, the user specifies a time value and a single notification gets published at the end of that time.</li> </ul>
<code>ntfn_grouping_value</code>	Time-period of grouping notifications specified in seconds, meaning the time after which grouping notification would be sent periodically until <code>ntfn_grouping_repeat_count</code> is exhausted.
<code>ntfn_grouping_type</code>	<ul style="list-style-type: none"> <li>NTFN_GROUPING_TYPE_SUMMARY - Summary of all notifications that occurred in the time interval. (Default)</li> <li>NTFN_GROUPING_TYPE_LAST - Last notification that occurred in the interval.</li> </ul>
<code>ntfn_grouping_start_time</code>	Notification grouping start time. Notification grouping can start from a user-specified time that should be a valid timestamp with time zone. If <code>ntfn_grouping_start_time</code> is not specified when using grouping, the default is to current timestamp with time zone
<code>ntfn_grouping_repeat_count</code>	Grouping notifications will be sent as many times as specified by the notification grouping repeat count and after that revert to regular notifications. The <code>ntfn_grouping_repeat_count</code> , if not specified, will default to <ul style="list-style-type: none"> <li>NTFN_GROUPING_FOREVER - Keep sending grouping notifications forever.</li> </ul>

## Usage Notes

You can use the following notification mechanisms:

- OCI callback
- e-mail callback
- PL/SQL callback

[Table 243–7](#) shows the actions performed for nonpersistent queues for different notification mechanisms when RAW presentation is specified. [Table 243–8](#) shows the actions performed when XML presentation is specified.

**Table 243–7 Actions Performed for Nonpersistent Queues When RAW Presentation Specified**

Queue Payload Type	OCI Callback	E-mail	PL/SQL Callback
RAW	OCI callback receives the RAW data in the payload.	Not supported	PL/SQL callback receives the RAW data in the payload.
Oracle object type	Not supported	Not supported	Not supported

**Table 243–8 Actions Performed for Nonpersistent Queues When XML Presentation Specified**

<b>Queue Payload Type</b>	<b>OCI Callback</b>	<b>E-mail</b>	<b>PL/SQL Callback</b>
RAW	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.
Oracle object type	OCI callback receives the XML data in the payload.	XML data is formatted as a SOAP message and e-mailed to the registered e-mail address.	PL/SQL callback receives the XML data in the payload.



## AQ\$ REG\_INFO\_LIST Type

Identifies the list of registrations to a queue.

**See Also:** ["AQ\\$ REG\\_INFO Type"](#) on page 243-12

### Syntax

```
TYPE SYS.AQ$ REG_INFO_LIST AS VARRAY(1024) OF SYS.AQ$ REG_INFO;
```

## AQ\$\_SUBSCRIBER\_LIST\_T Type

Identifies the list of subscribers that subscribe to a queue.

**See Also:** ["AQ\\$\\_AGENT Type"](#) on page 243-3

### Syntax

```
TYPE SYS.AQ$_SUBSCRIBER_LIST_T IS TABLE OF SYS.AQ$_AGENT  
INDEX BY BINARY_INTEGER;
```

## DEQUEUE\_OPTIONS\_T Type

Specifies the options available for the dequeue operation.

### Syntax

```
TYPE DEQUEUE_OPTIONS_T IS RECORD (
  consumer_name    VARCHAR2(30)    DEFAULT NULL,
  dequeue_mode     BINARY_INTEGER  DEFAULT REMOVE,
  navigation       BINARY_INTEGER  DEFAULT NEXT_MESSAGE,
  visibility       BINARY_INTEGER  DEFAULT ON_COMMIT,
  wait             BINARY_INTEGER  DEFAULT FOREVER,
  msgid            RAW(16)         DEFAULT NULL,
  correlation      VARCHAR2(128)   DEFAULT NULL,
  deq_condition    VARCHAR2(4000)  DEFAULT NULL,
  signature        aq$_sig_prop    DEFAULT NULL,
  transformation   VARCHAR2(61)   DEFAULT NULL,
  delivery_mode    PLS_INTEGER     DEFAULT PERSISTENT);
```

### Attributes

**Table 243–9 DEQUEUE\_OPTIONS\_T Attributes**

Attribute	Description
consumer_name	Name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should be set to NULL.  For secure queues, consumer_name must be a valid AQ agent name, mapped to the database user performing the dequeue operation, through dbms_aqadm.enable_db_access procedure call.
dequeue_mode	Specifies the locking behavior associated with the dequeue. Possible settings are:  BROWSE: Read the message without acquiring any lock on the message. This specification is equivalent to a select statement.  LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This setting is equivalent to a select for update statement.  REMOVE: Read the message and delete it. This setting is the default. The message can be retained in the queue table based on the retention properties.  REMOVE_NODATA: Mark the message as updated or deleted. The message can be retained in the queue table based on the retention properties.

**Table 243–9 (Cont.) DEQUEUE\_OPTIONS\_T Attributes**

Attribute	Description
navigation	<p>Specifies the position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. Possible settings are:</p> <p><b>NEXT_MESSAGE:</b> Retrieve the next message that is available and matches the search criteria. If the previous message belongs to a message group, then AQ retrieves the next available message that matches the search criteria and belongs to the message group. This setting is the default.</p> <p><b>NEXT_TRANSACTION:</b> Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This setting can only be used if message grouping is enabled for the current queue.</p> <p><b>FIRST_MESSAGE:</b> Retrieves the first message which is available and matches the search criteria. This setting resets the position to the beginning of the queue.</p> <p><b>FIRST_MESSAGE_MULTI_GROUP:</b> indicates that a call to <code>DBMS_AQ.DEQUEUE_ARRAY</code> will reset the position to the beginning of the queue and dequeue messages (possibly across different transaction groups) that are available and match the search criteria, until reaching the <code>ARRAY_SIZE</code> limit. Refer to the <code>TRANSACTION_GROUP</code> attribute for the message to distinguish between transaction groups.</p> <p><b>NEXT_MESSAGE_MULTI_GROUP:</b> indicates that a call to <code>DBMS_AQ.DEQUEUE_ARRAY</code> will dequeue the next set of messages (possibly across different transaction groups) that are available and match the search criteria, until reaching the <code>ARRAY_SIZE</code> limit. Refer to the <code>TRANSACTION_GROUP</code> attribute for the message to distinguish between transaction groups.</p>
visibility	<p>Specifies whether the new message is dequeued as part of the current transaction. The visibility parameter is ignored when using the <code>BROWSE</code> dequeue mode. Possible settings are:</p> <p><b>ON_COMMIT:</b> The dequeue will be part of the current transaction. This setting is the default.</p> <p><b>IMMEDIATE:</b> The dequeue operation is not part of the current transaction, but an autonomous transaction which commits at the end of the operation.</p>
wait	<p>Specifies the wait time if there is currently no message available which matches the search criteria. Possible settings are:</p> <p><b>FOREVER:</b> Wait forever. This setting is the default.</p> <p><b>NO_WAIT:</b> Do not wait.</p> <p><b>number:</b> Wait time in seconds.</p>
msgid	<p>Specifies the message identifier of the message to be dequeued.</p>
correlation	<p>Specifies the correlation identifier of the message to be dequeued. Special pattern matching characters, such as the percent sign (%) and the underscore (_) can be used. If more than one message satisfies the pattern, then the order of dequeuing is undetermined.</p>

**Table 243–9 (Cont.) DEQUEUE\_OPTIONS\_T Attributes**

Attribute	Description
deq_condition	<p>A conditional expression based on the message properties, the message data properties, and PL/SQL functions.</p> <p>A <code>deq_condition</code> is specified as a Boolean expression using syntax similar to the <code>WHERE</code> clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the <code>WHERE</code> clause of a SQL query). Message properties include <code>priority</code>, <code>corrid</code> and other columns in the queue table.</p> <p>To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with <code>tab.user_data</code> as a qualifier to indicate the specific column of the queue table that stores the payload. The <code>deq_condition</code> parameter cannot exceed 4000 characters. If more than one message satisfies the dequeue condition, then the order of dequeuing is undetermined.</p>
signature	Currently not implemented
transformation	Specifies a transformation that will be applied after dequeuing the message. The source type of the transformation must match the type of the queue.
delivery_mode	The dequeuer specifies the delivery mode of the messages it wishes to dequeue in the dequeue options. It can be <code>BUFFERED</code> or <code>PERSISTENT</code> or <code>PERSISTENT_OR_BUFFERED</code> . The message properties of the dequeued message indicate the delivery mode of the dequeued message. Array dequeue is only supported for buffered messages with an array size of '1'.

## ENQUEUE\_OPTIONS\_T Type

Specifies the options available for the enqueue operation.

### Syntax

```
TYPE SYS.ENQUEUE_OPTIONS_T IS RECORD (
  visibility          BINARY_INTEGER  DEFAULT ON_COMMIT,
  relative_msgid     RAW(16)          DEFAULT NULL,
  sequence_deviation BINARY_INTEGER  DEFAULT NULL,
  transformation     VARCHAR2(61)    DEFAULT NULL,
  delivery_mode      PLS_INTEGER     NOT NULL DEFAULT PERSISTENT);
```

### Attributes

**Table 243–10 ENQUEUE\_OPTIONS\_T Attributes**

Attribute	Description
visibility	Specifies the transactional behavior of the enqueue request. Possible settings are:  ON_COMMIT: The enqueue is part of the current transaction. The operation is complete when the transaction commits. This setting is the default.  IMMEDIATE: The enqueue operation is not part of the current transaction, but an autonomous transaction which commits at the end of the operation. This is the only value allowed when enqueueing to a non-persistent queue.
relative_msgid	Specifies the message identifier of the message which is referenced in the sequence deviation operation. This field is valid only if BEFORE is specified in sequence_deviation. This parameter is ignored if sequence deviation is not specified.
sequence_deviation	Specifies whether the message being enqueued should be dequeued before other messages already in the queue. Possible settings are:  BEFORE: The message is enqueued ahead of the message specified by relative_msgid.  TOP: The message is enqueued ahead of any other messages.
transformation	Specifies a transformation that will be applied before enqueueing the message. The return type of the transformation function must match the type of the queue.
delivery_mode	The enqueuer specifies the delivery mode of the messages it wishes to enqueue in the enqueue options. It can be BUFFERED or PERSISTENT. The message properties of the enqueued message indicate the delivery mode of the enqueued message. Array enqueue is only supported for buffered messages with an array size of '1'.

## SYS.MSG\_PROP\_T Type

This type is used in PL/SQL notification, as one field in `aq$_descriptor`, to pass message properties of an AQ message to the PL/SQL notification client callback.

### Syntax

```
CREATE or replace TYPE sys.msg_prop_t AS OBJECT (
  priority          NUMBER,
  delay             NUMBER,
  expiration        NUMBER,
  correlation        VARCHAR2(128),
  attempts          NUMBER,
  exception_queue   VARCHAR2(51),
  enqueue_time      DATE,
  state             NUMBER,
  sender_id         aq$_agent,
  original_msgid    RAW(16),
  delivery_mode     NUMBER);
```

### Parameters

**Table 243–11** SYS.MSG\_PROP\_T Type Attributes

Parameter	Description
priority	Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
delay	Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by <code>msgid</code> overrides the delay specification. A message enqueued with <code>delay</code> set is in the <code>WAITING</code> state, and when the delay expires, the message goes to the <code>READY</code> state. <code>DELAY</code> processing requires the queue monitor to be started. However the queue monitor is started automatically by the system if needed. Delay is set by the producer who enqueues the message.  The possible settings follow: <code>NO_DELAY</code> : The message is available for immediate dequeuing number: The number of seconds to delay the message
expiration	Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the time the message is ready for dequeue. Expiration processing requires the queue monitor to be running. However the queue monitor is started automatically by the system if needed.  The possible settings follow: <code>NEVER</code> : The message does not expire number: The number of seconds message remains in <code>READY</code> state. If the message is not dequeued before it expires, then it is moved to the exception queue in the <code>EXPIRED</code> state.
correlation	Returns the identifier supplied by the producer of the message at enqueue time.
attempts	Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.

**Table 243–11 (Cont.) SYS.MSG\_PROP\_T Type Attributes**

Parameter	Description
exception_queue	<p>Specifies the name of the queue into which the message is moved if it cannot be processed successfully.</p> <p>Messages are moved automatically into the exception queue. Messages are moved into the exception queue in the following cases:</p> <ul style="list-style-type: none"> <li>■ <code>RETRY_COUNT</code>, the number of unsuccessful dequeue attempts, has exceeded the specification for the <code>MAX_RETRIES</code> parameter in the <code>DBMS_AQADM.CREATE_QUEUE</code> procedure during queue creation.</li> </ul> <p>For multiconsumer queues, the message becomes eligible to be moved to the exception queue even if failed dequeue attempts exceeds the <code>MAX_RETRIES</code> parameter for only one of the consumers. But the message will not be moved until either all other consumers have successfully consumed the message or failed more than <code>MAX_RETRIES</code>. You can view <code>MAX_RETRIES</code> for a queue in the <code>ALL_QUEUES</code> data dictionary view.</p> <p>If a dequeue transaction fails because the server process dies (including <code>ALTER SYSTEM KILL SESSION</code>) or <code>SHUTDOWN ABORT</code> on the instance, then <code>RETRY_COUNT</code> is not incremented.</p> <ul style="list-style-type: none"> <li>■ A message was not dequeued before the expiration time elapsed.</li> <li>■ Message propagation to the specified destination queue failed with one of the following errors: <ul style="list-style-type: none"> <li>* There were no recipients for the multiconsumer destination queue.</li> <li>* Recipients were specified for a single-consumer destination queue.</li> <li>* Destination queue was an exception queue</li> <li>* There was an error when applying transformation.</li> </ul> </li> </ul> <p>The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is specified, then the parameter returns a <code>NULL</code> value at dequeue time.</p>
enqueue_time	<p>Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.</p>
state	<p>Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow:</p> <ul style="list-style-type: none"> <li>■ <code>DBMS_AQ.READY</code>: The message is ready to be processed.</li> <li>■ <code>DBMS_AQ.WAITING</code>: The message delay has not yet been reached.</li> <li>■ <code>DBMS_AQ.PROCESSED</code>: The message has been processed and is retained.</li> <li>■ <code>DBMSAQ.EXPIRED</code>: The message has been moved to the exception queue.</li> </ul>



**Table 243–11 (Cont.) SYS.MSG\_PROP\_T Type Attributes**

<b>Parameter</b>	<b>Description</b>
sender_id	<p>The application-sender identification specified at enqueue time by the message producer. Sender id is of type aq\$_agent.</p> <p>Sender name is required for secure queues at enqueue time. This must be a valid AQ agent name, mapped to the database user performing the enqueue operation, through dbms_aqadm.enable_db_access procedure call. Sender address and protocol should not be specified.</p> <p>The Sender id in the message properties returned at dequeue time may have a sender address if the message was propagated from another queue. The value of the address is the source_queue, source database name if it was a remote database [format source_queue@source_database_name]</p>
original_msgid	This parameter is used by Oracle Streams AQ for propagating messages.
delivery_mode	DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT.

## MESSAGE\_PROPERTIES\_T Type

This type is defined inside the DBMS\_AQ package, and describes the information that AQ uses to convey the state of individual messages. These are set at enqueue time, and their values are returned at dequeue time.

**See Also:** [AQ\\$\\_RECIPIENT\\_LIST\\_T Type](#) on page 243-11

### Syntax

```

TYPE message_properties_t IS RECORD (
  priority          BINARY_INTEGER NOT NULL DEFAULT 1,
  delay            BINARY_INTEGER NOT NULL DEFAULT NO_DELAY,
  expiration       BINARY_INTEGER NOT NULL DEFAULT NEVER,
  correlation      VARCHAR2(128)  DEFAULT NULL,
  attempts         BINARY_INTEGER,
  recipient_list   AQ$_RECIPIENT_LIST_T,
  exception_queue  VARCHAR2(61)   DEFAULT NULL,
  enqueue_time     DATE,
  state            BINARY_INTEGER,
  sender_id        SYS.AQ$_AGENT  DEFAULT NULL,
  original_msgid   RAW(16)        DEFAULT NULL,
  signature        aq$_sig_prop   DEFAULT NULL,
  transaction_group VARCHAR2(30)  DEFAULT NULL,
  user_property    SYS.ANYDATA    DEFAULT NULL,
  delivery_mode    PLS_INTEGER    NOT NULL DEFAULT DBMS_AQ.PERSISTENT);

```

### Attributes

**Table 243–12** MESSAGE\_PROPERTIES\_T Attributes

Attribute	Description
priority	Specifies the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
delay	<p>Specifies the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by <code>msgid</code> overrides the delay specification. A message enqueued with delay set is in the <code>WAITING</code> state, and when the delay expires, the message goes to the <code>READY</code> state. <code>DELAY</code> processing requires the queue monitor to be started. However the queue monitor is started automatically by the system if needed. Delay is set by the producer who enqueues the message.</p> <p>The possible settings follow:</p> <p><code>NO_DELAY</code>: The message is available for immediate dequeuing</p> <p>number: The number of seconds to delay the message</p>

**Table 243–12 (Cont.) MESSAGE\_PROPERTIES\_T Attributes**

Attribute	Description
expiration	<p>Specifies the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the time the message is ready for dequeue. Expiration processing requires the queue monitor to be running. However the queue monitor is started automatically by the system if needed.</p> <p>The possible settings follow:</p> <p>NEVER: The message does not expire</p> <p>number: The number of seconds message remains in READY state. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state.</p>
correlation	Returns the identifier supplied by the producer of the message at enqueue time.
attempts	Returns the number of attempts that have been made to dequeue the message. This parameter cannot be set at enqueue time.
recipient_list	<p>This parameter is only valid for queues that allow multiple consumers. The default recipients are the queue subscribers. This parameter is not returned to a consumer at dequeue time.</p> <p>For type definition, see the "<a href="#">AQ\$_AGENT Type</a>" on page 243-3.</p>

**Table 243–12 (Cont.) MESSAGE\_PROPERTIES\_T Attributes**

Attribute	Description
exception_queue	<p data-bbox="626 260 1330 317">Specifies the name of the queue into which the message is moved if it cannot be processed successfully.</p> <p data-bbox="626 327 1330 405">Messages are moved automatically into the exception queue. Messages are moved into the exception queue in the following cases:</p> <ul data-bbox="626 422 1330 527" style="list-style-type: none"> <li data-bbox="626 422 1330 527">■ <code>RETRY_COUNT</code>, the number of unsuccessful dequeue attempts, has exceeded the specification for the <code>MAX_RETRIES</code> parameter in the <code>DBMS_AQADM.CREATE_QUEUE</code> procedure during queue creation.</li> </ul> <p data-bbox="675 537 1330 747">For multiconsumer queues, the message becomes eligible to be moved to the exception queue even if failed dequeue attempts exceeds the <code>MAX_RETRIES</code> parameter for only one of the consumers. But the message will not be moved until either all other consumers have successfully consumed the message or failed more than <code>MAX_RETRIES</code>. You can view <code>MAX_RETRIES</code> for a queue in the <code>ALL_QUEUES</code> data dictionary view.</p> <p data-bbox="675 758 1330 863">If a dequeue transaction fails because the server process dies (including <code>ALTER SYSTEM KILL SESSION</code>) or <code>SHUTDOWN ABORT</code> on the instance, then <code>RETRY_COUNT</code> is not incremented.</p> <ul data-bbox="626 884 1330 1209" style="list-style-type: none"> <li data-bbox="626 884 1330 940">■ A message was not dequeued before the expiration time elapsed.</li> <li data-bbox="626 951 1330 1209">■ Message propagation to the specified destination queue failed with one of the following errors: <ul data-bbox="675 1014 1330 1209" style="list-style-type: none"> <li data-bbox="675 1014 1330 1071">* There were no recipients for the multiconsumer destination queue.</li> <li data-bbox="675 1081 1330 1138">* Recipients were specified for a single-consumer destination queue.</li> <li data-bbox="675 1148 1330 1176">* Destination queue was an exception queue</li> <li data-bbox="675 1186 1330 1209">* There was an error when applying transformation.</li> </ul> </li> </ul> <p data-bbox="626 1230 1330 1388">The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert log. If the default exception queue is specified, then the parameter returns a <code>NULL</code> value at dequeue time.</p>
enqueue_time	<p data-bbox="626 1409 1330 1486">Specifies the time the message was enqueued. This value is determined by the system and cannot be set by the user at enqueue time.</p>
state	<p data-bbox="626 1507 1330 1585">Specifies the state of the message at the time of the dequeue. This parameter cannot be set at enqueue time. The possible states follow:</p> <ul data-bbox="626 1596 1330 1820" style="list-style-type: none"> <li data-bbox="626 1596 1330 1623">■ <code>DBMS_AQ.READY</code>: The message is ready to be processed.</li> <li data-bbox="626 1633 1330 1690">■ <code>DBMS_AQ.WAITING</code>: The message delay has not yet been reached.</li> <li data-bbox="626 1701 1330 1757">■ <code>DBMS_AQ.PROCESSED</code>: The message has been processed and is retained.</li> <li data-bbox="626 1768 1330 1820">■ <code>DBMSAQ.EXPIRED</code>: The message has been moved to the exception queue.</li> </ul>

**Table 243–12 (Cont.) MESSAGE\_PROPERTIES\_T Attributes**

Attribute	Description
sender_id	<p>The application-sender identification specified at enqueue time by the message producer. Sender id is of type aq\$_agent.</p> <p>Sender name is required for secure queues at enqueue time. This must be a valid AQ agent name, mapped to the database user performing the enqueue operation, through dbms_aqadm.enable_db_access procedure call. Sender address and protocol should not be specified.</p> <p>The Sender id in the message properties returned at dequeue time may have a sender address if the message was propagated from another queue. The value of the address is the source_queue, source database name if it was a remote database [format source_queue@source_database_name]</p>
original_msgid	This parameter is used by Oracle Streams AQ for propagating messages.
signature	Currently not implemented
transaction_group	<p>Specifies the transaction_group for the dequeued message. Messages belonging to the same transaction group will have the same value for this attribute. This attribute is only set by the DBMS_AQ.DEQUEUE_ARRAY. This attribute cannot be used to set the transaction group of a message through DBMS_AQ.ENQUEUE or DBMS_AQ.ENQUEUE_ARRAY calls.</p>
user_property	This optional attribute is used to store additional information about the payload.
delivery_mode	<p>The message publisher specifies the delivery mode in the message_properties. This can be DBMS_AQ.BUFFERED or DBMS_AQ.PERSISTENT. Array enqueue is only supported for buffered messages with an array size of '1'.</p>

## MESSAGE\_PROPERTIES\_ARRAY\_T Type

This type is used by `dbms_aq.enqueue_array` and `dbms_aq.dequeue_array` calls to hold the set of message properties. Each element in the `payload_array` should have a corresponding element in the `MESSAGE_PROPERTIES_ARRAY_T` VARRAY.

**See Also:** ["MESSAGE\\_PROPERTIES\\_T Type"](#) on page 243-24

### Syntax

```
TYPE MESSAGE_PROPERTIES_ARRAY_T IS VARRAY (2147483647)
  OF MESSAGE_PROPERTIES_T;
```

## MSGID\_ARRAY\_T Type

The `msgid_array_t` type is used in `dbms_aq.enqueue_array` and `dbms_aq.dequeue_array` calls to hold the set of message IDs that correspond to the enqueued or dequeued messages.

### Syntax

```
TYPE MSGID_ARRAY_T IS TABLE OF RAW(16) INDEX BY BINARY_INTEGER
```





---

---

## DBFS Content Interface Types

This chapter describes public types defined to support the [DBMS\\_DBFS\\_CONTENT](#) interface.

This chapter contains the following topics:

- [Using Content Types](#)
  - Overview
  - Security Model
- [Data Structures](#)

## Using Content Types

- [Overview](#)
- [Security Model](#)

## Overview

The type definitions described in this chapter support the [DBMS\\_DBFS\\_CONTENT](#) interface in implementing metadata tables, packages, views, dependent application-side entities, and service-provider entities.

## Security Model

The user can access the content operational and administrative interfaces (packages, types, tables, and so on) by way of the `DBFS_ROLE`. This role can be granted to users as needed.

## Data Structures

Types that support the [DBMS\\_DBFS\\_CONTENT](#) interface include both Object and Table types.

### Object Types

- [DBMS\\_DBFS\\_CONTENT\\_CONTEXT\\_T](#) Object Type
- [DBMS\\_DBFS\\_CONTENT\\_LIST\\_ITEM\\_T](#) Object Type
- [DBMS\\_DBFS\\_CONTENT\\_PROPERTY\\_T](#) Object Type

### Table Types

- [DBMS\\_DBFS\\_CONTENT\\_LIST\\_ITEMS\\_T](#) Table Type
- [DBMS\\_DBFS\\_CONTENT\\_PROPERTIES\\_T](#) Table Type
- [DBMS\\_DBFS\\_CONTENT\\_RAW\\_T](#) Table Type

## DBMS\_DBFS\_CONTENT\_CONTEXT\_T Object Type

This type describes the execution context for the providers. It provides the user performing the operation with the Access Control List, the owner of the item(s), a timestamp for doing `asof` queries, and whether or not the item(s) are `read_only`. This type can be used both as input, in the case of path item creation functions, and output, in the case of path item query, or both.

### Syntax

```
CREATE OR REPLACE TYPE dbms_dbfs_content_context_t
  AUTHID DEFINER
AS OBJECT (
  principal      VARCHAR2(32),
  acl            VARCHAR2(1024),
  owner         VARCHAR2(32),
  asof          TIMESTAMP,
  read_only     INTEGER);
```

### Fields

**Table 244–1** *DBMS\_DBFS\_CONTENT\_CONTEXT\_T Fields*

Field	Description
<code>principal</code>	File system user
<code>acl</code>	Access control list
<code>owner</code>	Path item owner
<code>asof</code>	Timestamp
<code>read_only</code>	Nonzero if the path item is read-only

## DBMS\_DBFS\_CONTENT\_LIST\_ITEM\_T Object Type

This type describes a type to assist in listing the contents of a directory.

### Syntax

```
CREATE OR REPLACE TYPE dbms_dbfs_content_list_item_t
  AUTHID DEFINER
AS OBJECT (
  path          VARCHAR2(1024),
  item_name     VARCHAR2(256),
  item_type     INTEGER);
```

### Fields

**Table 244–2** DBMS\_DBFS\_CONTENT\_LIST\_ITEM\_T Fields

Field	Description
path	Path to the path item
item_name	Name of the path item
item_type	Type of path item. (See <a href="#">DBMS_DBFS_CONTENT Constants - Path Name Types</a> )

## DBMS\_DBFS\_CONTENT\_PROPERTY\_T Object Type

This type describes a single (name, value, typecode) property tuple. All properties (standard, optional, and user-defined) are described using such tuples.

The type is used by both the client-facing interfaces and by store providers for the DBMS\_DBFS\_CONTENT interface.

### Syntax

```
CREATE OR REPLACE TYPE dbms_dbfs_content_property_t
  AUTHID DEFINER
AS OBJECT (
  propname      VARCHAR2(32),
  propvalue     VARCHAR2(1024),
  typecode      INTEGER);
```

### Fields

**Table 244–3** DBMS\_DBFS\_CONTENT\_PROPERTY\_T Fields

Field	Description
prop_name	Name of property
prop_value	Value of property
typecode	Property type (See Constants in DBMS_TYPES)



## DBMS\_DBFS\_CONTENT\_LIST\_ITEMS\_T Table Type

This type is a variable-sized array of [DBMS\\_DBFS\\_CONTENT\\_LIST\\_ITEM\\_T Object Type](#). It is used by both the client-facing interfaces and by store providers for the [DBMS\\_DBFS\\_CONTENT](#) interface.

### Syntax

```
CREATE OR REPLACE TYPE dbms_dbfs_content_list_items_t AS  
TABLE OF dbms_dbfs_content_list_item_t;
```

## DBMS\_DBFS\_CONTENT\_PROPERTIES\_T Table Type

This type is a variable-sized array of property tuples of [DBMS\\_DBFS\\_CONTENT\\_PROPERTY\\_T Object Type](#). It is used by both the client-facing interfaces and by store providers for the DBMS\_DBFS\_CONTENT interface.

### Syntax

```
CREATE OR REPLACE TYPE dbms_dbfs_content_properties_t AS  
TABLE OF dbms_dbfs_content_property_t;
```

## DBMS\_DBFS\_CONTENT\_RAW\_T Table Type

This type is an array of RAW. It is to enable RAW data transport for batch interfaces in the [DBMS\\_DBFS\\_CONTENT](#) interface.

### Syntax

```
CREATE OR REPLACE TYPE dbms_dbfs_content_raw_t AS  
TABLE OF RAW(32767);
```



---

---

## Database URI TYPEs

Oracle supports the `UriType` family of types that can be used to store and query Uri-refs inside the database. The `UriType` itself is an abstract object type and the `HTTPURITYPE`, `XDBURITYPE` and `DBURITYPE` are subtypes of it.

You can create a `UriType` column and store instances of the `DBURITYPE`, `XDBURITYPE` or the `HTTPURITYPE` inside of it. You can also define your own subtypes of the `UriType` to handle different URL protocols.

Oracle also provides a `UriFactory` package that can be used as a factory method to automatically generate various instances of these `UriTypes` by scanning the prefix, such as `http://` or `/oradb`. You can also register your subtype and provide the prefix that you support. For instance, if you have written a subtype to handle the gopher protocol, you can register the prefix `gopher://` to be handled by your subtype. The `UriFactory` will then generate your subtype instance for any URL starting with that prefix.

This chapter contains the following topics:

- [Summary of URITYPE Supertype Subprograms](#)
- [Summary of HTTPURITYPE Subtype Subprograms](#)
- [Summary of DBURITYPE Subtype Subprograms](#)
- [Summary of XDBURITYPE Subtype Subprograms](#)
- [Summary of URIFACTORY Package Subprograms](#)

**See Also:**

- *Oracle XML DB Developer's Guide*

## Summary of URITYPE Supertype Subprograms

The `UriType` is the abstract super type. It provides a standard set of functions to get the value pointed to by the URI. The actual implementation of the protocol must be defined by the subtypes of this type.

Instances of this type cannot be created directly. However, you can create columns of this type and store subtype instances in it, and also select from columns without knowing the instance of the URL stored.

**Table 245–1 URITYPE Type Subprograms**

Method	Description
<a href="#">GETBLOB</a> on page 245-3	Returns the BLOB located at the address specified by the URL.
<a href="#">GETCLOB</a> on page 245-4	Returns the CLOB located at the address specified by the URL.
<a href="#">GETCONTENTTYPE</a> on page 245-5	Returns the URL, in escaped format, stored inside the <code>UriType</code> instance.
<a href="#">GETEXTERNALURL</a> on page 245-6	Returns the URL, in escaped format, stored inside the <code>UriType</code> instance.
<a href="#">GETURL</a> on page 245-7	Returns the URL, in non-escaped format, stored inside the <code>UriType</code> instance.
<a href="#">GETXML</a> on page 245-8	Returns the <code>XMLType</code> located at the address specified by the URL.

## GETBLOB

This function returns the BLOB located at the address specified by the URL. This function can be overridden in the subtype instances. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getBlob() RETURN BLOB;</pre>	This function returns the BLOB located at the address specified by the URL.
<pre>MEMBER FUNCTION getBlob(     content OUT     VARCHAR2) RETURN BLOB;</pre>	This function returns the BLOB located at the address specified by the URL and the content type.
<pre>FUNCTION getBlob(     csid IN NUMBER) RETURN BLOB;</pre>	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error

## GETCLOB

This function returns the CLOB located at the address specified by the URL. This function can be overridden in the subtype instances. This function returns either a permanent CLOB or a temporary CLOB. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

<b>Syntax</b>	<b>Description</b>
<pre>MEMBER FUNCTION getClob()  RETURN clob;</pre>	This function returns the CLOB located at the address specified by the URL.
<pre>MEMBER FUNCTION getClob(     content OUT     VARCHAR2)  RETURN clob;</pre>	This function returns the CLOB located at the address specified by the URL and the content type.

<b>Parameter</b>	<b>IN / OUT</b>	<b>Description</b>
content	(OUT)	Content type of the document to which URI is pointing.



## GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI. This function can be overridden in the subtype instances. This function returns the content type as VARCHAR2.

### Syntax

```
MEMBER FUNCTION getContentType()  
RETURN VARCHAR2;
```

## GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `UriType` instance. The subtype instances override this member function to provide additional semantics. For instance, the `HTTPURITYPE` function does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType` instance.

### Syntax

```
MEMBER FUNCTION getExternalUrl()  
RETURN varchar2;
```

## GETURL

This function returns the URL, in non-escaped format, stored inside the `UriType` instance. The subtype instances override this member function to provide additional semantics. For instance, the `HTTPURITYPE` function does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType` instance.

### Syntax

```
MEMBER FUNCTION getUrl()  
RETURN varchar2;
```

## GETXML

This function returns the `XMLType` located at the address specified by the URL. This function can be overridden in the subtype instances. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getXML()  RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL.
<pre>MEMBER FUNCTION getXML(     content OUT     VARCHAR2) RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

---

## Summary of HTTPURITYPE Subtype Subprograms

The HTTPURITYPE is a subtype of the `UriType` that provides support for the HTTP protocol. This uses the `UTL_HTTP` package underneath to access the HTTP URLs. Proxy and secure wallets are not supported in this release.

**Table 245–2 HTTPURITYPE Type Subprograms**

Method	Description
<a href="#">CREATEURI</a> on page 245-10	Creates an instance of HTTPURITYPE from the given URI.
<a href="#">GETBLOB</a> on page 245-11	Returns the BLOB located at the address specified by the URL.
<a href="#">GETCLOB</a> on page 245-12	Returns the CLOB located at the address specified by the URL.
<a href="#">GETCONTENTTYPE</a> on page 245-13	Returns the content type of the document pointed to by the URI.
<a href="#">GETEXTERNALURL</a> on page 245-14	Returns the URL, in escaped format, stored inside the <code>UriType</code> instance.
<a href="#">GETURL</a> on page 245-7	Returns the URL, in non-escaped format, stored inside the <code>UriType</code> instance.
<a href="#">GETXML</a> on page 245-16	Returns the XMLType located at the address specified by the URL
<a href="#">HTTPURITYPE</a> on page 245-17	Creates an instance of HTTPURITYPE from the given URI.

## CREATEURI

This static function constructs a `HTTPURITYPE` instance. The `HTTPURITYPE` instance does not contain the prefix `http://` in the stored URL.

### Syntax

```
STATIC FUNCTION createUri(  
    url IN varchar2)  
RETURN HTTPURITYPE;
```

---

Parameter	IN / OUT	Description
-----------	----------	-------------

---

<code>url</code>	(IN)	The URL string containing a valid HTTP URL; escaped format.
------------------	------	---

---

## GETBLOB

This function returns the BLOB located at the address specified by the HTTP URL.

Syntax	Description
<pre>MEMBER FUNCTION getBlob() RETURN blob;</pre>	This function returns the BLOB located at the address specified by the HTTP URL.
<pre>MEMBER FUNCTION getBlob(     content OUT     VARCHAR2) RETURN blob;</pre>	This function returns the BLOB located at the address specified by the HTTP URL and the content type.
<pre>FUNCTION getBlob(     csid IN NUMBER) RETURN BLOB;</pre>	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.

## GETCLOB

This function returns the CLOB located by the HTTP URL address. If a temporary CLOB is returned, it must be freed.

Syntax	Description
MEMBER FUNCTION getClob()  RETURN clob;	Returns the CLOB located at the address specified by the HTTP URL.
MEMBER FUNCTION getClob( content OUT VARCHAR2) RETURN clob;	Returns the CLOB located at the address specified by the HTTP URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.



## GETCONTENTTYPE

Returns the content type of the document pointed to by the URI.

### **Syntax**

```
MEMBER FUNCTION getContentType()  
RETURN VARCHAR2;
```

## GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `HTTPURITYPE` instance. The subtype instances override this member function. The `HTTPURITYPE` function does not store the prefix `http://`, but generates it for the external URL.

### Syntax

```
MEMBER FUNCTION getExternalUrl()  
RETURN varchar2;
```

## GETURL

This function returns the URL, in non-escaped format, stored inside the `HTTPURITYPE` instance.

### Syntax

```
MEMBER FUNCTION getUrl()  
RETURN varchar2;
```

## GETXML

This function returns the `XMLType` located at the address specified by the URL. An error is thrown if the address does not point to a valid XML document.

Syntax	Description
<pre>MEMBER FUNCTION getXML()  RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL.
<pre>MEMBER FUNCTION getXML(     content OUT     VARCHAR2) RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
<code>content</code>	(OUT)	Content type of the document to which URI is pointing.

## HTTPURITYPE

This constructs a `HTTPURITYPE` instance. The `HTTPURITYPE` instance does not contain the prefix `http://` in the stored URL.

### Syntax

```
CONSTRUCTOR FUNCTION HTTPURITYPE(  
    url IN varchar2);
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid HTTP URL. The URL string is expected in escaped format. For example, non-url characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

---

## Summary of DBURITYPE Subtype Subprograms

The `DBURITYPE` is a subtype of the `UriType` that provides support for `DBUri`-refs. A `DBUri`-ref is an intra-database URL that can be used to reference any row or row-column data in the database. The URL is specified as an XPath expression over a XML visualization of the database. The schemas become elements which contain tables and views. These tables and view further contain the rows and columns inside them.

**Table 245–3** *DBURITYPE Type Subprograms*

Method	Description
<a href="#">CREATEURI</a> on page 245-19	Constructs a <code>DBURITYPE</code> instance.
<a href="#">DBURITYPE</a> on page 245-20	Creates an instance of <code>DBURITYPE</code> from the given URI.
<a href="#">GETBLOB</a> on page 245-21	Returns the <code>BLOB</code> located at the address specified by the <code>DBURITYPE</code> instance.
<a href="#">GETCLOB</a> on page 245-22	Returns the <code>CLOB</code> located at the address specified by the <code>DBURITYPE</code> instance.
<a href="#">GETCONTENTTYPE</a> on page 245-23	Returns the content type of the document pointed to by the URI.
<a href="#">GETEXTERNALURL</a> on page 245-24	Returns the URL, in escaped format, stored inside the <code>DBURITYPE</code> instance.
<a href="#">GETURL</a> on page 245-25	Returns the URL, in non-escaped format, stored inside the <code>DBURITYPE</code> instance.
<a href="#">GETXML</a> on page 245-26	Returns the <code>XMLType</code> located at the address specified by the URL.

## CREATEURI

This static function constructs a `DBURITYPE` instance. Parses the URL given and creates a `DBURITYPE` instance.

### Syntax

```
STATIC FUNCTION createUri(  
    url IN varchar2)  
RETURN DBURITYPE;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid <code>DBURITYPE</code> .

## DBURITYPE

This constructs a `DBURITYPE` instance.

### Syntax

```
CONSTRUCTOR FUNCTION DBURITYPE(  
    url IN varchar2);
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid <code>DBURITYPE</code> . The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.



## GETBLOB

This function returns the BLOB located at the address specified by the URL. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getBlob() RETURN blob;</pre>	This function returns the BLOB located at the address specified by the URL.
<pre>MEMBER FUNCTION getBlob(     content OUT     VARCHAR2) RETURN blob;</pre>	This function returns the BLOB located at the address specified by the URL and the content type.
<pre>FUNCTION getBlob(     csid IN NUMBER) RETURN BLOB;</pre>	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.

## GETCLOB

This function returns the CLOB located at the address specified by the `DBURITYPE` instance. If a temporary CLOB is returned, it must be freed. The document returned may be an XML document or a text document. When the `DBUri-ref` identifies an element in the XPath, the result is a well-formed XML document. On the other hand, if it identifies a text node, then what is returned is only the text content of the column or attribute. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getClob()  RETURN clob;</pre>	Returns the CLOB located at the address specified by the <code>DBURITYPE</code> instance.
<pre>MEMBER FUNCTION getClob(     content OUT     VARCHAR2) RETURN clob;</pre>	Returns the CLOB located at the address specified by the <code>DBURITYPE</code> instance and the content type.

Parameter	IN / OUT	Description
<code>content</code>	(OUT)	Content type of the document to which URI is pointing.

## GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI.

### Syntax

```
MEMBER FUNCTION getContentType()  
RETURN VARCHAR2;
```

## GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `DBURITYPE` instance. The `DBUri` servlet URL that processes the `DBURITYPE` has to be appended before using the escaped URL in Web pages.

### Syntax

```
MEMBER FUNCTION getExternalUrl()  
RETURN varchar2;
```

## GETURL

This function returns the URL, in non-escaped format, stored inside the DBURITYPE instance.

### Syntax

```
MEMBER FUNCTION getUrl()  
RETURN varchar2;
```

## GETXML

This function returns the `XMLType` located at the address specified by the URL. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getXML()  RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL.
<pre>MEMBER FUNCTION getXML(     content OUT     VARCHAR2) RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

---

## Summary of XDBURITYPE Subtype Subprograms

XDBURITYPE is a new subtype of URITYPE. It provides a way to expose documents in the Oracle XML DB hierarchy as URIs that can be embedded in any URITYPE column in a table. The URL part of the URI is the hierarchical name of the XML document it refers to. The optional fragment part uses the XPath syntax, and is separated from the URL part by '#'. The more general XPointer syntax for specifying a fragment is not currently supported.

**Table 245–4 XDBURITYPE Type Subprograms**

Method	Description
<a href="#">CREATEURI</a> on page 245-28	Returns the <code>UriType</code> corresponding to the specified URL.
<a href="#">GETBLOB</a> on page 245-29	Returns the <code>BLOB</code> corresponding to the contents of the document specified by the <code>XDBURITYPE</code> instance.
<a href="#">GETCLOB</a> on page 245-22	Returns the <code>CLOB</code> corresponding to the contents of the document specified by the <code>XDBURITYPE</code> instance.
<a href="#">GETCONTENTTYPE</a> on page 245-31	Returns the content type of the document pointed to by the URI.
<a href="#">GETEXTERNALURL</a> on page 245-24	Returns the URL, in escaped format, stored inside the <code>XDBURITYPE</code> instance.
<a href="#">GETURL</a> on page 245-25	Returns the URL, in non-escaped format, stored inside the <code>XDBURITYPE</code> instance.
<a href="#">GETXML</a> on page 245-34	Returns the <code>XMLType</code> corresponding to the contents of the document specified by the URL.
<a href="#">XDBURITYPE</a> on page 245-35	Creates an instance of <code>XDBURITYPE</code> from the given URI.

## CREATEURI

This static function constructs a `XDBURITYPE` instance. Parses the URL given and creates a `XDBURITYPE` instance.

### Syntax

```
STATIC FUNCTION createUri(  
    url IN varchar2)  
RETURN XDBURITYPE
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid <code>XDBURITYPE</code> .



## GETBLOB

This function returns the BLOB located at the address specified by the XDBURITYPE instance. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getBlob() RETURN blob;</pre>	This function returns the BLOB located at the address specified by the URL.
<pre>MEMBER FUNCTION getBlob(     content OUT     VARCHAR2) RETURN blob;</pre>	This function returns the BLOB located at the address specified by the URL and the content type.
<pre>FUNCTION getBlob(     csid IN NUMBER) RETURN BLOB;</pre>	This function returns the BLOB located at the address specified by the URL in the specified character set.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.
csid	(IN)	Character set id of the document. Must be a valid Oracle id and greater than 0; otherwise returns an error.

## GETCLOB

This function returns the CLOB located at the address specified by the `XDBURITYPE` instance. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getClob()  RETURN clob;</pre>	Returns the CLOB located at the address specified by the <code>XDBUriType</code> instance.
<pre>MEMBER FUNCTION getClob(     content OUT     VARCHAR2) RETURN clob;</pre>	Returns the CLOB located at the address specified by the <code>XDBUriType</code> instance and the content type.

Parameter	IN / OUT	Description
<code>content</code>	(OUT)	Content type of the document to which URI is pointing.

## GETCONTENTTYPE

This function returns the content type of the document pointed to by the URI. This function returns the content type as VARCHAR2.

### **Syntax**

```
MEMBER FUNCTION getContentType()  
RETURN VARCHAR2;
```

## GETEXTERNALURL

This function returns the URL, in escaped format, stored inside the `XDBURITYPE` instance.

### Syntax

```
MEMBER FUNCTION getExternalUrl()  
RETURN varchar2;
```

## GETURL

This function returns the URL, in non-escaped format, stored inside the XDBURITYPE instance.

### Syntax

```
MEMBER FUNCTION getUrl()  
RETURN varchar2;
```

## GETXML

This function returns the `XMLType` located at the address specified by the URL. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION getXML()  RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL.
<pre>MEMBER FUNCTION getXML(     content OUT     VARCHAR2)  RETURN XMLType;</pre>	This function returns the <code>XMLType</code> located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
<code>content</code>	(OUT)	Content type of the document to which URI is pointing.

## XDBURITYPE

This constructs a XDBURITYPE instance.

### Syntax

```
CONSTRUCTOR FUNCTION XDBURITYPE(
    url      IN  VARCHAR2,
    flags    IN  RAW := NULL)
RETURN self AS RESULT;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid XDBURITYPE. The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.
flags	(IN)	Possible values are: <ul style="list-style-type: none"> <li>■ 1 - Expand all XInclude elements before returning the result contents. If any XInclude element cannot be successfully resolved according to the XInclude fallback semantics, then an error is raised.</li> <li>■ 2 - Indicates that any errors during document retrieval should be suppressed</li> <li>■ 3 - Both flag bits (1, 2) are enabled</li> </ul>

## Summary of URIFACTORY Package Subprograms

The `UriFactory` package contains factory methods that can be used to generate the appropriate instance of the URI types without having to hard code the implementation in the program.

The `UriFactory` package also provides the ability to register new subtypes of the `UriType` to handle various other protocols. For example, you can invent a new protocol `ecom://` and define a subtype of the `UriType` to handle that protocol and register it with `UriFactory`. After that any factory method would generate the new subtype instance if it sees the `ecom://` prefix.

**Table 245–5 URIFACTORY Type Subprograms**

Method	Description
<a href="#">GETURI</a> on page 245-37	Returns the correct URL handler for the given URL string.
<a href="#">ESCAPEURI</a> on page 245-38	Returns a URL in escaped format.
<a href="#">UNESCAPEURI</a> on page 245-39	Returns a URL in unescaped format.
<a href="#">REGISTERURLHANDLER</a> on page 245-40	Registers a particular type name for handling a particular URL.
<a href="#">UNREGISTERURLHANDLER</a> on page 245-41	Unregisters a URL handler.



## GETURI

This factory method returns the correct URI handler for the given URI string. It returns a subtype instance of the `UriType` that can handle the protocol. By default, it always creates an `XDBURITYPE` instance, if it cannot resolve the URL. A URL handler can be registered for a particular prefix using the [REGISTERURLHANDLER](#) function. If the prefix matches, [GETURI](#) would then use that subtype.

### Syntax

```
FUNCTION getUri(  
    url IN Varchar2)  
RETURN UriType;
```

Parameter	IN / OUT	Description
uri	(IN)	The URL string, in escaped format, containing a valid HTTP URL.

## ESCAPEURI

This function returns a URL in escaped format. The subtype instances override this member function to provide additional semantics. For instance, the `HTTPURITYPE` does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the [GETEXTERNALURL](#) function or the [GETURI](#) function to get to the URL value instead of using the attribute present in the `UriType`.

### Syntax

```
MEMBER FUNCTION escapeUri()  
RETURN varchar2;
```

Parameter	IN / OUT	Description
<code>url</code>	(IN)	The URL string to be returned in escaped format.

## UNESCAPEURI

This function returns a URL in unescaped format. This function is the reverse of the [ESCAPEURI](#) function. This function scans the string and converts any non-URL hexadecimal characters into the equivalent UTF-8 characters. Since the return type is a `VARCHAR2`, the characters would be converted into the equivalent characters as defined by the database character set.

### Syntax

```
FUNCTION unescapeUri()  
RETURN varchar2;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string to be returned in unescaped format.

## REGISTERURLHANDLER

Registers a particular type name for handling a particular URL. The type specified must be valid and must be a subtype of the `UriType` or one of its subtypes. It must also implement the `createUrl` static member function. This function is called by the [GETURI](#) function to generate an instance of the type. The `stripprefix` parameter indicates that the prefix must be stripped off before calling this function.

### Syntax

```
PROCEDURE registerUrlHandler(  
    prefix IN varchar2,  
    schemaName IN varchar2,  
    typename IN varchar2,  
    ignoreCase IN boolean := true,  
    stripprefix IN boolean := true);
```

Parameter	IN / OUT	Description
<code>prefix</code>	(IN)	The prefix to handle; for example, <code>http://</code> .
<code>schemaName</code>	(IN)	Name of the schema where the type resides; case sensitive.
<code>typename</code>	(IN)	The name of the type to handle the URL; case sensitive.
<code>ignoreCase</code>	(IN)	Ignore case when matching prefixes.
<code>stripprefix</code>	(IN)	Strip prefix before generating the instance of the type.

## UNREGISTERURLHANDLER

This procedure unregisters a URL handler. This only unregisters user registered handler prefixes and not predefined system prefixes such as `http://`.

### Syntax

```
PROCEDURE unregisterUrlHandler(  
    prefix IN varchar2);
```

Parameter	IN / OUT	Description
prefix	(IN)	The prefix to be unregistered.



PL/SQL users can use the `DBMS_AQ` package to enqueue and dequeue messages from JMS queues. The JMS types member and static functions and procedures in this chapter are needed to populate JMS messages for enqueueing or to interpret a dequeued JMS message.

This chapter contains these topics:

- [Using JMS Types](#)
  - Overview
  - Java Versus PL/SQL Data Types
  - More on Bytes, Stream and Map Messages
  - Upcasting and Downcasting Between General and Specific Messages
  - JMS Types Error Reporting
  - Oracle JMS Type Constants
  - `CONVERT_JMS_SELECTOR`
- [Summary of JMS Types](#)

---

## Using JMS Types

- [Overview](#)
- [Java Versus PL/SQL Data Types](#)
- [More on Bytes, Stream and Map Messages](#)
- [Upcasting and Downcasting Between General and Specific Messages](#)
- [JMS Types Error Reporting](#)
- [Oracle JMS Type Constants](#)
- [JMS Types Error Reporting](#)
- [Oracle JMS Type Constants](#)
- [CONVERT\\_JMS\\_SELECTOR](#)



## Overview

Java Message Service (JMS) is a well known public standard interface for accessing messaging systems. Oracle JMS (OJMS) implements JMS based on Oracle Streams Advanced Queuing (AQ) and a relational database system (RDBMS). Messages are stored in queues as OJMS specific ADTs. Java clients use OJMS packages to enqueue, dequeue, and manipulate these messages.

PL/SQL users, on the other hand, use the `DBMS_AQ` package to enqueue and dequeue JMS messages and the member functions in this chapter to populate and interpret them. Oracle Streams AQ offers such member functions for the following JMS ADTs:

- `aq$_jms_header`
- `aq$_jms_message`
- `aq$_jms_text_message`
- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

In addition to these populating and interpreting member functions, Oracle Streams AQ offers:

- Casting between `aq$_jms_message` and other message ADTs.
- PL/SQL stored procedures for converting JMS selectors to equivalent Oracle Streams AQ rules

## Java Versus PL/SQL Data Types

Data types do not map one-to-one between PL/SQL and Java.

Some Java types, such as `BYTE` and `SHORT`, are not present in PL/SQL. PL/SQL type `INT` was chosen to represent these types. If a PL/SQL `INT` value intended to hold a Java `BYTE` or `SHORT` value exceeds the corresponding range Java enforces, an out-of-range error is thrown.

Other Java types have more than one counterpart in PL/SQL with different capabilities. A Java String can be represented by both `VARCHAR2` and `CLOB`, but `VARCHAR2` has a maximum limit of 4000 bytes. When retrieving `TEXT` data from map, stream, and bytes message types, a `CLOB` is always returned. When updating the map, stream and bytes message types, users can submit either a `VARCHAR2` or `CLOB`.

Similarly, a Java `BYTE ARRAY` can be represented by both `RAW` and `BLOB`, with `RAW` having a maximum size of 32767. When retrieving `BYTE ARRAY` data from map, stream, and bytes message types, a `BLOB` is always returned. When updating the map, stream and bytes message types, users can submit either a `RAW` or `BLOB`.

**See Also:** JMS specification 3.11.3, Conversion Provided by `StreamMessage` and `MapMessage`

### New JMS Support in Oracle Database 10g

In Oracle Database 10g, a new `AQ$_JMS_VALUE` ADT has been added in the `SYS` schema for OJMS PL/SQL users. It is specifically used to implement the `read_object` procedure of `aq$_jms_stream_message` and `get_object` procedure of `aq$_jms_map_message`, to mimic the Java general object class `Object`. `AQ$_JMS_VALUE` ADT can represent any data type that JMS `StreamMessage` and `MapMessage` can hold.

The collection ADT `AQ$_JMS_NAMEARRAY` was added for the `getNames` method of `MapMessage`. It holds an array of names.

In this release the ADT `AQ$_JMS_EXCEPTION` was added to represent a Java exception thrown in an OJMS JAVA stored procedure on the PL/SQL side. Now you can retrieve a Java exception thrown by an OJMS stored procedure and analyze it on the PL/SQL side.

## More on Bytes, Stream and Map Messages

Oracle uses Java stored procedure to implement some of the procedures of `AQ$_MAP_MESSAGE`, `AQ$_JMS_STREAM_MESSAGE`, and `AQ$_JMS_BYTES_MESSAGE` types. These types have some common functionalities that are different from `AQ$_JMS_TEXT_MESSAGE` type. This section discusses these common functionalities.

This section contains these topics:

- [Using Java Stored Procedures to Encode and Decode Oracle Streams AQ Messages](#)
- [Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages](#)
- [Differences Between Bytes and Stream Messages](#)
- [Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes](#)

### Using Java Stored Procedures to Encode and Decode Oracle Streams AQ Messages

The major difference between map, stream, bytes, and other messages is that the message payload is encoded as a byte stream by JAVA. Retrieving and updating these payloads in PL/SQL therefore requires Oracle JAVA stored procedures.

A message payload is stored in two places during processing. On the PL/SQL side it is stored as the data members of a JMS message ADT, and on the Jserv side it is stored as a static variable. (Jserv is the JVM inside Oracle Database.) When the payload is processed, the payload data is first transformed to a static variable on the Jserv side. Once the static variable is initialized, all later updates on the message payload are performed on this static variable. At the end of processing, payload data is flushed back to the PL/SQL side.

Oracle provides member procedures that maintain the status of the Jserv static variable and enforce rules when calling these member procedures. These procedures are in the following ADTs:

- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

### Initialize the Jserv Static Variable

Before you make any other calls to manipulate the payload data, the Jserv static variable must be properly initialized. This is done by calling the `prepare` or `clear_body` procedure. The `prepare` procedure uses the payload data in PL/SQL ADTs to initialize the static variable, while `clear_body` initializes the static variable to an empty payload (empty hashtable or stream).

---

**Note:** It is important to call the `prepare` or `clear_body` procedure before any other calls to properly initialize the Jserv static variables. Usually these two methods are called once at the beginning. But they can be called multiple times for one message. Any call of these two methods without first calling the `flush` procedure wipes out all updates made to the messages.

---

## Get the Payload Data Back to PL/SQL

Calling the `flush` procedure synchronizes changes made to the `Jserv` static variable back to the PL/SQL ADTs. The `flush` call is required when you want the changes made to be reflected in the ADT payload. It is important to synchronize the changes back to the ADT, because it is the ADT payload that matters.

## Garbage Collect the Static Variable

The `clean` procedure forces garbage collection of the static variable. It is there to do cleanup and free JVM memory. You can avoid memory leaks by doing it immediately after finishing processing the message.

## Use a Message Store: A Static Variable Collection

Instead of a single static variable, Oracle uses a collection of static variables to process the message payload on the `Jserv` side. This collection is called the message store. Each `map`, `bytes`, or `stream` message type has its own message store within one session.

Oracle uses the operation ID parameter to locate the correct static variable to work on within the message store. Initialization calls such as `prepare` and `clear_body` give users an operation ID, which is used in later message access.

After users complete message processing, they must call the `clean` procedure with the operation ID to clean up the message store. This avoids possible memory leaks. The `clean_all` static procedures of message ADTs `aq$_jms_bytes_message`, `aq$_jms_map_message`, and `aq$_jms_stream_message` clean up all static variables of their corresponding message stores.

## Typical Calling Sequences

This section describes typical procedures for retrieving and populating messages.

Here is a typical procedure for retrieving messages

1. Call `prepare` for a message.  
This call also gives you an operation ID if you do not specify one.
2. Call multiple retrieving procedures with the provided operation ID.
3. Call the `clean` procedure with the provided operation ID.

Here is a typical procedure for populating messages:

1. Call `clear_body` for a message.  
For `aq$_jms_map_message`, you can also call `prepare` to update the message based on the existing payload. This call also gives you an operation ID if you do not specify one.
2. Call multiple updating procedures with the provided operation ID.
3. Call the `flush` method with the provided operation ID.
4. Call the `clean` procedure with the provided operation ID.

## Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages

According to the JMS specification, when a message is received, its body is read-only. Users can call the `clear_body` method to make the body writable. This method erases the current message body and sets the message body to be empty.

The OJMS JAVA API follows the rule set by JMS specification. In updating the JMS message ADTs in PL/SQL, however, Oracle enforces the rule selectively:

- Map messages

The restriction is relaxed, because adding more entries on top of a existing map payload is a convenient way for users to update the payload. Therefore there are no read-only or write-only modes for map messages.

- Stream and bytes messages

The restriction is not relaxed, because these payloads use a stream when reading and writing data. It is difficult to update the payload while in the middle of a stream. Oracle enforces read-only and write-only modes in processing stream and bytes message payloads. Calling the `prepare` procedure initializes the message payload in read-only mode. Calling the `clear_body` procedure initializes the message payload in write-only mode.

Calling the `reset` procedure resets the pointer to the beginning of the stream and switches the mode from write-only to read-only. The `reset` procedure keeps the updates made to the message payload in the `Jserv` static variable.

The `prepare` procedure, on the other hand, overwrites the message payload in the `Jserv` static variable with the payload in the PL/SQL ADT.

Oracle provides member function `get_mode` for users to query the mode.

## Differences Between Bytes and Stream Messages

Member functions of bytes messages are not exactly the same as those of stream messages. Stream messages are encoded using `Java ObjectOutputStream` and bytes messages are encoded using `Java DataOutputStream`. In stream messages each primitive type is written and read as a `Java Object`, but in a bytes message they are written and read as raw bytes according to the encoding mechanism of `DataOutputStream`.

For stream messages, the `read_bytes` method works on a stream of bytes to the end of the byte array field written by the corresponding `write_bytes` method. The `read_bytes` method of bytes message works on a stream of bytes to the end of the whole byte stream. This is why the `read_bytes` member procedure of `aq$_bytes_message` also requires a `length` parameter to tell how long it is to read.

You will not see a type conversion error raised by bytes message, because bytes messages do not support type conversion.

Methods `get_unsigned_byte` and `get_unsigned_short` are available for bytes messages, but not for stream messages. This is because stream messages read `Java` objects, and there are no `Java` objects as unsigned bytes or unsigned shorts.

Methods `read_string` and `write_string` methods are not available for bytes messages. The bytes message ADT must enforce some character encoding. It has methods `read_utf` and `write_utf` which support `utf-8` encoding.

---

**Note:** All data written by bytes messages use `DataOutputStream` as the basis. See JDK API documentation [JavaSoft.com](http://JavaSoft.com) for details on how the data is encoded into bytes.

---

## Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

The payloads of bytes, map, and stream message types are stored as either RAW or BLOB in the database. In this release Oracle Streams AQ provides the following member functions to set and get these payloads as raw bytes without interpreting them:

```
set_bytes(payload IN BLOB)
set_bytes(payload IN RAW)
get_bytes(payload OUT BLOB)
get_bytes(payload OUT RAW)
```

These functions were provided for bytes messages in Oracle9i Release 2 (9.2).

## Upcasting and Downcasting Between General and Specific Messages

OJMS ADT `aq$_jms_message` is used to represent a general message, so that different types of messages can reside on the same Oracle Streams AQ queue. Oracle Streams AQ supports retrieving and populating of `aq$_jms_message` by supporting upcasting and downcasting between this ADT and ADTs of specific message types.

To read an `aq$_jms_message`, you must first downcast it to a specific message type according to its `message_type` field

To populate an `aq$_jms_message`, you must first populate a specific message and upcast it to `aq$_jms_message`. This avoids copying all member functions of other specific message ADTs to this ADT. It also guarantees that the manipulation of this ADT is consistent with other specific message ADTs.

## JMS Types Error Reporting

Table 246–1 lists Oracle JMS types related errors.

**Table 246–1 Oracle JMS Types Errors**

ORA error number	dbms_jms_plsql package constants	Explanation
ORA-24190	ERROR_DATA_OVERFLOW	The payload data exceeds the size that an out parameter can hold. For example, the <code>get_text</code> procedure with a <code>VARCHAR2</code> parameter of <code>aq\$_jms_text_message</code> or <code>get_bytes</code> procedure with a <code>RAW</code> parameter of <code>aq\$_jms_bytes_message</code> .
ORA-24191	ERROR_PROP_NAME_EXIST	Setting a property that is previous set
ORA-24192	ERROR_PROP_NAME_NULL	Occurs when setting a property with null property name.
ORA-24193	ERROR_EXCEED_RANGE	PL/SQL number type exceeds the valid range of the respective Java type. For example <code>set_byte_property</code> , <code>set_short_property</code> of <code>aq\$_jms_head</code> ADT; <code>set_byte</code> and <code>set_short</code> of <code>aq\$_jms_map_message</code> ADT; <code>write_byte</code> and <code>write_short</code> of <code>aq\$_jms_stream_message</code> and <code>aq\$_jms_bytes_message</code> ADT.
ORA-24194	ERROR_TYPE_MISMATCH	The type conversion between the Java type of the retrieving method and the Java type of a field of the payload is not valid.
ORA-24195	ERROR_MAP_TOO_LARGE	The size of the map exceeds the <code>aq\$_jms_namearray</code> ADT capacity. The current size limit is 1024. You can use the <code>get_names</code> function with <code>offset</code> and <code>length</code> parameters to retrieve the name array in multiple small chunks.
ORA-24196	ERROR_WRONG_MODE	The message payload is being accessed with a wrong access mode. For example, trying to read a message payload with write-only mode or trying to write a message payload with the read-only mode.
ORA-24197	ERROR_JAVA_EXCEPTION	ORA-24197 error is raised when a Java exception is raised that does not fit in any of the other error categories. You can use the <code>get_exception</code> static procedure of <code>aq\$_jms_map_message</code> , <code>aq\$_jms_bytes_message</code> , and <code>aq\$_jms_stream_message</code> to retrieve the exception information last thrown by the Java stored procedure.  A single static variable is used to store the last exception and is overwritten if another exception is thrown before you retrieve it. A new ADT <code>aq\$_jms_exception</code> is created to represent the exception information on the PL/SQL side.
ORA-24198	ERROR_INVALID_ID	An invalid operation ID is being provided to access a message.
ORA-24199	ERROR_STORE_OVERFLOW	The number of messages (with the same type) that users are trying to manipulate exceeds the size of the message store on the Java stored procedure side. The current size of the store is 20. It unusual to need to manipulate more than 20 messages at the same time. A common mistake is to forget to call the <code>clean</code> procedure after using one message. The <code>clean</code> procedure frees the message slot for use by other messages attempting access.



## Oracle JMS Type Constants

This section lists some useful constants when dealing with message type functions.

### DBMS\_AQ Package Constants

DBMS\_AQ package constants specify different types of JMS messages. They are useful when dealing with general message types during upcasting and downcasting or constructing a general message with a specific message type:

```
JMS_TEXT_MESSAGE    CONSTANT BINARY_INTEGER;
JMS_BYTES_MESSAGE   CONSTANT BINARY_INTEGER;
JMS_STREAM_MESSAGE  CONSTANT BINARY_INTEGER;
JMS_MAP_MESSAGE     CONSTANT BINARY_INTEGER;
JMS_OBJECT_MESSAGE  CONSTANT BINARY_INTEGER;
```

### SYS.DBMS\_JMS\_PLSQL Package Constants

SYS.DBMS\_JMS\_PLSQL package constants are new in Oracle Database 10g.

These constants specify the mode of message payload. They are useful when interpreting the mode of the message payload returned from the `get_mode` function:

```
MESSAGE_ACCESS_READONLY  CONSTANT PLS_INTEGER;
MESSAGE_ACCESS_WRITEONLY CONSTANT PLS_INTEGER;
```

These constants specify the ADT type of an Oracle Streams AQ queue. They are useful during the conversion of JMS selectors to Oracle Streams AQ rules:

```
DESTPLOAD_JMSTYPE  CONSTANT PLS_INTEGER;
DESTPLOAD_USERADT  CONSTANT PLS_INTEGER;
DESTPLOAD_ANYDATA  CONSTANT PLS_INTEGER;
```

These constants specify the type of data that can be held by a `aq$_jms_value` type. They are useful when interpreting the `aq$_jms_value` returned by the `get_object` method of `AQ$_JMS_MAP_MESSAGE` or `read_object` method of `AQ$_JMS_STREAM_MESSAGE`:

```
DATA_TYPE_BYTE          CONSTANT PLS_INTEGER;
DATA_TYPE_SHORT         CONSTANT PLS_INTEGER;
DATA_TYPE_INTEGER       CONSTANT PLS_INTEGER;
DATA_TYPE_LONG          CONSTANT PLS_INTEGER;
DATA_TYPE_FLOAT         CONSTANT PLS_INTEGER;
DATA_TYPE_DOUBLE        CONSTANT PLS_INTEGER;
DATA_TYPE_BOOLEAN       CONSTANT PLS_INTEGER;
DATA_TYPE_CHARACTER     CONSTANT PLS_INTEGER;
DATA_TYPE_STRING        CONSTANT PLS_INTEGER;
DATA_TYPE_BYTES         CONSTANT PLS_INTEGER;
DATA_TYPE_UNSIGNED_BYTE CONSTANT PLS_INTEGER;
DATA_TYPE_UNSIGNED_SHORT CONSTANT PLS_INTEGER;
```

These constants specify the error number of the ORA errors that can be raised by the functions of message type ADTs. They are useful in user error handlers:

```
ERROR_DATA_OVERFLOW    CONSTANT PLS_INTEGER := -24190;
ERROR_PROP_NAME_EXIST  CONSTANT PLS_INTEGER := -24191;
ERROR_PROP_NAME_NULL   CONSTANT PLS_INTEGER := -24192;
ERROR_EXCEED_RANGE     CONSTANT PLS_INTEGER := -24193;
ERROR_TYPE_MISMATCH    CONSTANT PLS_INTEGER := -24194;
ERROR_MAP_TOO_LARGE    CONSTANT PLS_INTEGER := -24195;
ERROR_WRONG_MODE       CONSTANT PLS_INTEGER := -24196;
```

```
ERROR_JAVA_EXCEPTION CONSTANT PLS_INTEGER := -24197;  
ERROR_INVALID_ID      CONSTANT PLS_INTEGER := -24198;  
ERROR_STORE_OVERFLOW  CONSTANT PLS_INTEGER := -24199;
```

## CONVERT\_JMS\_SELECTOR

Oracle Database includes three stored procedures to help users convert JMS selectors into Oracle Streams AQ rules. These rules can be used in `ADD_SUBSCRIBER` operations as subscriber rules or in `DEQUEUE` operations as dequeue conditions. These procedures are in the `SYS.dbms_jms_plsql` package.

### Convert with Minimal Specification

The first procedure assumes the destination payload type is one of the JMS ADTs whose corresponding constant is `dbms_jms_plsql.DESTPLOAD_JMSTYPE` and also assumes that the J2EE compliant mode is true.

#### Syntax

```
Function convert_jms_selector(selector IN VARCHAR2) RETURN VARCHAR2
```

#### Returns

The converted Oracle Streams AQ rule or null if there is any conversion error.

#### Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

### Convert with Destination Payload Type Specified

The second procedure takes one more parameter: `dest_pload_type`. The conversion of a JMS selector to an Oracle Streams AQ rule happens only if this parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if `dest_pload_type` parameter is none of these three constants.

This function assumes that the J2EE compliant mode is true.

#### Syntax

```
Function convert_jms_selector(
    selector IN VARCHAR2,
    dest_pload_type IN PLS_INTEGER)
RETURN VARCHAR2
```

#### Returns

The converted Oracle Streams AQ rule or null if there is any conversion error.

#### Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

### Convert with Destination Payload Type and Compliant Mode Specified

The third procedure takes a `dest_pload_type` parameter and a `compliant` parameter. The conversion of a JMS selector to an Oracle Streams AQ rule happens only if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type`

parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if the `dest_pload_type` parameter is none of these three constants.

The `compliant` parameter controls if the conversion is in J2EE compliant mode or not. The noncompliant conversion of a JMS selector is for backward compatibility.

### Syntax

```
Function convert_jms_selector(  
    selector          IN  VARCHAR2,  
    dest_pload_type  IN  PLS_INTEGER,  
    compliant        IN  BOOLEAN )
```

### Returns

The converted Oracle Streams AQ rule or null if there is any conversion error.

### Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

## Summary of JMS Types

- [SYS.AQ\\$\\_JMS\\_MESSAGE](#) Type
- [SYS.AQ\\$\\_JMS\\_TEXT\\_MESSAGE](#) Type
- [SYS.AQ\\$\\_JMS\\_BYTES\\_MESSAGE](#) Type
- [SYS.AQ\\$\\_JMS\\_MAP\\_MESSAGE](#) Type
- [SYS.AQ\\$\\_JMS\\_STREAM\\_MESSAGE](#) Type
- [SYS.AQ\\$\\_JMS\\_OBJECT\\_MESSAGE](#) Type
- [SYS.AQ\\$\\_JMS\\_NAMESARRAY](#) Type
- [SYS.AQ\\$\\_JMS\\_VALUE](#) Type
- [SYS.AQ\\$\\_JMS\\_EXCEPTION](#) Type

## SYS.AQ\$\_JMS\_MESSAGE Type

This ADT type can represent any of five different JMS message types: text message, bytes message, stream message, map message, or object message. Queues created using this ADT can therefore store all five types of JMS messages.

This section contains these topics:

- [CONSTRUCT Static Functions](#)
- [Cast Methods](#)
- [JMS Header Methods](#)
- [System Properties Methods](#)
- [User Properties Methods](#)
- [Payload Methods](#)

### Syntax

```

TYPE AQ$_JMS_MESSAGE AS OBJECT(
  header          aq$_jms_header,
  senderid        varchar2(100),
  message_type    INT,
  text_len        INT,
  bytes_len       INT,
  text_vc         varchar2(4000),
  bytes_raw       raw(2000),
  text_lob        clob,
  bytes_lob       blob,
  STATIC FUNCTION construct (mtype          IN INT)
    RETURN aq$_jms_message,
  STATIC FUNCTION construct (text_msg      IN aq$_jms_text_message)
    RETURN aq$_jms_message,
  STATIC FUNCTION construct (bytes_msg     IN aq$_jms_bytes_message)
    RETURN aq$_jms_message,
  STATIC FUNCTION construct (stream_msg   IN aq$_jms_stream_message)
    RETURN aq$_jms_message,
  STATIC FUNCTION construct (map_msg      IN aq$_jms_map_message)
    RETURN aq$_jms_message,
  STATIC FUNCTION construct (object_msg   IN aq$_jms_object_message)
    RETURN aq$_jms_message,
  MEMBER FUNCTION cast_to_bytes_msg      RETURN aq$_jms_bytes_message,
  MEMBER FUNCTION cast_to_map_msg       RETURN aq$_jms_map_message,
  MEMBER FUNCTION cast_to_object_msg    RETURN aq$_jms_object_message,
  MEMBER FUNCTION cast_to_stream_msg    RETURN aq$_jms_stream_message,
  MEMBER FUNCTION cast_to_text_msg      RETURN aq$_jms_text_message,
  MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
  MEMBER PROCEDURE set_type      (type   IN VARCHAR),
  MEMBER PROCEDURE set_userid    (userid IN VARCHAR),
  MEMBER PROCEDURE set_appid     (appid  IN VARCHAR),
  MEMBER PROCEDURE set_groupid   (groupid IN VARCHAR),
  MEMBER PROCEDURE set_groupseq  (groupseq IN INT),
  MEMBER FUNCTION get_replyto    RETURN sys.aq$_agent,
  MEMBER FUNCTION get_type      RETURN VARCHAR,
  MEMBER FUNCTION get_userid    RETURN VARCHAR,
  MEMBER FUNCTION get_appid     RETURN VARCHAR,
  MEMBER FUNCTION get_groupid   RETURN VARCHAR,
  MEMBER FUNCTION get_groupseq  RETURN INT,
  MEMBER PROCEDURE clear_properties,

```

```

MEMBER PROCEDURE set_boolean_property (property_name IN VARCHAR,
    property_value IN BOOLEAN),
MEMBER PROCEDURE set_byte_property    (property_name IN VARCHAR,
    property_value IN INT),
MEMBER PROCEDURE set_double_property  (property_name IN VARCHAR,
    property_value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float_property   (property_name IN VARCHAR,
    property_value IN FLOAT),
MEMBER PROCEDURE set_int_property      (property_name IN VARCHAR,
    property_value IN INT),
MEMBER PROCEDURE set_long_property     (property_name IN VARCHAR,
    property_value IN NUMBER),
MEMBER PROCEDURE set_short_property    (property_name IN VARCHAR,
    property_value IN INT),
MEMBER PROCEDURE set_string_property   (property_name IN VARCHAR,
    property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property  (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property   (property_name IN VARCHAR)
    RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property     (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property       (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property     (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property    (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_text              (payload IN VARCHAR2),
MEMBER PROCEDURE set_text              (payload IN CLOB),
MEMBER PROCEDURE set_bytes             (payload IN RAW),
MEMBER PROCEDURE set_bytes             (payload IN BLOB),
MEMBER PROCEDURE get_text              (payload OUT VARCHAR2),
MEMBER PROCEDURE get_text              (payload OUT CLOB),
MEMBER PROCEDURE get_bytes             (payload OUT RAW),
MEMBER PROCEDURE get_bytes             (payload OUT BLOB);

```

## CONSTRUCT Static Functions

There are six CONSTRUCT static functions in this type.

### **STATIC FUNCTION construct (mtype IN INT) RETURN aq\$\_jms\_message**

Creates an instance of `aq$_jms_message`, which can hold a specific type of JMS message (`TextMessage`, `BytesMessage`, `MapMessage`, `StreamMessage` or `ObjectMessage`). The message type of the created `aq$_jms_message` instance depends on the `mtype` parameter passed to the `construct` method. Once a message has been constructed, it can be used to store JMS messages of the type it has been constructed to hold.

The `mtype` parameter must be one of the following constants described in "[Oracle JMS Type Constants](#)" on page 246-11:

```

DBMS_AQ.JMS_TEXT_MESSAGE
DBMS_AQ.JMS_BYTES_MESSAGE
DBMS_AQ.JMS_STREAM_MESSAGE
DBMS_AQ.JMS_MAP_MESSAGE
DBMS_AQ.JMS_OBJECT_MESSAGE

```

### **STATIC FUNCTION construct (text\_msg IN aq\$\_jms\_text\_message) RETURN aq\$\_jms\_message**

Creates an `aq$_jms_message` from an `aq$_jms_text_message`.

### **STATIC FUNCTION construct (bytes\_msg IN aq\$\_jms\_bytes\_message) RETURN aq\$\_**

**jms\_message;**

Creates an aq\$\_jms\_message from an aq\$\_jms\_bytes\_message.

**STATIC FUNCTION construct (stream\_msg IN aq\$\_jms\_stream\_message) RETURN aq\$\_jms\_message;**

Creates an aq\$\_jms\_message from an aq\$\_jms\_stream\_message.

**STATIC FUNCTION construct (map\_msg IN aq\$\_jms\_map\_message) RETURN aq\$\_jms\_message;**

Creates an aq\$\_jms\_message from an aq\$\_jms\_map\_message.

**STATIC FUNCTION construct (object\_msg IN aq\$\_jms\_object\_message) RETURN aq\$\_jms\_message;**

Creates an aq\$\_jms\_message from an aq\$\_jms\_object\_message.

## Cast Methods

**cast\_to\_bytes\_msg RETURN aq\$\_jms\_bytes\_message**

Casts an aq\$\_jms\_message to an aq\$\_jms\_bytes\_message. Returns an aq\$\_jms\_bytes\_message or null if the message\_type attribute of the aq\$\_jms\_message is not DBMS\_AQ.JMS\_BYTES\_MESSAGE. This function raises ORA-24198 if the message\_type field of the aq\$\_jms\_message is not DBMS\_AQJMS.JMS\_BYTES\_MESSAGE.

**cast\_to\_map\_msg RETURN aq\$\_jms\_map\_message**

Casts an aq\$\_jms\_message to an aq\$\_jms\_map\_message. Returns an aq\$\_jms\_map\_message or null if the message\_type attribute of the aq\$\_jms\_message is not DBMS\_AQ.JMS\_MAP\_MESSAGE. This function raises ORA-24198 if the message\_type field of the aq\$\_jms\_message is not DBMS\_AQJMS.JMS\_MAP\_MESSAGE.

**cast\_to\_object\_msg RETURN aq\$\_jms\_object\_message**

Casts an aq\$\_jms\_message to an aq\$\_jms\_object\_message. Returns an aq\$\_jms\_object\_message or null if the message\_type attribute of the aq\$\_jms\_message is not DBMS\_AQ.JMS\_OBJECT\_MESSAGE. This function raises ORA-24198 if the message\_type field of the aq\$\_jms\_message is not DBMS\_AQJMS.JMS\_OBJECT\_MESSAGE.

**cast\_to\_stream\_msg RETURN aq\$\_jms\_stream\_message**

Casts an aq\$\_jms\_message to an aq\$\_jms\_stream\_message. Returns an aq\$\_jms\_stream\_message or null if the message\_type attribute of the aq\$\_jms\_message is not DBMS\_AQ.JMS\_STREAM\_MESSAGE. This function raises ORA-24198 if the message\_type field of the aq\$\_jms\_message is not DBMS\_AQJMS.JMS\_STREAM\_MESSAGE.

**cast\_to\_text\_msg RETURN aq\$\_jms\_text\_message**

Casts an aq\$\_jms\_message to an aq\$\_jms\_text\_message. Returns an aq\$\_jms\_text\_message or null if the message\_type attribute of the aq\$\_jms\_message is not DBMS\_AQ.JMS\_TEXT\_MESSAGE. This function raises ORA-24198 if the message\_type field of the aq\$\_jms\_message is not DBMS\_AQJMS.JMS\_TEXT\_MESSAGE.

## JMS Header Methods

**set\_replyto (replyto IN sys.aq\$\_agent)**

Sets the replyto parameter, which corresponds to JMSReplyTo.



**get\_replyto RETURN sys.aq\$\_agent**

Returns `replyto`, which corresponds to `JMSReplyTo`.

**set\_type (type IN VARCHAR)**

Sets the JMS type, which can be any text and corresponds to `JMSType`.

**get\_type RETURN VARCHAR**

Returns `type`, which corresponds to `JMSType`.

**System Properties Methods****set\_userid (userid IN VARCHAR)**

Sets `userid`, which corresponds to `JMSXUserID`.

**set\_appid (appid IN VARCHAR)**

Sets `appid`, which corresponds to `JMSXAppID`.

**set\_groupid (groupid IN VARCHAR)**

Sets `groupid`, which corresponds to `JMSXGroupID`.

**set\_groupseq (groupseq IN INT)**

Sets `groupseq`, which corresponds to `JMSXGroupSeq`.

**get\_userid RETURN VARCHAR**

Returns `userid`, which corresponds to `JMSXUserID`.

**get\_appid RETURN VARCHAR**

Returns `appid`, which corresponds to `JMSXAppID`.

**get\_groupid RETURN VARCHAR**

Returns `groupid`, which corresponds to `JMSXGroupID`.

**get\_groupseq RETURN VARCHAR**

Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

**User Properties Methods****clear\_properties**

Clears all user properties. This procedure does not affect system properties.

**set\_boolean\_property (property\_name IN VARCHAR, property\_value IN BOOLEAN)**

Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value` in an internal representation (a `NUMBER` type). Raises exception `ORA-24191` if the property name exists or `ORA-24192` if the property name is null.

**set\_byte\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `byte` datatype. Raises exception `ORA-24191` if the property name exists, `ORA-24192` if the property name is null, or `ORA-24193` if the property value exceeds the valid range.

**set\_double\_property (property\_name IN VARCHAR, property\_value IN DOUBLE PRECISION)**

Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception `ORA-24191` if the property name exists or `ORA-24192` if the property name is null.

**set\_float\_property (property\_name IN VARCHAR, property\_value IN FLOAT)**

Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_int\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because the `INT` datatype is 38 bits in PL/SQL and Oracle Database. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_long\_property (property\_name IN VARCHAR, property\_value IN NUMBER)**

Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_short\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_string\_property (property\_name IN VARCHAR, property\_value IN VARCHAR)**

Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**get\_boolean\_property (property\_name IN VARCHAR) RETURN BOOLEAN**

If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_byte\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_double\_property (property\_name IN VARCHAR) RETURN DOUBLE PRECISION**

If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_float\_property (property\_name IN VARCHAR) RETURN FLOAT**

If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_int\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_long\_property (property\_name IN VARCHAR) RETURN NUMBER**

If the property with the corresponding property name passed in exists, and if it is a long property, then this function returns the value of the property. Otherwise it returns a null.

**get\_short\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a short property, then this function returns the value of the property. Otherwise it returns a null.

**get\_string\_property (property\_name IN VARCHAR) RETURN VARCHAR**

If the property with the corresponding property name passed in exists, and if it is a STRING property, then this function returns the value of the property. Otherwise it returns a null.

## Payload Methods

**set\_text (payload IN VARCHAR2)**

Sets the payload, a VARCHAR2 value, to an internal representation.

**set\_text (payload IN CLOB),**

Sets the payload, a CLOB value, to an internal representation.

**set\_bytes (payload IN RAW)**

Sets the payload, a RAW value, to an internal representation.

**set\_bytes (payload IN BLOB)**

Sets the payload, a BLOB value, to an internal representation.

**get\_text (payload OUT VARCHAR2)**

Puts the internal representation of the payload into a VARCHAR2 variable payload.

**get\_text (payload OUT CLOB)**

Puts the internal representation of the payload into a CLOB variable payload.

**get\_bytes (payload OUT RAW)**

Puts the internal representation of the payload into a RAW variable payload.

**get\_bytes (payload OUT BLOB)**

Puts the internal representation of the payload into a BLOB variable payload.

## SYS.AQ\$\_JMS\_TEXT\_MESSAGE Type

This type is the ADT used to store a `TextMessage` in an Oracle Streams AQ queue.

This section contains these topics:

- [CONSTRUCT Function](#)
- [JMS Header Methods](#)
- [System Properties Methods](#)
- [User Properties Methods](#)
- [Payload Methods](#)

### Syntax

```

TYPE AQ$_JMS_TEXT_MESSAGE AS OBJECT(
  header      aq$_jms_header,
  text_len    INT,
  text_vc     varchar2(4000),
  text_lob    clob,
  STATIC FUNCTION construct      RETURN aq$_jms_text_message,
  MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
  MEMBER PROCEDURE set_type     (type     IN VARCHAR),
  MEMBER FUNCTION  get_replyto  RETURN sys.aq$_agent,
  MEMBER FUNCTION  get_type     RETURN VARCHAR,
  MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
  MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
  MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
  MEMBER PROCEDURE set_groupseq (groupseq IN INT),
  MEMBER FUNCTION  get_userid   RETURN VARCHAR,
  MEMBER FUNCTION  get_appid    RETURN VARCHAR,
  MEMBER FUNCTION  get_groupid  RETURN VARCHAR,
  MEMBER FUNCTION  get_groupseq RETURN INT,
  MEMBER PROCEDURE clear_properties,
  MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
    property_value IN BOOLEAN),
  MEMBER PROCEDURE set_byte_property  (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
    property_value IN DOUBLE PRECISION),
  MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
    property_value IN FLOAT),
  MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
    property_value IN NUMBER),
  MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
    property_value IN VARCHAR),
  MEMBER FUNCTION  get_boolean_property (property_name IN VARCHAR)
    RETURN BOOLEAN,
  MEMBER FUNCTION  get_byte_property    (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION  get_double_property  (property_name IN VARCHAR)
    RETURN DOUBLE PRECISION,
  MEMBER FUNCTION  get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
  MEMBER FUNCTION  get_int_property     (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION  get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
  MEMBER FUNCTION  get_short_property   (property_name IN VARCHAR) RETURN INT,

```

```

MEMBER FUNCTION get_string_property (property_name IN VARCHAR)
RETURN VARCHAR,
MEMBER PROCEDURE set_text          (payload IN VARCHAR2),
MEMBER PROCEDURE set_text          (payload IN CLOB),
MEMBER PROCEDURE get_text          (payload OUT VARCHAR2),
MEMBER PROCEDURE get_text          (payload OUT CLOB);

```

## CONSTRUCT Function

### **STATIC FUNCTION construct RETURN aq\$\_jms\_text\_message**

Creates an empty aq\$\_jms\_text\_message.

## JMS Header Methods

### **set\_replyto (replyto IN sys.aq\$\_agent)**

Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

### **set\_type (type IN VARCHAR)**

Sets the JMS type, which can be any text, and which corresponds to JMSType in JMS.

### **get\_replyto RETURN sys.aq\$\_agent**

Returns replyto, which corresponds to JMSReplyTo.

### **get\_type RETURN VARCHAR**

Returns type, which corresponds to JMSType.

## System Properties Methods

### **set\_userid (userid IN VARCHAR)**

Sets userid, which corresponds to JMSXUserID in JMS.

### **set\_appid (appid IN VARCHAR)**

Sets appid, which corresponds to JMSXAppID in JMS.

### **set\_groupid (groupid IN VARCHAR)**

Sets groupid, which corresponds to JMSXGroupID in JMS.

### **set\_groupseq (groupseq IN INT)**

Sets groupseq, which corresponds to JMSXGroupSeq in JMS.

### **get\_userid RETURN VARCHAR**

Returns userid, which corresponds to JMSXUserID.

### **get\_appid RETURN VARCHAR**

Returns appid, which corresponds to JMSXAppID.

### **get\_groupid RETURN VARCHAR**

Returns groupid, which corresponds to JMSXGroupID.

### **get\_groupseq RETURN INT**

Returns groupseq, which corresponds to JMSXGroupSeq.

## User Properties Methods

### **clear\_properties**

Clears all user properties. This procedure does not affect system properties.

**set\_boolean\_property (property\_name IN VARCHAR, property\_value IN BOOLEAN)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the `property_value` exists or ORA-24192 if the `property_name` is null.

**set\_byte\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_double\_property (property\_name IN VARCHAR, property\_value IN DOUBLE PRECISION)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_float\_property (property\_name IN VARCHAR, property\_value IN FLOAT)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_int\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_long\_property (property\_name IN VARCHAR, property\_value IN NUMBER)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_short\_property property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_string\_property (property\_name IN VARCHAR, property\_value IN VARCHAR)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**get\_boolean\_property (property\_name IN VARCHAR) RETURN BOOLEAN**

If the property with the corresponding `property_name` passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_byte\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_double\_property (property\_name IN VARCHAR) RETURN DOUBLE PRECISION**

If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_float\_property (property\_name IN VARCHAR) RETURN FLOAT**

If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_int\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_long\_property (property\_name IN VARCHAR) RETURN NUMBER**

If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_short\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_string\_property (property\_name IN VARCHAR) RETURN VARCHAR**

If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

## Payload Methods

**set\_text (payload IN VARCHAR2)**

Sets the payload, a `VARCHAR2` value, to an internal representation.

**set\_text (payload IN CLOB)**

Sets the payload, a `CLOB` value, to an internal representation.

**get\_text (payload OUT VARCHAR2)**

Puts the internal representation of the payload into a `VARCHAR2` variable payload.

**get\_text (payload OUT CLOB)**

Puts the internal representation of the payload into a `CLOB` variable payload.

## SYS.AQ\$\_JMS\_BYTES\_MESSAGE Type

This type is the ADT used to store a BytesMessage in an Oracle Streams AQ queue.

This section contains these topics:

- [CONSTRUCT Function](#)
- [JMS Header Methods](#)
- [System Properties Methods](#)
- [User Properties Methods](#)
- [Payload Methods](#)

### Syntax

```

TYPE AQ$_JMS_BYTES_MESSAGE AS OBJECT(
  header      aq$_jms_header,
  bytes_len   INT,
  bytes_raw   raw(2000),
  bytes_lob   blob,
  STATIC FUNCTION construct RETURN aq$_jms_bytes_message,
  MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
  MEMBER PROCEDURE set_type    (type    IN VARCHAR),
  MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
  MEMBER FUNCTION get_type     RETURN VARCHAR,
  MEMBER PROCEDURE set_userid  (userid  IN VARCHAR),
  MEMBER PROCEDURE set_appid   (appid   IN VARCHAR),
  MEMBER PROCEDURE set_groupid (groupid IN VARCHAR),
  MEMBER PROCEDURE set_groupseq (groupseq IN INT),
  MEMBER FUNCTION get_userid   RETURN VARCHAR,
  MEMBER FUNCTION get_appid    RETURN VARCHAR,
  MEMBER FUNCTION get_groupid  RETURN VARCHAR,
  MEMBER FUNCTION get_groupseq RETURN INT,
  MEMBER PROCEDURE clear_properties,
  MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
    property_value IN BOOLEAN),
  MEMBER PROCEDURE set_byte_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
    property_value IN DOUBLE PRECISION),
  MEMBER PROCEDURE set_float_property (property_name IN VARCHAR,
    property_value IN FLOAT),
  MEMBER PROCEDURE set_int_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_long_property (property_name IN VARCHAR,
    property_value IN NUMBER),
  MEMBER PROCEDURE set_short_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
    property_value IN VARCHAR),
  MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
  MEMBER FUNCTION get_byte_property (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_double_property (property_name IN VARCHAR)
    RETURN DOUBLE PRECISION,
  MEMBER FUNCTION get_float_property (property_name IN VARCHAR) RETURN FLOAT,
  MEMBER FUNCTION get_int_property (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_long_property (property_name IN VARCHAR) RETURN NUMBER,
  MEMBER FUNCTION get_short_property (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_string_property (property_name IN VARCHAR) RETURN VARCHAR,

```



```

MEMBER PROCEDURE set_bytes          (payload IN RAW),
MEMBER PROCEDURE set_bytes          (payload IN BLOB),
MEMBER PROCEDURE get_bytes          (payload OUT RAW),
MEMBER PROCEDURE get_bytes          (payload OUT BLOB),
MEMBER FUNCTION  prepare            (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset              (id IN PLS_INTEGER),
MEMBER PROCEDURE flush              (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body         (id IN PLS_INTEGER),
MEMBER PROCEDURE clean              (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION  get_mode            (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_boolean        (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION  read_byte           (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_bytes          (id IN PLS_INTEGER,
    value OUT NOCOPY BLOB, length IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_char           (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION  read_double         (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION  read_float          (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION  read_int            (id IN PLS_INTEGER) RETURN INT,
MEMBER FUNCTION  read_long           (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION  read_short          (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_unsigned_byte  (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE read_utf            (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE write_boolean       (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte          (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN RAW,
    offset IN PLS_INTEGER, length IN PLS_INTEGER),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN BLOB,
    offset IN INT, length IN INT),
MEMBER PROCEDURE write_char          (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double        (id IN PLS_INTEGER,
    value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_float         (id IN PLS_INTEGER, value IN FLOAT),
MEMBER PROCEDURE write_int           (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_long          (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short         (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_utf           (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_utf           (id IN PLS_INTEGER, value IN CLOB));

```

## CONSTRUCT Function

### **STATIC FUNCTION construct RETURN aq\$\_jms\_bytes\_message**

Creates an empty aq\$\_jms\_bytes\_message.

## JMS Header Methods

### **set\_replyto (replyto IN sys.aq\$\_agent)**

Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

### **set\_type (type IN VARCHAR)**

Sets the JMS type, which can be any text, and which corresponds to JMSType in JMS.

### **get\_replyto RETURN sys.aq\$\_agent**

Returns replyto, which corresponds to JMSReplyTo.

**get\_type RETURN VARCHAR**

Returns `type`, which corresponds to `JMSType`.

**System Properties Methods****set\_userid (userid IN VARCHAR)**

Sets `userid`, which corresponds to `JMSXUserID` in JMS.

**set\_appid (appid IN VARCHAR)**

Sets `appid`, which corresponds to `JMSXAppID` in JMS.

**set\_groupid (groupid IN VARCHAR)**

Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

**set\_groupseq (groupseq IN INT)**

Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

**get\_userid RETURN VARCHAR**

Returns `userid`, which corresponds to `JMSXUserID`.

**get\_appid RETURN VARCHAR**

Returns `appid`, which corresponds to `JMSXAppID`.

**get\_groupid RETURN VARCHAR**

Returns `groupid`, which corresponds to `JMSXGroupID`.

**get\_groupseq RETURN NUMBER**

Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

**User Properties Methods****clear\_properties**

Clears all user properties. This procedure does not affect system properties.

**set\_boolean\_property (property\_name IN VARCHAR, property\_value IN BOOLEAN)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_byte\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_double\_property (property\_name IN VARCHAR, property\_value IN DOUBLE PRECISION)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_float\_property (property\_name IN VARCHAR, property\_value IN FLOAT)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_int\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_long\_property (property\_name IN VARCHAR, property\_value IN NUMBER)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_short\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_string\_property (property\_name IN VARCHAR, property\_value IN VARCHAR)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**get\_boolean\_property (property\_name IN VARCHAR) RETURN BOOLEAN**

If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_byte\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_double\_property (property\_name IN VARCHAR) RETURN DOUBLE PRECISION**

If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_float\_property (property\_name IN VARCHAR) RETURN FLOAT**

If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_int\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_long\_property (property\_name IN VARCHAR) RETURN NUMBER**

If the property with the corresponding property name passed in exists, and if it is a `Long` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_short\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_string\_property (property\_name IN VARCHAR) RETURN VARCHAR**

If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

## Payload Methods

**set\_bytes (payload in RAW)**

Sets the payload, a `RAW` value, to an internal representation.

**set\_bytes (payload in BLOB)**

Sets the payload, a `BLOB` value, to an internal representation.

**get\_bytes (payload out RAW)**

Puts the internal representation of the payload into a `RAW` variable payload. Raises exception ORA-24190 if the length of the internal payload is more than 32767 (the maximum length of `RAW` in PL/SQL).

**get\_bytes (payload out BLOB)**

Puts the internal representation of the payload into a `BLOB` variable payload.

**prepare (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Takes the byte array stored in `aq$_jms_bytes_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function also sets the message access mode to `MESSAGE_ACCESS_READONLY`. Subsequent calls of `write_XXX` procedure raise an ORA-24196 error. Users can call the `clear_body` procedure to set the message access mode to `MESSAGE_ACCESS_READONLY`.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

**reset (id IN PLS\_INTEGER)**

Resets the starting position of the stream to the beginning and puts the bytes message in read-only mode. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**flush (id IN PLS\_INTEGER)**

Takes the static variable in Jserv and synchronizes the content back to the `aq$_jms_bytes_message`. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**clear\_body (id IN PLS\_INTEGER)**

Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session

memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

It also sets the message access mode to `MESSAGE_ACCESS_WRITEONLY`. Later calls of `read_XXX` procedure raise ORA-24196 error. Users can call the `reset` or `prepare` procedures to set the message access mode to `MESSAGE_ACCESS_READONLY`. Write-only and read-only modes affect only the payload functions of `AQ$_JMS_BYTES_MESSAGE`. They do not affect the header functions.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

#### **clean (id IN PLS\_INTEGER)**

Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

#### **clean\_all**

Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

#### **get\_mode (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Returns the current mode of this message. The return value is either `SYS.dbms_jms.plsql.MESSAGE_ACCESS_READONLY` or `SYS.dbms_jms.plsql.MESSAGE_ACCESS_WRITEONLY`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

#### **read\_boolean (id IN PLS\_INTEGER) RETURN BOOLEAN**

Reads a Boolean value from the bytes message and returns the Boolean value read. Null is returned if the end of the message stream has been reached. Parameter `id` is the operation ID. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

#### **read\_byte (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads a `BYTE` value from the bytes message and returns the `BYTE` value read. Null is returned if the end of the stream has been reached. Because there is no `BYTE` type in PL/SQL, Oracle Database uses `PLS_INTEGER` to represent a `BYTE`. Although PL/SQL users get a `PLS_INTEGER`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `write_byte` function, then there will not be an out of range error. Parameter `id` is the operation ID. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

#### **read\_bytes (id IN PLS\_INTEGER, value OUT NO COPY BLOB, length IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads length of the bytes from bytes message stream into `value` and returns the total number of bytes read. If there is no more data (because the end of the stream has been reached), then it returns -1. Raises exceptions ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_char (id IN PLS\_INTEGER) RETURN CHAR**

Reads a character value from the bytes message and returns the character value read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_double (id IN PLS\_INTEGER) RETURN DOUBLE PRECISION**

Reads a double from the bytes message and returns the character value read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_float (id IN PLS\_INTEGER) RETURN FLOAT**

Reads a float from the bytes message and returns the float read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_int (id IN PLS\_INTEGER) RETURN INT**

Reads an `INT` from the bytes message and returns the `INT` read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_long (id IN PLS\_INTEGER) RETURN NUMBER**

Reads a long from the bytes message and returns the long read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_short (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads a short value from the bytes message and returns the short value read. Null is returned if the end of the stream has been reached. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a `BYTE`. Although PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `write_short` function, then there will not be an out of range error. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_unsigned\_byte (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads an unsigned 8-bit number from the bytes message stream and returns the next byte from the bytes message stream, interpreted as an unsigned 8-bit number. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_unsigned\_short (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads an unsigned 16-bit number from the bytes message stream and returns the next two bytes from the bytes message stream, interpreted as an unsigned 16-bit integer. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_utf (id IN PLS\_INTEGER, value OUT NOCOPY CLOB)**

Reads a string that has been encoded using a UTF-8 format from the bytes message. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_boolean (id IN PLS\_INTEGER, value IN BOOLEAN)**

Writes a Boolean to the bytes message stream as a 1-byte value. The value `true` is written as the value (byte)1. The value `false` is written as the value (byte)0. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_byte (id IN PLS\_INTEGER, value IN PLS\_INTEGER)**

Writes a byte to the bytes message. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a `BYTE`. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN RAW)**

Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN BLOB)**

Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN RAW, offset IN PLS\_INTEGER, length IN PLS\_INTEGER)**

Writes a portion of a byte array to the bytes message stream. Parameter `offset` is the initial offset within the byte array. If the range [`offset`, `offset+length`] exceeds the boundary of the byte array value, then a Java `IndexOutOfBoundsException` exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN BLOB, offset IN INT, length IN INT)**

Writes a portion of a byte array to the bytes message stream. Parameter `offset` is the initial offset within the byte array. If the range [`offset`, `offset+length`] exceeds the boundary of the byte array value, then a Java `IndexOutOfBoundsException` exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_char (id IN PLS\_INTEGER, value IN CHAR)**

Writes a character value to the bytes message. If this value has multiple characters, it is the first character that is written. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_double (id IN PLS\_INTEGER, value IN DOUBLE PRECISION)**

Writes a double to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_float (id IN PLS\_INTEGER, value IN FLOAT)**

Writes a float to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_int (id IN PLS\_INTEGER, value IN PLS\_INTEGER)**

Writes an `INT` to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_long (id IN PLS\_INTEGER, value IN NUMBER)**

Writes a long to the bytes message. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_short (id IN PLS\_INTEGER, value IN PLS\_INTEGER)**

Writes a short to the bytes message as two bytes, high byte first. Because there is no short type in PL/SQL, `INT` is used to represent a short. Raises exception ORA-24193 if the parameter value exceeds the valid range, ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_utf (id IN PLS\_INTEGER, value IN VARCHAR2)**

Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_utf (id IN PLS\_INTEGER, value IN CLOB)**

Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.



## SYS.AQ\$\_JMS\_MAP\_MESSAGE Type

This type is the ADT used to store a MapMessage in an Oracle Streams AQ queue.

This section contains these topics:

- [CONSTRUCT Function](#)
- [JMS Header Methods](#)
- [System Properties Methods](#)
- [User Properties Methods](#)
- [Payload Methods](#)

### Syntax

```

TYPE aq$_jms_map_message AS object(
  header      aq$_jms_header,
  bytes_len  int,
  bytes_raw  raw(2000),
  bytes_lob  blob,
  STATIC FUNCTION construct RETURN aq$_jms_map_message,
  MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
  MEMBER PROCEDURE set_type (type IN VARCHAR),
  MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
  MEMBER FUNCTION get_type RETURN VARCHAR,
  MEMBER PROCEDURE set_userid (userid IN VARCHAR),
  MEMBER PROCEDURE set_appid (appid IN VARCHAR),
  MEMBER PROCEDURE set_groupid (groupid IN VARCHAR),
  MEMBER PROCEDURE set_groupseq (groupseq IN INT),
  MEMBER FUNCTION get_userid RETURN VARCHAR,
  MEMBER FUNCTION get_appid RETURN VARCHAR,
  MEMBER FUNCTION get_groupid RETURN VARCHAR,
  MEMBER FUNCTION get_groupseq RETURN INT,
  MEMBER PROCEDURE clear_properties,
  MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
    property_value IN BOOLEAN),
  MEMBER PROCEDURE set_byte_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
    property_value IN DOUBLE PRECISION),
  MEMBER PROCEDURE set_float_property (property_name IN VARCHAR,
    property_value IN FLOAT),
  MEMBER PROCEDURE set_int_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_long_property (property_name IN VARCHAR,
    property_value IN NUMBER),
  MEMBER PROCEDURE set_short_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
    property_value IN VARCHAR),
  MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
  MEMBER FUNCTION get_byte_property (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_double_property (property_name IN VARCHAR)
    RETURN DOUBLE PRECISION,
  MEMBER FUNCTION get_float_property (property_name IN VARCHAR) RETURN FLOAT,
  MEMBER FUNCTION get_int_property (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_long_property (property_name IN VARCHAR) RETURN NUMBER,
  MEMBER FUNCTION get_short_property (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_string_property (property_name IN VARCHAR) RETURN VARCHAR,

```

```

MEMBER PROCEDURE set_bytes (payload IN RAW),
MEMBER PROCEDURE set_bytes (payload IN BLOB),
MEMBER PROCEDURE get_bytes (payload OUT RAW),
MEMBER PROCEDURE get_bytes (payload OUT BLOB),
MEMBER FUNCTION prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE flush (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body (id IN PLS_INTEGER),
MEMBER PROCEDURE clean (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER PROCEDURE set_boolean (id IN PLS_INTEGER, name IN VARCHAR2,
value IN BOOLEAN),
MEMBER PROCEDURE set_byte (id IN PLS_INTEGER, name IN VARCHAR2,
value IN PLS_INTEGER),
MEMBER PROCEDURE set_bytes (id IN PLS_INTEGER, name IN VARCHAR2,
value IN RAW),
MEMBER PROCEDURE set_bytes (id IN PLS_INTEGER, name IN VARCHAR2,
value IN RAW, offset IN INT, length IN INT),
MEMBER PROCEDURE set_bytes (id IN PLS_INTEGER, name IN VARCHAR2,
value IN BLOB),
MEMBER PROCEDURE set_bytes (id IN PLS_INTEGER, name IN VARCHAR2,
value IN BLOB, offset IN INT, length IN INT),
MEMBER PROCEDURE set_char (id IN PLS_INTEGER, name IN VARCHAR2,
value IN CHAR),
MEMBER PROCEDURE set_double (id IN PLS_INTEGER, name IN VARCHAR2,
value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float (id IN PLS_INTEGER, name IN VARCHAR2,
value IN FLOAT),
MEMBER PROCEDURE set_int (id IN PLS_INTEGER, name IN VARCHAR2,
value IN PLS_INTEGER),
MEMBER PROCEDURE set_long (id IN PLS_INTEGER, name IN VARCHAR2,
value IN NUMBER),
MEMBER PROCEDURE set_short (id IN PLS_INTEGER, name IN VARCHAR2,
value IN PLS_INTEGER),
MEMBER PROCEDURE set_string (id IN PLS_INTEGER, name IN VARCHAR2,
value IN VARCHAR2),
MEMBER PROCEDURE set_string (id IN PLS_INTEGER, name IN VARCHAR2,
value IN CLOB),
MEMBER FUNCTION get_boolean (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN BOOLEAN,
MEMBER FUNCTION get_byte (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN PLS_INTEGER,
MEMBER PROCEDURE get_bytes (id IN PLS_INTEGER, name IN VARCHAR2,
value OUT NOCOPY BLOB),
MEMBER FUNCTION get_char (id IN PLS_INTEGER, name IN VARCHAR2) RETURN CHAR,
MEMBER FUNCTION get_double (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT,
MEMBER FUNCTION get_int (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN PLS_INTEGER,
MEMBER FUNCTION get_long (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN NUMBER,
MEMBER FUNCTION get_short (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN PLS_INTEGER,
MEMBER PROCEDURE get_string (id IN PLS_INTEGER, name IN VARCHAR2,
value OUT NOCOPY CLOB),
MEMBER FUNCTION get_names (id IN PLS_INTEGER) RETURN aq$_jms_namearray,
MEMBER FUNCTION get_names (id IN PLS_INTEGER, names OUT aq$_jms_namearray,
offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE get_object (id IN PLS_INTEGER, name IN VARCHAR2,
value OUT NOCOPY AQ$_JMS_VALUE),

```

```

MEMBER FUNCTION get_size    (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION item_exists (id IN PLS_INTEGER, name IN VARCHAR2)
RETURN BOOLEAN);

```

## CONSTRUCT Function

**STATIC FUNCTION construct RETURN aq\$\_jms\_map\_message**

Creates an empty aq\$\_jms\_map\_message object.

## JMS Header Methods

**set\_replyto (replyto IN sys.aq\$\_agent)**

Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

**set\_type (type IN VARCHAR)**

Sets the JMS type, which can be any text, and which corresponds to JMSType in JMS.

**get\_replyto RETURN sys.aq\$\_agent**

Returns replyto, which corresponds to JMSReplyTo.

**get\_type RETURN VARCHAR**

Returns type, which corresponds to JMSType.

## System Properties Methods

**set\_userid (userid IN VARCHAR)**

Sets userid, which corresponds to JMSXUserID in JMS.

**set\_appid (appid IN VARCHAR)**

Sets appid, which corresponds to JMSXAppID in JMS.

**set\_groupid (groupid IN VARCHAR)**

Sets groupid, which corresponds to JMSXGroupID in JMS.

**set\_groupseq (groupseq IN INT)**

Sets groupseq, which corresponds to JMSXGroupSeq in JMS.

**get\_userid RETURN VARCHAR**

Returns userid, which corresponds to JMSXUserID.

**get\_appid RETURN VARCHAR**

Returns appid, which corresponds to JMSXAppID.

**get\_groupid RETURN VARCHAR**

Returns groupid, which corresponds to JMSXGroupID.

**get\_groupseq RETURN NUMBER**

Returns groupseq, which corresponds to JMSXGroupSeq.

## User Properties Methods

**clear\_properties**

Clears all user properties. This procedure does not affect system properties.

**set\_boolean\_property (property\_name IN VARCHAR, property\_value IN BOOLEAN)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the `property_value` exists or ORA-24192 if the `property_name` is null.

**set\_byte\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_double\_property (property\_name IN VARCHAR, property\_value IN DOUBLE PRECISION)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_float\_property (property\_name IN VARCHAR, property\_value IN FLOAT)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_int\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_long\_property (property\_name IN VARCHAR, property\_value IN NUMBER)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_short\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_string\_property (property\_name IN VARCHAR, property\_value IN VARCHAR)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**get\_boolean\_property (property\_name IN VARCHAR) RETURN BOOLEAN**

If the property with the corresponding `property_name` passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_byte\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_double\_property (property\_name IN VARCHAR) RETURN DOUBLE PRECISION**

If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_float\_property (property\_name IN VARCHAR) RETURN FLOAT**

If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_int\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_long\_property (property\_name IN VARCHAR) RETURN NUMBER**

If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_short\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_string\_property (property\_name IN VARCHAR) RETURN VARCHAR**

If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

## Payload Methods

**set\_bytes (payload IN RAW)**

Sets the internal payload as a `RAW` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a `RAW` variable without interpreting it.

**set\_bytes (payload IN BLOB)**

Sets the internal payload as a `BLOB` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a `BLOB` variable without interpreting it.

**get\_bytes (payload OUT RAW)**

Puts the internal payload into a `RAW` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as raw bytes without interpreting it. Raises exceptions `ORA-24190` if the length of internal payload is more than 32767.

**get\_bytes (payload OUT BLOB)**

Puts the internal payload into a `BLOB` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as a `BLOB` without interpreting it.

**prepare (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Takes the byte array stored in `aq$_jms_map_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

**flush (id IN PLS\_INTEGER)**

Takes the static variable in Jserv and synchronizes the content back to `aq$_jms_map_message`. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**clear\_body (id IN PLS\_INTEGER)**

Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

**clean (id IN PLS\_INTEGER)**

Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**clean\_all**

Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

**set\_boolean (id IN PLS\_INTEGER, name IN VARCHAR2, value IN BOOLEAN)**

Sets the Boolean value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_byte (id IN PLS\_INTEGER, name IN VARCHAR2, value IN PLS\_INTEGER)**

Sets the BYTE value with the specified name in the map. Because there is no BYTE type in PL/SQL, `PLS_INTEGER` is used to represent a byte. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_bytes (id IN PLS\_INTEGER, name IN VARCHAR2, value IN RAW)**

Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_bytes (id IN PLS\_INTEGER, name IN VARCHAR2, value IN RAW, offset IN INT,**

**length IN INT)**

Sets a portion of the byte array value with the specified name in the map. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [`offset ... offset+length`] exceeds the boundary of the byte array value, then a `Java IndexOutOfBoundsException` exception is thrown in the Java stored procedure and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_bytes (id IN PLS\_INTEGER, name IN VARCHAR2, value IN BLOB)**

Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_bytes (id IN PLS\_INTEGER, name IN VARCHAR2, value IN BLOB, offset IN INT, length IN INT)**

Sets a portion of the byte array value with the specified name in the map. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [`offset ... offset+length`] exceeds the boundary of the byte array value, then a `Java IndexOutOfBoundsException` exception is thrown in the Java stored procedure, and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_char (id IN PLS\_INTEGER, name IN VARCHAR2, value IN CHAR)**

Sets the character value with the specified name in the map. If this value has multiple characters, then it is the first character that is used. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_double (id IN PLS\_INTEGER, name IN VARCHAR2, value IN DOUBLE PRECISION)**

Sets the double value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_float (id IN PLS\_INTEGER, name IN VARCHAR2, value IN FLOAT)**

This procedure is to set the float value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_int (id IN PLS\_INTEGER, name IN VARCHAR2, value IN PLS\_INTEGER)**

Sets the int value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_long (id IN PLS\_INTEGER, name IN VARCHAR2, value IN NUMBER)**

Sets the long value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_short (id IN PLS\_INTEGER, name IN VARCHAR2, value IN PLS\_INTEGER)**

Sets the short value with the specified name in the map. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a short. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_string (id IN PLS\_INTEGER, name IN VARCHAR2, value IN VARCHAR2)**

Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**set\_string (id IN PLS\_INTEGER, name IN VARCHAR2, value IN CLOB)**

Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**get\_boolean (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN BOOLEAN**

Retrieves the Boolean value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_byte (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN PLS\_INTEGER**

Retrieves the BYTE value with the specified name. If there is no item by this name, then null is returned. Because there is no BYTE type in PL/SQL, PLS\_INTEGER is used to represent a byte. Although the PL/SQL users get an PLS\_INTEGER, they are guaranteed that the value is in the Java BYTE value range. If this value is issued with a set\_byte function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_bytes (id IN PLS\_INTEGER, name IN VARCHAR2, value OUT NOCOPY BLOB)**

Retrieves the byte array value with the specified name. If there is no item by this name, then null is returned. Because the size of the array might be larger than the limit of PL/SQL RAW type, a BLOB is always returned here. The BLOB returned is a copy, which means it can be modified without affecting the message payload. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_char (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN CHAR**

Retrieves and returns the character value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

**get\_double (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN DOUBLE PRECISION**

Retrieves and returns the double value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

**get\_float (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN FLOAT**

Retrieves the float value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_int (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN PLS\_INTEGER**

Retrieves the INT value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type



of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_long (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN NUMBER**

Retrieves the long value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_short (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN PLS\_INTEGER**

Retrieves the short value with the specified name. If there is no item by this name, then null is returned. Because there is no `short` type in PL/SQL, `INT` is used to represent a `short`. Although the PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `set_short` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_string (id IN PLS\_INTEGER, name IN VARCHAR2, value OUT NOCOPY CLOB)**

Retrieves the string value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**get\_names (id IN PLS\_INTEGER) RETURN aq\$\_jms\_namearray**

Retrieves all the names within the map message and returns them in a varray. Because `aq$_jms_namearray` has a size as 1024 and each element is a `VARCHAR(200)`, this function will return an error if the size of the name array of the payload exceeds the limit. Raises exception ORA-24195 if the size of the name array or the size of a name exceeds the limit.

**get\_names (id IN PLS\_INTEGER, names OUT aq\$\_jms\_namearray, offset IN PLS\_INTEGER, length IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Retrieves a portion of the names within the map message. Because `aq$_jms_namearray` has a size as 1024 and each element is a `VARCHAR(200)`, this function will return an error if either limits are exceeded during the retrieval. (This means there is no sense to put a `length` parameter greater than 1024.) The index of the names of a map messages begins from 0. Parameter `offset` is the offset from which to start retrieving.

The function returns the number of names that have been retrieved. The names retrieved is the intersection of the interval `[offset, offset+length-1]` and interval `[0, size-1]` where `size` is the size of this map message. If the intersection is an empty set, then names will be returned as null and the function returns 0 as the number of names retrieved. If users iterate the names by retrieving in small steps, then this can be used to test that there are no more names to read from map message.

Raises exception ORA-24195 if the size of the name array or the size of a name exceed the limit, ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**get\_object (id IN PLS\_INTEGER, name IN VARCHAR2, value OUT NOCOPY AQ\$\_JMS\_VALUE)**

Returns a general value ADT `AQ$_JMS_VALUE`. If there is no item by this name, then null is returned. Users can use the `type` attribute of this ADT to interpret the data. See

the map in the AQ\$\_JMS\_VALUE ADT for the correspondence among `dbms_jms_` `plssql` package constants, Java data type and AQ\$\_JMS\_VALUE attribute. Note this member procedure might bring additional overhead compared to other `get` member procedures or functions. It is used only if the user does not know the data type of the fields within a message before hand. Otherwise it is a good idea to use a specific `get` member procedure or function. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**get\_size (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Retrieves the size of the map message. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**item\_exists (id IN PLS\_INTEGER, name IN VARCHAR2) RETURN BOOLEAN**

Indicates that an item exists in this map message by returning `TRUE`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

## SYS.AQ\$\_JMS\_STREAM\_MESSAGE Type

This type is the ADT used to store a `StreamMessage` in an Oracle Streams AQ queue.

This section contains these topics:

- [CONSTRUCT Function](#)
- [JMS Header Methods](#)
- [System Properties Methods](#)
- [User Properties Methods](#)
- [Payload Methods](#)

### Syntax

```

TYPE aq$_jms_stream_message AS object(
  header      aq$_jms_header,
  bytes_len   int,
  bytes_raw   raw(2000),
  bytes_lob   blob,
  STATIC FUNCTION construct RETURN aq$_jms_stream_message,
  MEMBER PROCEDURE set_replyto (replyto IN sys.aq$_agent),
  MEMBER PROCEDURE set_type    (type    IN VARCHAR),
  MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
  MEMBER FUNCTION get_type     RETURN VARCHAR,
  MEMBER PROCEDURE set_userid  (userid  IN VARCHAR),
  MEMBER PROCEDURE set_appid   (appid   IN VARCHAR),
  MEMBER PROCEDURE set_groupid (groupid IN VARCHAR),
  MEMBER PROCEDURE set_groupseq (groupseq IN INT),
  MEMBER FUNCTION get_userid   RETURN VARCHAR,
  MEMBER FUNCTION get_appid    RETURN VARCHAR,
  MEMBER FUNCTION get_groupid  RETURN VARCHAR,
  MEMBER FUNCTION get_groupseq RETURN INT,
  MEMBER PROCEDURE clear_properties,
  MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
    property_value IN BOOLEAN),
  MEMBER PROCEDURE set_byte_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
    property_value IN DOUBLE PRECISION),
  MEMBER PROCEDURE set_float_property (property_name IN VARCHAR,
    property_value IN FLOAT),
  MEMBER PROCEDURE set_int_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_long_property (property_name IN VARCHAR,
    property_value IN NUMBER),
  MEMBER PROCEDURE set_short_property (property_name IN VARCHAR,
    property_value IN INT),
  MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
    property_value IN VARCHAR),
  MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
  MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
    RETURN DOUBLE PRECISION,
  MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
  MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
  MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
  MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,

```

```

MEMBER PROCEDURE set_bytes          (payload IN RAW),
MEMBER PROCEDURE set_bytes          (payload IN BLOB),
MEMBER PROCEDURE get_bytes          (payload OUT RAW),
MEMBER PROCEDURE get_bytes          (payload OUT BLOB),
MEMBER FUNCTION  prepare            (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset              (id IN PLS_INTEGER),
MEMBER PROCEDURE flush              (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body         (id IN PLS_INTEGER),
MEMBER PROCEDURE clean              (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION  get_mode            (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_boolean        (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION  read_byte           (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_bytes          (id IN PLS_INTEGER) RETURN BLOB,
MEMBER PROCEDURE read_bytes          (id IN PLS_INTEGER, value OUT NOCOPY BLOB),
MEMBER FUNCTION  read_char           (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION  read_double         (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION  read_float          (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION  read_int            (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_long           (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION  read_short          (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_string         RETURN CLOB,
MEMBER PROCEDURE read_string         (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE read_object        (id IN PLS_INTEGER,
value OUT NOCOPY AQ$_JMS_VALUE),
MEMBER PROCEDURE write_boolean       (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte          (id IN PLS_INTEGER, value IN INT),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN RAW,
offset IN INT, length IN INT),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes         (id IN PLS_INTEGER, value IN BLOB,
offset IN INT, length IN INT),
MEMBER PROCEDURE write_char          (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double        (id IN PLS_INTEGER, value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_float         (id IN PLS_INTEGER, value IN FLOAT),
MEMBER PROCEDURE write_int           (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_long          (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short         (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_string        (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_string        (id IN PLS_INTEGER, value IN CLOB));

```

## CONSTRUCT Function

### **STATIC FUNCTION construct RETURN aq\$\_jms\_stream\_message**

Creates an empty aq\$\_jms\_stream\_message object.

## JMS Header Methods

### **set\_replyto (replyto IN sys.aq\$\_agent)**

Sets the replyto parameter, which corresponds to JMSReplyTo in JMS.

### **set\_type (type IN VARCHAR)**

Sets the JMS type, which can be any text, and which corresponds to JMSType in JMS.

### **get\_replyto RETURN sys.aq\$\_agent**

Returns replyto, which corresponds to JMSReplyTo.

**get\_type RETURN VARCHAR**

Returns `type`, which corresponds to `JMSType`.

**System Properties Methods****set\_userid (userid IN VARCHAR)**

Sets `userid`, which corresponds to `JMSXUserID` in JMS.

**set\_appid (appid IN VARCHAR)**

Sets `appid`, which corresponds to `JMSXAppID` in JMS.

**set\_groupid (groupid IN VARCHAR)**

Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

**set\_groupseq (groupseq IN INT)**

Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

**get\_userid RETURN VARCHAR**

Returns `userid`, which corresponds to `JMSXUserID`.

**get\_appid RETURN VARCHAR**

Returns `appid`, which corresponds to `JMSXAppID`.

**get\_groupid RETURN VARCHAR**

Returns `groupid`, which corresponds to `JMSXGroupID`.

**get\_groupseq RETURN NUMBER**

Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

**User Properties Methods****clear\_properties**

Clears all user properties. This procedure does not affect system properties.

**set\_boolean\_property (property\_name IN VARCHAR, property\_value IN BOOLEAN)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_byte\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the `property_name` exists, ORA-24192 if the `property_name` is null, or ORA-24193 if the `property_value` exceeds the valid range.

**set\_double\_property (property\_name IN VARCHAR, property\_value IN DOUBLE PRECISION)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_float\_property (property\_name IN VARCHAR, property\_value IN FLOAT)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the `property_name` exists or ORA-24192 if the `property_name` is null.

**set\_int\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_long\_property (property\_name IN VARCHAR, property\_value IN NUMBER)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**set\_short\_property (property\_name IN VARCHAR, property\_value IN INT)**

Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

**set\_string\_property (property\_name IN VARCHAR, property\_value IN VARCHAR)**

Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

**get\_boolean\_property (property\_name IN VARCHAR) RETURN BOOLEAN**

If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_byte\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_double\_property (property\_name IN VARCHAR) RETURN DOUBLE PRECISION**

If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_float\_property (property\_name IN VARCHAR) RETURN FLOAT**

If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_int\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_long\_property (property\_name IN VARCHAR) RETURN NUMBER**

If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_short\_property (property\_name IN VARCHAR) RETURN INT**

If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

**get\_string\_property (property\_name IN VARCHAR) RETURN VARCHAR**

If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

**Payload Methods****get\_bytes (payload OUT RAW)**

Puts the internal payload into a `RAW` variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as raw bytes without interpreting it. Raises exception `ORA-24190` if the length of internal payload is more than 32767.

**get\_bytes (payload OUT BLOB)**

Puts the internal payload into a `BLOB` variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as a `BLOB` variable without interpreting it.

**set\_bytes (payload IN RAW)**

Sets the internal payload as the `RAW` variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as raw bytes without interpreting it.

**set\_bytes (payload IN BLOB)**

Sets the internal payload as the `BLOB` variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a `BLOB` variable without interpreting it.

**prepare (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Takes the byte array stored in `aq$_jms_stream_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in `Jserv` session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

This function also sets the message access mode to `MESSAGE_ACCESS_READONLY`. Subsequent calls of `write_XXX` procedure raise an `ORA-24196` error. Users can call the `clear_body` procedure to set the message access mode to `MESSAGE_ACCESS_READONLY`.

This function raises `ORA-24197` if the Java stored procedure throws an exception during execution, `ORA-24198` if the operation ID is invalid, or `ORA-24199` if the Java stored procedure message store overflows.

**reset (id IN PLS\_INTEGER)**

Resets the starting position of the stream to the beginning and puts the stream message in `MESSAGE_ACCESS_READONLY` mode.

**flush (id IN PLS\_INTEGER)**

Takes the static variable in `Jserv` and synchronizes the content back to `aq$_jms_stream_message`. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**clear\_body (id IN PLS\_INTEGER)**

Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.

It also sets the message access mode to `MESSAGE_ACCESS_WRITEONLY`. Later calls of `read_XXX` procedure raise ORA-24196 error. Users can call the `reset` or `prepare` procedures to set the message access mode to `MESSAGE_ACCESS_READONLY`. Write-only and read-only modes affect only the payload functions of `AQ$_JMS_BYTES_MESSAGE`. They do not affect the header functions.

This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

**clean (id IN PLS\_INTEGER)**

Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**clean\_all**

Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

**get\_mode (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Returns the current mode of this message. The return value is either `SYS.dbms_aqjms.READ_ONLY` or `SYS.dbms_aqjms.WRITE_ONLY`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

**read\_boolean (id IN PLS\_INTEGER) RETURN BOOLEAN**

Reads and returns a Boolean value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_byte (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads and returns a byte value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `write_byte` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.



**read\_bytes (id IN PLS\_INTEGER) RETURN BLOB**

Reads and returns a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

**read\_bytes (id IN PLS\_INTEGER, value OUT NOCOPY BLOB)**

Reads a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_char (id IN PLS\_INTEGER) RETURN CHAR**

Reads and returns a character value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_double (id IN PLS\_INTEGER) RETURN DOUBLE PRECISION**

Reads and returns a double from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_float (id IN PLS\_INTEGER) RETURN FLOAT**

Reads and returns a float from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_int (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads and returns an `INT` from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_long (id IN PLS\_INTEGER) RETURN NUMBER**

Reads and returns a long from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_short (id IN PLS\_INTEGER) RETURN PLS\_INTEGER**

Reads and returns a short value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no short type in PL/SQL, `INT` is used to represent a byte. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `write_short` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type

is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_string RETURN CLOB**

Reads and returns a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

**read\_string (id IN PLS\_INTEGER, value OUT NOCOPY CLOB)**

Reads a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**read\_object (id IN PLS\_INTEGER, value OUT NOCOPY AQ\$\_JMS\_VALUE)**

Returns a general value ADT AQ\$\_JMS\_VALUE. Users can use the type attribute of this ADT to interpret the data. See [Table 246–2](#) on page 246-57 for the correspondence among `dbms_jms_plsql` package constants, Java data type and AQ\$\_JMS\_VALUE attribute. This member procedure might bring additional overhead compared to other read member procedures or functions. It is used only if the user does not know the data type of the fields within a message beforehand. Otherwise it is a good idea to use a specific read member procedure or function.

Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_boolean (id IN PLS\_INTEGER, value IN BOOLEAN)**

Writes a Boolean to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_byte (id IN PLS\_INTEGER, value IN INT)**

Writes a byte to the stream message. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN RAW)**

Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN RAW, offset IN INT, length IN INT)**

Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [`offset`, `offset+length`] exceeds the boundary of the byte array value, then a Java `IndexOutOfBoundsException` exception is thrown in the Java stored procedure. The index starts from 0.

Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN BLOB)**

Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_bytes (id IN PLS\_INTEGER, value IN BLOB, offset IN INT, length IN INT)**

Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range `[offset, offset+length]` exceeds the boundary of the byte array value, then a Java `IndexOutOfBoundsException` exception is thrown in the Java stored procedure. The index starts from 0.

Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_char (id IN PLS\_INTEGER, value IN CHAR)**

Writes a character value to the stream message. If this value has multiple characters, then it is the first character that is written. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_double (id IN PLS\_INTEGER, value IN DOUBLE PRECISION)**

Writes a double to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_float (id IN PLS\_INTEGER, value IN FLOAT)**

Writes a float to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_int (id IN PLS\_INTEGER, value IN PLS\_INTEGER)**

Writes an `INT` to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_long (id IN PLS\_INTEGER, value IN NUMBER)**

Writes a long to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_short (id IN PLS\_INTEGER, value IN PLS\_INTEGER)**

Writes a short to the stream message. Because there is no short type in PL/SQL, `INT` is used to represent a short. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_string (id IN PLS\_INTEGER, value IN VARCHAR2)**

Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

**write\_string (id IN PLS\_INTEGER, value IN CLOB)**

Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

## **SYS.AQ\$\_JMS\_OBJECT\_MESSAGE Type**

This type is the ADT used to store an `ObjectMessage` in an Oracle Streams AQ queue.

### **Syntax**

```
TYPE aq$_jms_object_message AS object(  
    header      aq$_jms_header,  
    bytes_len   int,  
    bytes_raw   raw(2000),  
    bytes_lob   blob);
```

## SYS.AQ\$\_JMS\_NAMESARRAY Type

This type represents the name array returned by the `get_names` procedure of `aq$_jms_map_message`. The maximum number of names this type can hold is 1024. The maximum length of each name is 200 characters.

### Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_NAMESARRAY AS VARRAY(1024) OF VARCHAR(100);
```

### Usage Notes

If the names array in the message payload is greater than 1024, then use the following function to retrieve the names in multiple portions:

```
MEMBER FUNCTION get_names(id IN PLS_INTEGER, names OUT aq$_jms_namearray,  
    offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER;
```

## SYS.AQ\$\_JMS\_VALUE Type

This type represents the general data returned by the `get_object` procedure of `aq$_jms_map_message` and the `read_object` procedure of `aq$_jms_stream_message`. The `type` field in this ADT is used to decide which type of data this object is really holding. [Table 246–2](#) lists the mapping between the `sys.dbms_jms_plsql` type constants, the corresponding Java type, and the data field of ADT `aq$_jms_value` which effectively holds the data.

### Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_VALUE AS object (
  type      number(2),
  num_val   number,
  char_val  char(1),
  text_val  clob,
  bytes_val blob);
```

**Table 246–2** *AQ\$\_JMS\_VALUE Type Fields and Java Fields*

Type	Java Type	aq\$_jms_value Data Field
DBMS_JMS_PLSQL.DATA_TYPE_BYTE	byte	num_val
DBMS_JMS_PLSQL.DATA_TYPE_SHORT	short	num_val
DBMS_JMS_PLSQL.DATA_TYPE_INTEGER	int	num_val
DBMS_JMS_PLSQL.DATA_TYPE_LONG	long	num_val
DBMS_JMS_PLSQL.DATA_TYPE_FLOAT	float	num_val
DBMS_JMS_PLSQL.DATA_TYPE_DOUBLE	double	num_val
DBMS_JMS_PLSQL.DATA_TYPE_BOOLEAN	boolean	num_val: 0 FALSE, 1 TRUE
DBMS_JMS_PLSQL.DATA_TYPE_CHARACTER	char	char_val
DBMS_JMS_PLSQL.DATA_TYPE_STRING	java.lang.String	text_val
DBMS_JMS_PLSQL.DATA_TYPE_BYTES	byte[]	bytes_val

## SYS.AQ\$\_JMS\_EXCEPTION Type

This type represents a Java exception thrown on the Java stored procedure side. The `id` field is reserved for future use. The `exp_name` stores the Java exception name, the `err_msg` field stores the Java exception error message, and the `stack` field stores the stack trace of the Java exception.

### Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_EXCEPTION AS OBJECT (  
    id          number, -- Reserved and not used. Right now always return 0.  
    exp_name    varchar(200),  
    err_msg     varchar(500),  
    stack       varchar(4000));
```



---

---

## Expression Filter Types

---

---

**Note:** This functionality is deprecated with Oracle Database Release 11.2 and obsoleted with Release 12.1. For details regarding obsolescence, see My Oracle Support Note ID 1244535.1

---

---

The Expression Filter feature provides a set of predefined types and public synonyms for these types. Most of these types are used for configuring index parameters with the Expression Filter procedural APIs. The `EXF$TABLE_ALIAS` type is used to support expressions defined on one or more database tables.

**See Also:** *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information.

This chapter contains the following topics:

- [Using Expression Filter Types](#)
- [Summary of Expression Filter Types](#)

---

## Using Expression Filter Types

This section contains topics that relate to using the Expression Filter Types.

- [Security Model](#)

## Security Model

The Oracle Database installation runs the `catexf.sql` script to load the `DBMS_EXPFIL` package and create the required Expression Filter schema objects in the `EXFSYS` schema.

## Summary of Expression Filter Types

Table 247–1 describes the Expression Filter object types.

All the values and names passed to the types defined in this chapter are not case sensitive. To preserve the case, enclose the values with double quotation marks.

**Table 247–1 Expression Filter Object Types**

<b>Object Type Name</b>	<b>Description</b>
<a href="#">EXF\$ATTRIBUTE</a>	Specifies the stored and indexed attributes for the Expression Filter indexes
<a href="#">EXF\$ATTRIBUTE_LIST</a>	Specifies a list of stored and indexed attributes when configuring index parameters
<a href="#">EXF\$INDEXOPER</a>	Specifies a list of common operators in predicates with a stored or an indexed attribute
<a href="#">EXF\$TABLE_ALIAS</a>	Indicates a special form of elementary attribute used to manage expressions defined on one or more database tables
<a href="#">EXF\$TEXT</a>	Associates preferences to a text attribute in an attribute set or an event structure
<a href="#">EXF\$XPATH_TAG</a>	Configures an XML element or an XML attribute for indexing a set of XPath predicates
<a href="#">EXF\$XPATH_TAGS</a>	Specifies a list of XML tags when configuring the Expression Filter index parameters

## EXF\$ATTRIBUTE

The EXF\$ATTRIBUTE type is used to handle stored and indexed attributes for the Expression Filter indexes.

### Syntax

```
CREATE or REPLACE TYPE EXF$ATTRIBUTE AS OBJECT attr_name    VARCHAR2(350),
                                                attr_oper    EXF$INDEXOPER,
                                                attr_indexed VARCHAR2(5);
```

### Attributes

**Table 247–2 EXF\$ATTRIBUTE Attributes**

Attribute	Description
attr_name	The arithmetic expression that constitutes the stored or indexed attribute
attr_oper	The list of common operators in the predicates with the attribute. Default value: EXF\$INDEXOPER('all')
attr_indexed	TRUE if the attribute is indexed, else FALSE. Default value: FALSE.

### Usage Notes

- The EXF\$ATTRIBUTE type is used to specify the stored and indexed attributes for an Expression Filter index using the DBMS\_EXPFIL.DEFAULT\_INDEX\_PARAMETERS procedure. When values for attr\_oper and attr\_indexed fields are omitted during EXF\$ATTRIBUTE instantiation, it is considered a stored attribute with a default value for common operators (EXF\$INDEXOPER('all')).

### Examples

A stored attribute with no preference on the list of common operators is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)')
```

An indexed attribute is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)',
              attr_indexed => 'TRUE')
```

An indexed attribute with a list of common operators is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)',
              attr_oper => exf$indexoper('=', '<', '>', '>=', '<='),
              attr_indexed => 'TRUE')
```

## EXF\$ATTRIBUTE\_LIST

The EXF\$ATTRIBUTE\_LIST type specifies a list of stored and indexed attributes while configuring the index parameters.

### Syntax

```
CREATE or REPLACE TYPE EXF$ATTRIBUTE_LIST as VARRAY(490) of exf$attribute;
```

### Attributes

None.

### Usage Notes

- Also see the [DEFAULT\\_INDEX\\_PARAMETERS Procedure](#) for more information

### Examples

A list of stored and indexed attributes can be represented as follows:

```
exf$attribute_list (  
    exf$attribute (attr_name => 'Model',  
                  attr_oper => exf$indexoper('='),  
                  attr_indexed => 'TRUE'),  
    exf$attribute (attr_name => 'Price',  
                  attr_oper => exf$indexoper('all'),  
                  attr_indexed => 'TRUE'),  
    exf$attribute (attr_name => 'HorsePower(Model, Year)',  
                  attr_oper => exf$indexoper('=', '<', '>', '>=', '<='),  
                  attr_indexed => 'FALSE')  
)
```

## EXF\$INDEXOPER

The EXF\$INDEXOPER type specifies the list of common operators in predicates with a stored or an indexed attribute.

### Syntax

```
CREATE or REPLACE TYPE EXFSYS.EXF$INDEXOPER as VARRAY(20) of VARCHAR2(15);
```

The values for the EXF\$INDEXOPER array are expected to be from the list in the following table:

Value	Predicate Description
=	Equality predicates
>	Greater than predicates
<	Less than predicates
>=	Greater than or equal to predicates
<=	Less than or equal to predicates
!= or <> or ^=	Not equal to predicates
IS NULL	IS NULL predicates
IS NOT NULL	IS NOT NULL predicates
ALL	All the operators listed in this table starting with the equality predicate through the IS NOT NULL predicate
NVL	Predicates with NVL (equality) operator
LIKE	Predicates with LIKE operator
BETWEEN	BETWEEN predicates

### Attributes

None.

### Usage Notes

- A value of ALL for one of the EXF\$INDEXOPER items implies that all the simple operators (=,>,<,>=,<=,!=, IS NULL, IS NOT NULL) are common in the predicates with an attribute. This value can be used along with one or more complex operators (NVL, LIKE and BETWEEN).
- A predicate with a BETWEEN operator is treated as two predicates with binary operators, one with '>=' operator and another with '<=' operator. By default, only one of these operators is indexed, and the other operator is evaluated by value substitution. However, if predicates with the BETWEEN operator are common for an attribute (stored or indexed), both the binary operators resulting from the BETWEEN operator can be indexed by specifying BETWEEN in the EXF\$INDEXOPER VARRAY. However, because this uses additional space in the predicate table, this operator should be used only when majority of predicates with an attribute use the BETWEEN operator.
- When the LIKE operator is chosen as one of the common operators for an attribute, LIKE predicates on that attributes are indexed. Indexing a LIKE operator is beneficial only if the VARCHAR2 constant on the right-hand side of the

predicate does not lead with a wild-card character. For example, indexing a LIKE operator will filter the following predicates efficiently:

```
company LIKE 'General%'  
company LIKE 'Proctor%'
```

But, the following predicates are evaluated as sparse predicates in the last stage:

```
company LIKE '%Electric'  
company LIKE "%Gamble'
```

## Examples

An attribute with a list of common operators is represented as follows:

```
exf$attribute (attr_name => 'HorsePower(Model, Year)',  
              attr_oper => exf$indexoper('=', '<', '>', '>=', '<=', 'between'),  
              attr_indexed => 'TRUE')
```



## EXF\$TABLE\_ALIAS

A EXF\$TABLE\_ALIAS type is a special form of elementary attribute that can be included in the attribute set. These attributes manage expressions defined on one or more database tables.

### Syntax

```
CREATE or REPLACE TYPE EXF$TABLE_ALIAS AS OBJECT table_name VARCHAR2(70);
```

### Attributes

**Table 247–3 EXF\$TABLE\_ALIAS Attribute**

Attribute	Description
table_name	Name of the table with a possible schema extension

### Usage Notes

- The concept of a table alias attribute is captured in the Expression Filter dictionary and the corresponding attribute in the attribute set's object type is created with a VARCHAR2 data type. (Also see Oracle Database Rules Manager and Expression Filter Developer's Guide and [ADD\\_ELEMENTARY\\_ATTRIBUTE Procedures](#).)

### Examples

For a set of expressions defined on database tables, the corresponding table alias attributes are configured as follows:

```
BEGIN
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
    attr_set => 'HRAttrSet',
    attr_name => 'EMP',
    tab_alias => exf$table_alias('SCOTT.EMP'));
  DBMS_EXPFIL.ADD_ELEMENTARY_ATTRIBUTE (
    attr_set => 'HRAttrSet',
    attr_name => 'DEPT',
    tab_alias => exf$table_alias('DEPT'));
END;
/
```

The Expression column using the previous attribute set can store expressions of form EMP.JOB = 'Clerk' and EMP.NAME = 'Joe', where JOB and NAME are the names of the columns in the SCOTT.EMP table.

## EXF\$TEXT

A `EXF$TEXT` type associates preferences to a text attribute in an attribute set or an event structure.

### Syntax

```
CREATE or REPLACE TYPE EXFSYS.EXF$TEXT AS OBJECT(preferences VARCHAR2(1000));
```

### Attributes

**Table 247–4** *EXF\$TEXT Attribute*

Attribute	Description
preferences	Text preference specification, such as LEXER, CLASSIFIER, and WORDLIST

### Usage Notes

- The `EXF$TEXT` attribute is used to specify the preferences for a text attribute at the time of creation. The preferences specified through the instance of `EXF$TEXT` type are used in creation of the `CTXRULE` index for the text predicates. The syntax for the text preference specification is similar to the `PARAMETERS` clause specified for a `CTXRULE` `Indextype`. [See `CREATE INDEX` syntax for `CTXRULE` `Indextype` in *Oracle Text Reference*, Release 10g Release 2.]

### Examples

A text predicate with a `LEXER` and `WORDLIST` preferences can be created using the following instance of `EXF$TEXT` object as follows:

```
EXF$TEXT ('LEXER insrpt_lexer WORDLIST insrpt_wordlist');
```

## EXF\$XPATH\_TAG

The EXF\$XPATH\_TAG type configures an XML element or an XML attribute for indexing a set of XPath predicates.

### Syntax

```
CREATE or REPLACE TYPE EXF$XPATH_TAG AS OBJECT tag_name    VARCHAR2(350),
                                              tag_indexed  VARCHAR2(5),
                                              tag_type     VARCHAR2(30);
```

### Attributes

**Table 247–5 EXF\$XPATH\_TAG Attributes**

Attribute	Description
tag_name	Name of the XML element or attribute. The name for an XML attribute is formatted as: <ElementName>@<AttributeName>. Optionally, the element name can be prefixed with its namespace URL as in <Namespace URL>:<ElementName>@<AttributeName>.
tag_indexed	TRUE if XML tag is indexed; otherwise FALSE. Default: TRUE if the tag is a positional filter. FALSE if the tag is a value filter.
tag_type	Datatype for the value in the case of value filter. NULL for positional filters.

### Usage Notes

- EXF\$XPATH\_TAG type configures an XML element or an attribute as a positional or a value filter for an Expression Filter index (see the section on index tuning for XPath predicates in *Oracle Database Rules Manager and Expression Filter Developer's Guide*). An instance of the EXF\$XPATH\_TAG type with NULL value for tag\_type configures the XML tag as a positional filter. In the current release, the only other possible values for the tag\_type attribute are strings (CHAR or VARCHAR) and such tags are configured as value filters. By default, all positional filters are indexed and the value filters are not indexed. This behavior can be overridden by setting a TRUE or FALSE value for the tag\_indexed attribute accordingly.

### Examples

An XML element can be configured as a positional filter and be indexed using the following instance of the EXF\$XPATH\_TAG type.

```
exf$xpath_tag(tag_name    => 'stereo',      --- XML element
              tag_indexed => 'TRUE',      --- indexed predicate group
              tag_type     => null)      --- positional filter
```

An XML attribute can be configured as a value filter and be indexed using the following type instance.

```
exf$xpath_tag(tag_name    => 'stereo@make', --- XML attribute
              tag_indexed => 'TRUE',      --- indexed predicate group
              tag_type     => 'VARCHAR(15)') --- value filter
```

The following commands configure the two filters shown previously using the namespace URL for the corresponding elements.

```
exf$xpath_tag(tag_name => 'http://www.auto.com/car.xsd:stereo',
```

```
        tag_indexed => 'TRUE', --- indexed predicate group
        tag_type => null)      --- positional filter

exf$xpath_tag(tag_name => 'http://www.auto.com/car.xsd:stereo@make'
        tag_indexed => 'TRUE', --- indexed predicate group
        tag_type => 'VARCHAR(15)') --- value filter
```

## EXF\$XPATH\_TAGS

The EXF\$XPATH\_TAGS type specifies a list of XML tags while configuring the Expression Filter index parameters.

### Syntax

```
CREATE or REPLACE TYPE EXF$XPATH_TAGS as VARRAY(490) of EXF$XPATH_TAG;
```

### Attributes

None.

### Usage Notes

- EXF\$XPATH\_TAGS type specifies a list of XML tags while configuring the Expression Filter index parameters. (See [DEFAULT\\_INDEX\\_PARAMETERS Procedure](#).)

### Examples

A list of XML tags configured as positional and value filters can be represented as follows:

```
exf$xpath_tags(
  exf$xpath_tag(tag_name    => 'stereo@make', --- XML attribute
                tag_indexed => 'TRUE',
                tag_type    => 'VARCHAR(15)'), --- value filter
  exf$xpath_tag(tag_name    => 'stereo',      --- XML element
                tag_indexed => 'FALSE',
                tag_type    => null),         --- positional filter
  exf$xpath_tag(tag_name    => 'memory',      --- XML element
                tag_indexed => 'TRUE',
                tag_type    => 'VARCHAR(10)') --- value filter
)
```



---

---

## Logical Change Record TYPEs

This chapter describes the logical change record (LCR) types.

This chapter contains these topics:

- [Using Logical Change Record Types](#)
  - Overview
  - Security Model
- [Summary of Logical Change Record Types](#)
- [Common Subprograms for LCR\\$\\_DDL\\_RECORD and LCR\\$\\_ROW\\_RECORD](#)

---

## Using Logical Change Record Types

This section contains topics that relate to using the logical change record (LCR) types.

- [Overview](#)
- [Security Model](#)



## Overview

In Oracle Streams, logical change records (LCRs) are message payloads that contain information about changes to a database. These changes can include changes to the data, which are data manipulation language (DML) changes, and changes to database objects, which are data definition language (DDL) changes.

When you use Oracle Streams, the capture process captures changes in the form of LCRs and enqueues them into a queue. These LCRs can be propagated from a queue in one database to a queue in another database. Finally, the apply process can apply LCRs at a destination database. You also have the option of creating, enqueueing, and dequeuing LCRs manually.

**See Also:** *Oracle Streams Concepts and Administration* for more information about LCRs

## Security Model

PUBLIC is granted EXECUTE privilege on the types described in this chapter.

---

## Summary of Logical Change Record Types

**Table 248–1 Logical Change Record (LCR) Types**

Type	Description
<a href="#">LCR\$_DDL_RECORD</a> Type on page 248-6	Represents a data definition language (DDL) change to a database object
<a href="#">LCR\$_ROW_RECORD</a> Type on page 248-15	Represents a data manipulation language (DML) change to a database object
<a href="#">LCR\$_ROW_LIST</a> Type on page 248-47	Identifies a list of column values for a row in a table
<a href="#">LCR\$_ROW_UNIT</a> Type on page 248-48	Identifies the value for a column in a row

These logical change record (LCR) types can be used with the following Oracle-supplied PL/SQL packages:

- DBMS\_APPLY\_ADM
- DBMS\_AQ
- DBMS\_AQADM
- DBMS\_CAPTURE\_ADM
- DBMS\_PROPAGATION\_ADM
- DBMS\_RULE
- DBMS\_RULE\_ADM
- DBMS\_STREAMS
- DBMS\_STREAMS\_ADM
- DBMS\_TRANSFORM

## LCR\$\_DDL\_RECORD Type

This type represents a data definition language (DDL) change to a database object.

If you create or modify a DDL logical change record (DDL LCR), then make sure the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.

This section contains information about the constructor for DDL LCRs and information about the member subprograms for this type:

- [LCR\\$\\_DDL\\_RECORD Constructor](#)
- [Summary of LCR\\$\\_DDL\\_RECORD Subprograms](#), which also include the subprograms described in "Common Subprograms for LCR\$\_DDL\_RECORD and LCR\$\_ROW\_RECORD" on page 248-36

---



---

### Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
  - The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.
- 
- 

### LCR\$\_DDL\_RECORD Constructor

Creates a `SYS.LCR$_DDL_RECORD` object with the specified information.

```

STATIC FUNCTION CONSTRUCT(
    source_database_name IN VARCHAR2,
    command_type         IN VARCHAR2,
    object_owner         IN VARCHAR2,
    object_name          IN VARCHAR2,
    object_type          IN VARCHAR2,
    ddl_text             IN CLOB,
    logon_user           IN VARCHAR2,
    current_schema       IN VARCHAR2,
    base_table_owner     IN VARCHAR2,
    base_table_name      IN VARCHAR2,
    tag                  IN RAW          DEFAULT NULL,
    transaction_id       IN VARCHAR2   DEFAULT NULL,
    scn                  IN NUMBER     DEFAULT NULL,
    position             IN RAW          DEFAULT NULL,
    edition_name         IN VARCHAR2   DEFAULT NULL)
RETURN SYS.LCR$_DDL_RECORD;
```

## LCR\$\_DDL\_RECORD Constructor Function Parameters

**Table 248–2 Constructor Function Parameters for LCR\$\_DDL\_RECORD**

Parameter	Description
source_database_name	<p>The database where the DDL statement occurred</p> <p>If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is EXAMPLE.COM, then the function specifies DBS1.EXAMPLE.COM automatically. Set this parameter to a non-NULL value.</p>
command_type	<p>The type of command executed in the DDL statement</p> <p>Set this parameter to a non-NULL value.</p> <p><b>See Also:</b> The "SQL Command Codes" table in the <i>Oracle Call Interface Programmer's Guide</i> for a complete list of command types</p> <p>The following command types <i>are not supported</i> in DDL LCRs:</p> <pre>ALTER MATERIALIZED VIEW ALTER MATERIALIZED VIEW LOG ALTER SUMMARY CREATE SCHEMA CREATE MATERIALIZED VIEW CREATE MATERIALIZED VIEW LOG CREATE SUMMARY DROP MATERIALIZED VIEW DROP MATERIALIZED VIEW LOG DROP SUMMARY RENAME</pre> <p>The snapshot equivalents of the materialized view command types are also not supported.</p>
object_owner	The user who owns the object on which the DDL statement was executed
object_name	The database object on which the DDL statement was executed
object_type	<p>The type of object on which the DDL statement was executed</p> <p>The following are valid object types:</p> <pre>CLUSTER FUNCTION INDEX LINK OUTLINE PACKAGE PACKAGE BODY PROCEDURE SEQUENCE SYNONYM TABLE TRIGGER TYPE USER VIEW</pre> <p>LINK represents a database link.</p> <p>NULL is also a valid object type. Specify NULL for all object types not listed. The GET_OBJECT_TYPE member procedure returns NULL for object types not listed.</p>

**Table 248–2 (Cont.) Constructor Function Parameters for LCR\$\_DDL\_RECORD**

Parameter	Description
ddl_text	The text of the DDL statement Set this parameter to a non-NULL value.
logon_user	The user whose session executed the DDL statement
current_schema	The schema that is used if no schema is specified explicitly for the modified database objects in <code>ddl_text</code> If a schema is specified in <code>ddl_text</code> that differs from the one specified for <code>current_schema</code> , then the function uses the schema specified in <code>ddl_text</code> . Set this parameter to a non-NULL value.
base_table_owner	If the DDL statement is a table-related DDL (such as <code>CREATE TABLE</code> and <code>ALTER TABLE</code> ), or if the DDL statement involves a table (such as creating a trigger on a table), then <code>base_table_owner</code> specifies the owner of the table involved. Otherwise, <code>base_table_owner</code> is NULL.
base_table_name	If the DDL statement is a table-related DDL (such as <code>CREATE TABLE</code> and <code>ALTER TABLE</code> ), or if the DDL statement involves a table (such as creating a trigger on a table), then <code>base_table_name</code> specifies the name of the table involved. Otherwise, <code>base_table_name</code> is NULL.
tag	A binary tag that enables tracking of the LCR For example, this tag can be used to determine the original source database of the DDL statement if apply forwarding is used. <b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i>
transaction_id	The identifier of the transaction
scn	The SCN at the time when the change record for a captured LCR was written to the redo log The SCN value is meaningless for a user-created LCR.
position	The position of the LCR LCR position is commonly used in XStream configurations. Using XStream requires purchasing a license for the Oracle GoldenGate product. <b>See Also:</b> <i>Oracle Database XStream Guide</i>
edition_name	The name of the edition in which the DDL statement was executed

## Summary of LCR\$\_DDL\_RECORD Subprograms

**Table 248–3 LCR\$\_DDL\_RECORD Type Subprograms**

Subprogram	Description
<a href="#">EXECUTE Member Procedure</a> on page 248-9	Executes the LCR under the security domain of the current user
<a href="#">GET_BASE_TABLE_NAME Member Function</a> on page 248-9	Gets the base (dependent) table name
<a href="#">GET_BASE_TABLE_OWNER Member Function</a> on page 248-10	Gets the base (dependent) table owner
<a href="#">GET_CURRENT_SCHEMA Member Function</a> on page 248-10	Gets the default schema (user) name

**Table 248-3 (Cont.) LCR\$\_DDL\_RECORD Type Subprograms**

Subprogram	Description
<a href="#">GET_DDL_TEXT Member Procedure</a> on page 248-10	Gets the DDL text in a CLOB
<a href="#">GET_EDITION_NAME Member Function</a> on page 248-11	Gets the name of the edition in which the DDL statement was executed
<a href="#">GET_LOGON_USER Member Function</a> on page 248-11	Gets the logon user name
<a href="#">GET_OBJECT_TYPE Member Function</a> on page 248-11	Gets the type of the object involved for the DDL
<a href="#">SET_BASE_TABLE_NAME Member Procedure</a> on page 248-11	Sets the base (dependent) table name
<a href="#">SET_BASE_TABLE_OWNER Member Procedure</a> on page 248-11	Sets the base (dependent) table owner
<a href="#">SET_CURRENT_SCHEMA Member Procedure</a> on page 248-12	Sets the default schema (user) name
<a href="#">SET_DDL_TEXT Member Procedure</a> on page 248-12	Sets the DDL text
<a href="#">SET_EDITION_NAME Member Procedure</a> on page 248-12	Sets the name of the edition in which the DDL statement was executed
<a href="#">SET_LOGON_USER Member Procedure</a> on page 248-13	Sets the logon user name
<a href="#">SET_OBJECT_TYPE Member Procedure</a> on page 248-13	Sets the object type
Common Subprograms	See " <a href="#">Common Subprograms for LCR\$_DDL_RECORD and LCR\$_ROW_RECORD</a> " on page 248-36 for a list of subprograms common to the SYS.LCR\$_ROW_RECORD and SYS.LCR\$_DDL_RECORD types

## EXECUTE Member Procedure

Executes the DDL LCR under the security domain of the current user. Apply handlers are not run when the LCR is applied using this procedure.

---



---

**Note:** The EXECUTE member procedure can be invoked only in an apply handler for an Oracle Streams apply process.

---



---

## Syntax

```
MEMBER PROCEDURE EXECUTE;
```

## GET\_BASE\_TABLE\_NAME Member Function

Gets the base (dependent) table name.

## Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_NAME()
RETURN VARCHAR2;
```

### GET\_BASE\_TABLE\_OWNER Member Function

Gets the base (dependent) table owner.

#### Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_OWNER()  
RETURN VARCHAR2;
```

### GET\_CURRENT\_SCHEMA Member Function

Gets the current schema name.

#### Syntax

```
MEMBER FUNCTION GET_CURRENT_SCHEMA()  
RETURN VARCHAR2;
```

### GET\_DDL\_TEXT Member Procedure

Gets the DDL text in a CLOB.

For example, the following PL/SQL code uses this procedure to get the DDL text in a DDL LCR:

```
CREATE OR REPLACE PROCEDURE ddl_in_lcr (ddl_lcr in SYS.LCR$_DDL_RECORD)  
IS  
    ddl_text    CLOB;  
BEGIN  
    DBMS_OUTPUT.PUT_LINE( ' -----' );  
    DBMS_OUTPUT.PUT_LINE( '  Displaying DDL text in a DDL LCR: ' );  
    DBMS_OUTPUT.PUT_LINE( ' -----' );  
    DBMS_LOB.CREATETEMPORARY(ddl_text, true);  
    ddl_lcr.GET_DDL_TEXT(ddl_text);  
    DBMS_OUTPUT.PUT_LINE('DDL text: ' || ddl_text);  
    DBMS_LOB.FREETEMPORARY(ddl_text);  
END;  
/
```

---

---

**Note:** GET\_DDL\_TEXT is a member procedure and not a member function to make it easier for you to manage the space used by the CLOB. Notice that the previous example creates temporary space for the CLOB and then frees the temporary space when it is no longer needed.

---

---

#### Syntax

```
MEMBER FUNCTION GET_DDL_TEXT(  
    ddl_text IN/OUT CLOB);
```

#### Parameter

**Table 248–4** GET\_DDL\_TEXT Procedure Parameter

Parameter	Description
ddl_text	The DDL text in the DDL LCR



**GET\_EDITION\_NAME Member Function**

Gets the name of the edition in which the DDL statement was executed.

**See Also:** *Oracle Database Advanced Application Developer's Guide*

**Syntax**

```
MEMBER FUNCTION GET_EDITION_NAME()
RETURN VARCHAR2;
```

**GET\_LOGON\_USER Member Function**

Gets the logon user name.

**Syntax**

```
MEMBER FUNCTION GET_LOGON_USER()
RETURN VARCHAR2;
```

**GET\_OBJECT\_TYPE Member Function**

Gets the type of the object involved for the DDL.

**Syntax**

```
MEMBER FUNCTION GET_OBJECT_TYPE()
RETURN VARCHAR2;
```

**SET\_BASE\_TABLE\_NAME Member Procedure**

Sets the base (dependent) table name.

**Syntax**

```
MEMBER PROCEDURE SET_BASE_TABLE_NAME(
    base_table_name IN VARCHAR2);
```

**Parameter**

**Table 248–5 SET\_BASE\_TABLE\_NAME Procedure Parameter**

Parameter	Description
base_table_name	The name of the base table

**SET\_BASE\_TABLE\_OWNER Member Procedure**

Sets the base (dependent) table owner.

**Syntax**

```
MEMBER PROCEDURE SET_BASE_TABLE_OWNER(
    base_table_owner IN VARCHAR2);
```

## Parameter

**Table 248–6 SET\_BASE\_TABLE\_OWNER Procedure Parameter**

Parameter	Description
base_table_owner	The name of the base owner

## SET\_CURRENT\_SCHEMA Member Procedure

Sets the default schema (user) name.

## Syntax

```
MEMBER PROCEDURE SET_CURRENT_SCHEMA(  
    current_schema IN VARCHAR2);
```

## Parameter

**Table 248–7 SET\_CURRENT\_SCHEMA Procedure Parameter**

Parameter	Description
current_schema	The name of the schema to set as the current schema Set this parameter to a non-NULL value.

## SET\_DDL\_TEXT Member Procedure

Sets the DDL text.

## Syntax

```
MEMBER PROCEDURE SET_DDL_TEXT(  
    ddl_text IN CLOB);
```

## Parameter

**Table 248–8 SET\_DDL\_TEXT Procedure Parameter**

Parameter	Description
ddl_text	The DDL text Set this parameter to a non-NULL value.

## SET\_EDITION\_NAME Member Procedure

Sets the name of the edition in which the DDL statement was executed.

**See Also:** *Oracle Database Advanced Application Developer's Guide*

## Syntax

```
MEMBER PROCEDURE SET_EDITION_NAME(  
    edition_name IN VARCHAR2);
```

**Parameter****Table 248–9** *SET\_EDITION\_NAME Procedure Parameter*

Parameter	Description
edition_name	Name of the edition

**SET\_LOGON\_USER Member Procedure**

Sets the logon user name.

**Syntax**

```
MEMBER PROCEDURE SET_LOGON_USER(
    logon_user IN VARCHAR2);
```

**Parameter****Table 248–10** *SET\_LOGON\_USER Procedure Parameter*

Parameter	Description
logon_user	The name of the schema to set as the logon user

**SET\_OBJECT\_TYPE Member Procedure**

Sets the object type.

**Syntax**

```
MEMBER PROCEDURE SET_OBJECT_TYPE(
    object_type IN VARCHAR2);
```

## Parameter

**Table 248–11** *SET\_OBJECT\_TYPE Procedure Parameter*

<b>Parameter</b>	<b>Description</b>
<code>object_type</code>	<p>The object type</p> <p>The following are valid object types:</p> <ul style="list-style-type: none"><li>CLUSTER</li><li>FUNCTION</li><li>INDEX</li><li>LINK</li><li>OUTLINE</li><li>PACKAGE</li><li>PACKAGE BODY</li><li>PROCEDURE</li><li>SEQUENCE</li><li>SYNONYM</li><li>TABLE</li><li>TRIGGER</li><li>TYPE</li><li>USER</li><li>VIEW</li></ul> <p>LINK represents a database link.</p> <p>NULL is also a valid object type. Specify NULL for all object types not listed. The <code>GET_OBJECT_TYPE</code> member procedure returns NULL for object types not listed.</p>

## LCR\$\_ROW\_RECORD Type

This type represents a data manipulation language (DML) change to a row in a table. This type uses the LCR\$\_ROW\_LIST type.

If you create or modify a row logical change record (row LCR), then make sure the `command_type` attribute is consistent with the presence or absence of old column values and the presence or absence of new column values.

This section contains information about the constructor for DDL LCRs and information about the member subprograms for this type:

- [LCR\\$\\_ROW\\_RECORD Constructor](#)
- [Summary of LCR\\$\\_ROW\\_RECORD Subprograms](#), which also include the subprograms described in [Common Subprograms for LCR\\$\\_DDL\\_RECORD and LCR\\$\\_ROW\\_RECORD](#) on page 248-36

---



---

### Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
  - The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.
- 
- 

**See Also:** [LCR\\$\\_ROW\\_LIST Type](#) on page 248-47

## LCR\$\_ROW\_RECORD Constructor

Creates a SYS.LCR\$\_ROW\_RECORD object with the specified information.

```

STATIC FUNCTION CONSTRUCT(
  source_database_name IN VARCHAR2,
  command_type         IN VARCHAR2,
  object_owner         IN VARCHAR2,
  object_name          IN VARCHAR2,
  tag                  IN RAW          DEFAULT NULL,
  transaction_id       IN VARCHAR2    DEFAULT NULL,
  scn                  IN NUMBER      DEFAULT NULL,
  old_values           IN SYS.LCR$_ROW_LIST DEFAULT NULL,
  new_values           IN SYS.LCR$_ROW_LIST DEFAULT NULL,
  position             IN RAW          DEFAULT NULL,
  statement            IN VARCHAR2    DEFAULT NULL,
  bind_variables       IN SYS.LCR$_ROW_LIST DEFAULT NULL,
  bind_by_position     IN VARCHAR2    DEFAULT 'N')
RETURN SYS.LCR$_ROW_RECORD;
```

## LCR\$\_ROW\_RECORD Constructor Function Parameters

**Table 248–12 Constructor Function Parameters for LCR\$\_ROW\_RECORD**

Parameter	Description
source_database_name	<p>The database where the row change occurred</p> <p>If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is EXAMPLE.COM, then the function specifies DBS1.EXAMPLE.COM automatically. Set this parameter to a non-NULL value.</p>
command_type	<p>The type of command executed in the DML statement</p> <p>Set this parameter to a non-NULL value.</p> <p>Valid values are the following:</p> <p>INSERT UPDATE DELETE LOB ERASE LOB WRITE LOB TRIM</p> <p>If INSERT, then ensure that the LCR has a new_values collection that is not empty and an empty or NULL old_values collection.</p> <p>If UPDATE, then ensure that the LCR has a new_values collection that is not empty and an old_values collection that is not empty.</p> <p>If DELETE, then ensure that the LCR has a NULL or empty new_values collection and an old_values collection that is not empty.</p> <p>If LOB ERASE, LOB WRITE, or LOB TRIM, then ensure that the LCR has a new_values collection that is not empty and an empty or NULL old_values collection.</p>
object_owner	<p>The user who owns the table on which the row change occurred</p> <p>Set this parameter to a non-NULL value.</p>
object_name	<p>The table on which the DML statement was executed</p> <p>Set this parameter to a non-NULL value.</p>
tag	<p>A binary tag that enables tracking of the LCR</p> <p>For example, this tag can be used to determine the original source database of the DML change when apply forwarding is used.</p> <p><b>See Also:</b> <i>Oracle Streams Replication Administrator's Guide</i></p>
transaction_id	The identifier of the transaction
scn	<p>The SCN at the time when the change record was written to the redo log</p> <p>The SCN value is meaningless for a user-created LCR.</p>
old_values	<p>The column values for the row before the DML change</p> <p>If the DML statement is an UPDATE or a DELETE statement, then this parameter contains the values of columns in the row before the DML statement. If the DML statement is an INSERT statement, then there are no old values.</p>

**Table 248–12 (Cont.) Constructor Function Parameters for LCR\$\_ROW\_RECORD**

Parameter	Description
new_values	The column values for the row after the DML change  If the DML statement is an UPDATE or an INSERT statement, then this parameter contains the values of columns in the row after the DML statement. If the DML statement is a DELETE statement, then there are no new values.  If the LCR reflects a LOB operation, then this parameter contains the supplementally logged columns and any relevant LOB information.
position	The position of the LCR  LCR position is commonly used in XStream configurations. Using XStream requires purchasing a license for the Oracle GoldenGate product.  <b>See Also:</b> <i>Oracle Database XStream Guide</i>
statement	This parameter is reserved for internal use only.
bind_variables	This parameter is reserved for internal use only.
bind_by_position	This parameter is reserved for internal use only.

---

**Note:** Starting with Oracle Database 11g Release 2 (11.2.0.2), this constructor function includes the following new parameters: `statement`, `bind_variables`, and `bind_by_position`.

---

## Summary of LCR\$\_ROW\_RECORD Subprograms

**Table 248–13 LCR\$\_ROW\_RECORD Type Subprograms**

Subprogram	Description
<a href="#">ADD_COLUMN Member Procedure</a> on page 248-18	Adds the value as old or new, depending on the value type specified, for the column
<a href="#">CONVERT_LONG_TO_LOB_CHUNK Member Procedure</a> on page 248-20	Converts LONG data in a row LCR into fixed width CLOB, or converts LONG RAW data in a row LCR into a BLOB
<a href="#">DELETE_COLUMN Member Procedure</a> on page 248-20	Deletes the old value, the new value, or both, for the specified column, depending on the value type specified
<a href="#">EXECUTE Member Procedure</a> on page 248-20	Executes the LCR under the security domain of the current user
<a href="#">GET_LOB_INFORMATION Member Function</a> on page 248-22	Gets the LOB information for the column
<a href="#">GET_LOB_OFFSET Member Function</a> on page 248-22	Gets the LOB offset for the specified column
<a href="#">GET_LOB_OPERATION_SIZE Member Function</a> on page 248-23	Gets the operation size for the LOB column
<a href="#">GET_LONG_INFORMATION Member Function</a> on page 248-23	Gets the LONG information for the column
<a href="#">GET_ROW_TEXT Member Procedure</a> on page 248-24	Gets the SQL statement for the change that is encapsulated in the LCR

**Table 248–13 (Cont.) LCR\$\_ROW\_RECORD Type Subprograms**

Subprogram	Description
<a href="#">GET_VALUE Member Function</a> on page 248-25	Gets the old or new value for the specified column, depending on the value type specified
<a href="#">GET_VALUES Member Function</a> on page 248-26	Gets a list of old or new values, depending on the value type specified
<a href="#">GET_WHERE_CLAUSE Member Procedure</a> on page 248-26	Gets a WHERE clause for the change that is encapsulated in the row LCR
<a href="#">GET_XML_INFORMATION Member Function</a> on page 248-28	Gets the XML information for the specified column
<a href="#">IS_STATEMENT_LCR Member Function</a> on page 248-28	Reserved for internal use only
<a href="#">KEEP_COLUMNS Member Procedure</a> on page 248-28	Keeps a list of columns a row LCR
<a href="#">RENAME_COLUMN Member Procedure</a> on page 248-29	Renames a column in an LCR
<a href="#">SET_LOB_INFORMATION Member Procedure</a> on page 248-29	Sets LOB information for the column
<a href="#">SET_LOB_OFFSET Member Procedure</a> on page 248-30	Sets the LOB offset for the specified column
<a href="#">SET_LOB_OPERATION_SIZE Member Procedure</a> on page 248-31	Sets the operation size for the LOB column
<a href="#">SET_ROW_TEXT Member Procedure</a> on page 248-31	Reserved for internal use only
<a href="#">SET_VALUE Member Procedure</a> on page 248-31	Overwrites the value of the specified column
<a href="#">SET_VALUES Member Procedure</a> on page 248-33	Replaces the existing old or new values for the LCR, depending on the value type specified
<a href="#">SET_XML_INFORMATION Member Procedure</a> on page 248-35	Sets the XML information for the column
Common Subprograms	See <a href="#">Common Subprograms for LCR\$_DDL_RECORD and LCR\$_ROW_RECORD</a> on page 248-36 for a list of subprograms common to the SYS.LCR\$_ROW_RECORD and SYS.LCR\$_DDL_RECORD types

### ADD\_COLUMN Member Procedure

Adds the value as old or new, depending on the value type specified, for the column. An error is raised if a value of the same type already exists for the column.

---

**Note:** To set a column value that already exists, run [SET\\_VALUE](#).

---

**See Also:** [SET\\_VALUE Member Procedure](#) on page 248-31

### Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler and the handler is using LOB assembly (the `assemble_lob` parameter is set to `TRUE` for the handler), you use this member procedure in the handler procedure to add a LOB column to a row LCR. If `assemble_lob` is set to `FALSE` for



the handler, then you cannot use this member procedure to add a LOB column to a row LCR.

To use a DML or error handler to add a LOB column, specify the LOB locator for the `column_value` parameter in the member procedure. The `ADD_COLUMN` member procedure verifies that an `ANYDATA` encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated LOB locator.
- An attempt is made to add an LOB column that is set incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

---



---

**Note:**

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
  - When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure to add a LOB column.
  - When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure to add a `LONG` or `LONG RAW` column.
- 
- 

## Syntax

```
MEMBER PROCEDURE ADD_COLUMN(
    value_type    IN VARCHAR2,
    column_name   IN VARCHAR2,
    column_value  IN ANYDATA);
```

## Parameters

**Table 248–14** *ADD\_COLUMN Procedure Parameters*

Parameter	Description
<code>value_type</code>	The type of value to add for the column Specify <code>old</code> to add the old value of the column. Specify <code>new</code> to add the new value of the column.
<code>column_name</code>	The column name This name is not validated. An error can be raised during application of the LCRs if an invalid name is specified.
<code>column_value</code>	The value of the column If <code>NULL</code> , then this procedure raises an error. If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then a LOB locator can be specified. A <code>NULL</code> column value can be specified by encapsulating the <code>NULL</code> value in an <code>ANYDATA</code> wrapper.

## CONVERT\_LONG\_TO\_LOB\_CHUNK Member Procedure

Converts LONG data in a row LCR into a CLOB, or converts LONG RAW data in a row LCR into a BLOB.

This procedure can change the operation code from LONG WRITE to LOB WRITE for the row LCR.

This member procedure can be used in rule-based transformations.

The following restrictions apply to this member procedure:

- This member procedure cannot be used in apply handlers.
- LONG data can be sent as a part of a row LCR with one of the following operation codes: INSERT, UPDATE, or LONG\_WRITE. Because LONG data can be sent in multiple pieces, make sure that this method is invoked on either none or all LONG pieces.
- LOB to LONG conversion is not supported.
- A row LCR on which this procedure is executed must have been created by a capture process. That is, this procedure does not support persistent row LCRs.

**See Also:** *Oracle Streams Replication Administrator's Guide*

### Syntax

```
MEMBER PROCEDURE CONVERT_LONG_TO_LOB_CHUNK;
```

## DELETE\_COLUMN Member Procedure

Deletes the old value, the new value, or both, for the specified column, depending on the value type specified.

### Syntax

```
MEMBER PROCEDURE DELETE_COLUMN(
    column_name IN VARCHAR2,
    value_type  IN VARCHAR2 DEFAULT '*');
```

### Parameters

**Table 248–15** *DELETE\_COLUMN Procedure Parameters*

Parameter	Description
column_name	The column name An error is raised if the column does not exist in the LCR.
value_type	The type of value to delete for the column Specify old to delete the old value of the column. Specify new to delete the new value of the column. If * is specified, then the procedure deletes both the old and new values.

## EXECUTE Member Procedure

Executes the row LCR under the security domain of the current user. Any apply handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

This member procedure can be run on a row LCR under any of the following conditions:

- The LCR is being processed by an apply handler.
- The LCR is in a queue and was last enqueued by a mechanism other than an Oracle Streams capture process, such as an Oracle Streams apply process or an application.
- The LCR has been constructed using the `LCR$_ROW_RECORD` constructor function but has not been enqueued.
- The LCR is in the error queue.

---

**Note:** Do not run this member procedure in a custom rule-based transformation on a row LCR. Doing so could execute the row LCR outside of its transactional context.

---

### Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler, and the handler is using LOB assembly (the `assemble_lob` parameter is set to `TRUE` for the handler), this member procedure executes the assembled row LCR. An assembled row LCR represents a LOB value with a LOB locator or `NULL`.

If `assemble_lob` is set to `FALSE` for the handler, then this member procedure executes the nonassembled row LCRs. Nonassembled row LCRs represent LOB values with `VARCHAR2` and `RAW` data types. These nonassembled row LCRs might have been modified by the handler.

An error is raised under the following conditions:

- A DML or error handler configured with `assemble_lob` set to `FALSE` attempts to execute a row LCR that contains a LOB locator.
- A DML or error handler configured with `assemble_lob` set to `TRUE` attempts to execute a row LCR that contains one or more LOB values represented with `VARCHAR2` or `RAW` data types.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

## Syntax

```
MEMBER PROCEDURE EXECUTE(
    conflict_resolution IN BOOLEAN);
```

## Parameters

**Table 248–16 EXECUTE Procedure Parameters**

Parameter	Description
<code>conflict_resolution</code>	<p>If <code>TRUE</code>, then any conflict resolution defined for the table using the <code>SET_UPDATE_CONFLICT_HANDLER</code> procedure in the <code>DBMS_APPLY_ADM</code> package is used to resolve conflicts resulting from the execution of the LCR.</p> <p>If <code>FALSE</code>, then conflict resolution is not used.</p> <p>An error is raised if this parameter is not specified or is set to <code>NULL</code>.</p>

## GET\_LOB\_INFORMATION Member Function

Gets the LOB information for the column.

The return value can be one of the following:

```
DBMS_LCR.NOT_A_LOB          CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LOB         CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LOB       CONSTANT NUMBER := 3;
DBMS_LCR.EMPTY_LOB        CONSTANT NUMBER := 4;
DBMS_LCR.LOB_CHUNK        CONSTANT NUMBER := 5;
DBMS_LCR.LAST_LOB_CHUNK   CONSTANT NUMBER := 6;
```

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the `use_old` parameter is a convenient way to get the value of the columns.

### Syntax

```
MEMBER FUNCTION GET_LOB_INFORMATION(
  value_type  IN VARCHAR2,
  column_name IN VARCHAR2,
  use_old     IN VARCHAR2 DEFAULT 'Y')
RETURN NUMBER;
```

### Parameters

**Table 248–17 GET\_LOB\_INFORMATION Function Parameters**

Parameter	Description
<code>value_type</code>	The type of value to return for the column, either <code>old</code> or <code>new</code>
<code>column_name</code>	The name of the column
<code>use_old</code>	<p>If Y and <code>value_type</code> is <code>new</code>, and no new value exists, then the function returns the corresponding old value. If N and <code>value_type</code> is <code>new</code>, then the function does not return the old value if no new value exists.</p> <p>If <code>value_type</code> is <code>old</code> or if the <code>command_type</code> of the row LCR is not UPDATE, then the function ignores the value of the <code>use_old</code> parameter.</p> <p>NULL is not a valid specification for the <code>use_old</code> parameter.</p>

## GET\_LOB\_OFFSET Member Function

Gets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB. That is, the information is `DBMS_LCR.LAST_LOB_CHUNK` or `DBMS_LCR.LOB_CHUNK`
- The command type is LOB ERASE or LOB WRITE

Otherwise, returns NULL.

### Syntax

```
GET_LOB_OFFSET(
  value_type  IN VARCHAR2,
```

```

column_name IN VARCHAR2)
RETURN NUMBER;

```

## Parameters

**Table 248–18** *GET\_LOB\_OFFSET Procedure Parameters*

Parameter	Description
value_type	The type of value to return for the column Currently, only new can be specified.
column_name	The name of the LOB column

## GET\_LOB\_OPERATION\_SIZE Member Function

Gets the operation size for the LOB column in the number of characters for CLOB columns and the number of bytes for BLOB columns. Returns a non-NULL value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB
- The command type is LOB ERASE or LOB TRIM
- The information is DBMS\_LCR.LAST\_LOB\_CHUNK

Otherwise, returns NULL.

## Syntax

```

MEMBER FUNCTION GET_LOB_OPERATION_SIZE(
  value_type IN VARCHAR2,
  column_name IN VARCHAR2)
RETURN NUMBER,

```

## Parameters

**Table 248–19** *GET\_LOB\_OPERATION\_SIZE Function Parameters*

Parameter	Description
value_type	The type of value to return for the column Currently, only new can be specified.
column_name	The name of the LOB column

## GET\_LONG\_INFORMATION Member Function

Gets the LONG information for the column.

The return value can be one of the following:

```

DBMS_LCR.NOT_A_LONG          CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LONG          CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LONG        CONSTANT NUMBER := 3;
DBMS_LCR.LONG_CHUNK         CONSTANT NUMBER := 4;
DBMS_LCR.LAST_LONG_CHUNK    CONSTANT NUMBER := 5;

```

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the `use_old` parameter is a convenient way to get the value of the columns.

## Syntax

```
MEMBER FUNCTION GET_LONG_INFORMATION(
    value_type IN VARCHAR2,
    column_name IN VARCHAR2,
    use_old IN VARCHAR2 DEFAULT 'Y')
RETURN NUMBER;
```

## Parameters

**Table 248–20 GET\_LONG\_INFORMATION Function Parameters**

Parameter	Description
<code>value_type</code>	The type of value to return for the column, either <code>old</code> or <code>new</code>
<code>column_name</code>	The name of the column
<code>use_old</code>	<p>If Y and <code>value_type</code> is new, and no new value exists, then the function returns the corresponding old value. If N and <code>value_type</code> is new, then the function does not return the old value if no new value exists.</p> <p>If <code>value_type</code> is old or if the <code>command_type</code> of the row LCR is not UPDATE, then the function ignores the value of the <code>use_old</code> parameter.</p> <p>NULL is not a valid specification for the <code>use_old</code> parameter.</p>

## GET\_ROW\_TEXT Member Procedure

Gets the SQL statement for the change that is encapsulated in the row LCR. This method performs SQL generation in PL/SQL.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

## Syntax

The following procedure returns the SQL statement in a CLOB data type.

```
MEMBER PROCEDURE GET_ROW_TEXT(
    row_text IN/OUT CLOB);
```

The following procedure returns the SQL statement with bind variables in a CLOB data type.

```
MEMBER PROCEDURE GET_ROW_TEXT(
    row_text IN/OUT CLOB,
    variable_list IN/OUT LCR$_ROW_LIST,
    bind_var_syntax IN VARCHAR2 DEFAULT ':');
```

**See Also:** ["LCR\\$\\_ROW\\_LIST Type"](#) on page 248-47

## Parameters

**Table 248–21 GET\_ROW\_TEXT Procedure Parameters**

Parameter	Description
row_text	The SQL statement for the change that is encapsulated in the LCR
variable_list	The values for the bind variables in the order of the bind variables
bind_var_syntax	The syntax for the bind variables One of the following values is valid: <ul style="list-style-type: none"> <li>■ Specify :, the default, for bind values to be in the form :1, :2, and so on.</li> <li>■ Specify ? for bind values to be in the form ?.</li> </ul>

## GET\_VALUE Member Function

Gets the old or new value for the specified column, depending on the value type specified.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use\_old parameter is a convenient way to get the value of a column.

## Syntax

```
MEMBER FUNCTION GET_VALUE(
    value_type IN VARCHAR2,
    column_name IN VARCHAR2,
    use_old IN VARCHAR2 DEFAULT 'Y')
RETURN ANYDATA;
```

## Parameters

**Table 248–22 GET\_VALUE Function Parameters**

Parameter	Description
value_type	The type of value to return for the column Specify old to get the old value for the column. Specify new to get the new value for the column.
column_name	The column name If the column is present and has a NULL value, then the function returns an ANYDATA instance containing a NULL value. If the column value is absent, then the function returns a NULL.
use_old	If Y and value_type is new, and no new value exists, then the function returns the corresponding old value. If N and value_type is new, then the function returns NULL if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the function ignores the value of the use_old parameter. NULL is not a valid specification for the use_old parameter.

## GET\_VALUES Member Function

Gets a list of old or new values, depending on the value type specified.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the `use_old` parameter is a convenient way to get the values of all columns.

### Syntax

```
MEMBER FUNCTION GET_VALUES (
    value_type IN VARCHAR2,
    use_old    IN VARCHAR2 DEFAULT 'Y')
RETURN SYS.LCR$_ROW_LIST;
```

### Parameters

**Table 248–23 GET\_VALUES Function Parameters**

Parameter	Description
<code>value_type</code>	The type of values to return  Specify <code>old</code> to return a list of old values. Specify <code>new</code> to return a list of new values.
<code>use_old</code>	If Y and <code>value_type</code> is <code>new</code> , then the function returns a list of all new values in the LCR. If a new value does not exist in the list, then the function returns the corresponding old value. Therefore, the returned list contains all existing new values and the old values where there are no new values.  If N and <code>value_type</code> is <code>new</code> , then the function returns a list of all new values in the LCR without returning any old values.  If <code>value_type</code> is <code>old</code> or if the <code>command_type</code> of the row LCR is not UPDATE, then the function ignores the value of the <code>use_old</code> parameter.  NULL is not a valid specification for the <code>use_old</code> parameter.

## GET\_WHERE\_CLAUSE Member Procedure

Gets a WHERE clause for the change that is encapsulated in the row LCR.

Use the WHERE clause returned by `GET_WHERE_CLAUSE` instead of using the ROWID, because the ROWID is not ANSI compatible. The generated WHERE clause might not match the WHERE clause in the original DML operation.

The ROWID of an INSERT statement is the ROWID of the new row created by the INSERT. The WHERE clause generated for an INSERT operation identifies the new row. Therefore, the generated WHERE clause includes all of the new values inserted.

For example, consider the following insert into the `hr.departments` table:

```
INSERT INTO hr.departments (
    department_id, department_name, manager_id, location_id)
VALUES (10, 'HR', 20, 40);
```

The generated WHERE clause represents the row with the values 10, 'HR', 20, and 40. Hence, the generated WHERE clause is the following:

```
WHERE "DEPARTMENT_ID" = 10 AND "DEPARTMENT_NAME" = 'HR' AND
      "MANAGER_ID" = 20 AND "LOCATION_ID" = 40
```

The ROWID of an UPDATE statement is the ROWID of the row that was updated. The WHERE clause generated for an UPDATE operation identifies the row after the UPDATE



executes. The generated WHERE clause is based on the old and new values of the UPDATE.

For example, consider the following update to the `hr.departments` table:

```
UPDATE hr.departments SET department_name='Management'
WHERE department_name='Administration' AND location_id = 20 AND
      manager_id = 30 AND department_id = 10;
```

The values of the row after the UPDATE are 10, 'Management', 30, and 20. Hence, the generated WHERE clause to identify the row is the following:

```
WHERE "DEPARTMENT_ID" = 10 AND "DEPARTMENT_NAME" = 'MANAGEMENT' AND
      "MANAGER_ID" = 30 AND "LOCATION_ID" = 20
```

Notice that the new value is used for "DEPARTMENT\_NAME", because the new value is the value of the column after the UPDATE. For the rest of the columns, the old values are used.

The ROWID of a DELETE operation is the row that existed before it was deleted. The generated WHERE clause consists of all the old column values present in the DELETE operation.

LOB columns do not appear in generated WHERE clauses. The generated WHERE clause is not affected by the presence of LOB columns in the LCR.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

## Syntax

The following procedure returns the WHERE clause of a SQL statement in a CLOB data type.

```
MEMBER PROCEDURE GET_WHERE_CLAUSE(
  where_clause IN/OUT CLOB);
```

The following procedure returns the WHERE clause of a SQL statement with bind variables in a CLOB data type.

```
MEMBER PROCEDURE GET_WHERE_CLAUSE(
  where_clause IN/OUT CLOB,
  variable_list IN/OUT LCR$_ROW_LIST,
  bind_var_syntax IN VARCHAR2 DEFAULT ':');
```

### See Also:

- [LCR\\$\\_ROW\\_LIST Type](#) on page 248-47
- *Oracle Streams Concepts and Administration*

## Parameters

**Table 248–24 GET\_WHERE\_CLAUSE Procedure Parameters**

Parameter	Description
<code>where_clause</code>	The WHERE clause of the SQL statement for the change that is encapsulated in the LCR
<code>variable_list</code>	The values for the bind variables in the order of the bind variables

**Table 248–24 (Cont.) GET\_WHERE\_CLAUSE Procedure Parameters**

Parameter	Description
bind_var_syntax	<p>The syntax for the bind variables</p> <p>One of the following values is valid:</p> <ul style="list-style-type: none"> <li>■ Specify :, the default, for bind values to be in the form :1, :2, and so on.</li> <li>■ Specify ? for bind values to be in the form ?.</li> </ul>

## GET\_XML\_INFORMATION Member Function

Gets the XML information for the specified column.

The return value can be one of the following:

```
DBMS_LCR.NOT_XML    CONSTANT NUMBER := 1;
DBMS_LCR.XML_DOC   CONSTANT NUMBER := 2;
DBMS_LCR.XML_DIFF  CONSTANT NUMBER := 3;
```

DBMS\_LCR.NOT\_XML indicates that the column is not an XMLType column. DBMS\_LCR.XML\_DOC indicates that the column contains an XML document. DBMS\_LCR.XML\_DIFF indicates that the column contains differences between old and new XML documents for an update operation.

Returns NULL if the specified column does not exist.

## Syntax

```
MEMBER FUNCTION GET_XML_INFORMATION(
    column_name IN VARCHAR2)
RETURN NUMBER;
```

## Parameter

**Table 248–25 GET\_XML\_INFORMATION Function Parameter**

Parameter	Description
column_name	The column name

## IS\_STATEMENT\_LCR Member Function

This function is reserved for internal use only.

---



---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---



---

## KEEP\_COLUMNS Member Procedure

This procedure keeps a list of columns in a row LCR. The procedure deletes columns that are not in the list from the row LCR.

## Syntax

```
MEMBER PROCEDURE KEEP_COLUMNS(
    column_list IN VARCHAR2,
    value_type  IN VARCHAR2 DEFAULT '*');
```

## Parameters

**Table 248–26** *KEEP\_COLUMNS Procedure Parameters*

Parameter	Description
column_list	The names of the columns kept for the row LCR Specify a comma-delimited list of type VARCHAR2. This procedure removes columns that are not in the list from the current row LCR.
value_type	The type of value for which to keep the columns Specify <code>old</code> to keep the old values of the columns. An error is raised if the old values do not exist in the LCR. Specify <code>new</code> to keep the new values of the columns. An error is raised if the new values do not exist in the LCR. If <code>*</code> is specified, then the procedure keeps both the old and the new columns.

## RENAME\_COLUMN Member Procedure

Renames a column in a row LCR.

## Syntax

```
MEMBER PROCEDURE RENAME_COLUMN(
    from_column_name IN VARCHAR2,
    to_column_name   IN VARCHAR2,
    value_type       IN VARCHAR2 DEFAULT '*');
```

## Parameters

**Table 248–27** *RENAME\_COLUMN Procedure Parameters*

Parameter	Description
from_column_name	The existing column name
to_column_name	The new column name An error is raised if a column with the specified name already exists.
value_type	The type of value for which to rename the column Specify <code>old</code> to rename the old value of the column. An error is raised if the old value does not exist in the LCR. Specify <code>new</code> to rename the new value of the column. An error is raised if the new value does not exist in the LCR. If <code>*</code> is specified, then the procedure renames the column names for both old and new value. The procedure raises an error if either column value does not exist in the LCR.

## SET\_LOB\_INFORMATION Member Procedure

Sets LOB information for the column.

---

**Note:** When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

---

## Syntax

```
MEMBER PROCEDURE SET_LOB_INFORMATION(
  value_type      IN  VARCHAR2,
  column_name     IN  VARCHAR2,
  lob_information IN  NUMBER);
```

## Parameters

**Table 248–28** *SET\_LOB\_INFORMATION Procedure Parameters*

Parameter	Description												
value_type	The type of value to set for the column, either old or new Specify old only if lob_information is set to DBMS_LCR.NOT_A_LOB.												
column_name	The name of the column. An exception is raised if the column value does not exist. You might need to set this parameter for non-LOB columns.												
lob_information	Specify one of the following values: <table border="0" style="margin-left: 20px;"> <tr> <td>DBMS_LCR.NOT_A_LOB</td> <td>CONSTANT NUMBER := 1;</td> </tr> <tr> <td>DBMS_LCR.NULL_LOB</td> <td>CONSTANT NUMBER := 2;</td> </tr> <tr> <td>DBMS_LCR.INLINE_LOB</td> <td>CONSTANT NUMBER := 3;</td> </tr> <tr> <td>DBMS_LCR.EMPTY_LOB</td> <td>CONSTANT NUMBER := 4;</td> </tr> <tr> <td>DBMS_LCR.LOB_CHUNK</td> <td>CONSTANT NUMBER := 5;</td> </tr> <tr> <td>DBMS_LCR.LAST_LOB_CHUNK</td> <td>CONSTANT NUMBER := 6;</td> </tr> </table>	DBMS_LCR.NOT_A_LOB	CONSTANT NUMBER := 1;	DBMS_LCR.NULL_LOB	CONSTANT NUMBER := 2;	DBMS_LCR.INLINE_LOB	CONSTANT NUMBER := 3;	DBMS_LCR.EMPTY_LOB	CONSTANT NUMBER := 4;	DBMS_LCR.LOB_CHUNK	CONSTANT NUMBER := 5;	DBMS_LCR.LAST_LOB_CHUNK	CONSTANT NUMBER := 6;
DBMS_LCR.NOT_A_LOB	CONSTANT NUMBER := 1;												
DBMS_LCR.NULL_LOB	CONSTANT NUMBER := 2;												
DBMS_LCR.INLINE_LOB	CONSTANT NUMBER := 3;												
DBMS_LCR.EMPTY_LOB	CONSTANT NUMBER := 4;												
DBMS_LCR.LOB_CHUNK	CONSTANT NUMBER := 5;												
DBMS_LCR.LAST_LOB_CHUNK	CONSTANT NUMBER := 6;												

## SET\_LOB\_OFFSET Member Procedure

Sets the LOB offset for the specified column in the number of characters for CLOB columns and the number of bytes for BLOB columns.

---



---

**Note:** When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

---



---

## Syntax

```
MEMBER PROCEDURE SET_LOB_OFFSET(
  value_type      IN  VARCHAR2,
  column_name     IN  VARCHAR2,
  lob_offset      IN  NUMBER);
```

## Parameters

**Table 248–29** *SET\_LOB\_OFFSET Procedure Parameters*

Parameter	Description
value_type	The type of value to set for the column Currently, only new can be specified.
column_name	The column name An error is raised if the column value does not exist in the LCR.

**Table 248–29 (Cont.) SET\_LOB\_OFFSET Procedure Parameters**

Parameter	Description
lob_offset	The LOB offset number  Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.

**SET\_LOB\_OPERATION\_SIZE Member Procedure**

Sets the operation size for the LOB column in the number of characters for CLOB columns and bytes for BLOB columns.

---

**Note:** When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

---

**Syntax**

```
MEMBER PROCEDURE SET_LOB_OPERATION_SIZE(
  value_type      IN  VARCHAR2,
  column_name     IN  VARCHAR2,
  lob_operation_size IN NUMBER);
```

**Parameters****Table 248–30 SET\_LOB\_OPERATION\_SIZE Procedure Parameters**

Parameter	Description
value_type	The type of value to set for the column  Currently, only new can be specified.
column_name	The name of the LOB column  An exception is raised if the column value does not exist in the LCR.
lob_operation_size	If lob_information for the LOB is or will be DBMS_LOB.LAST_LOB_CHUNK, then this parameter can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE.  Otherwise, set to NULL.

**SET\_ROW\_TEXT Member Procedure**

This procedure is reserved for internal use only.

---

**Note:** This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

---

**SET\_VALUE Member Procedure**

Overwrites the old or new value of the specified column.

One reason to overwrite an old value for a column is to resolve an error that resulted from a conflict.

---

---

**Note:** To add a column to a row LCR, run `ADD_COLUMN`.

---

---

**See Also:** [ADD\\_COLUMN Member Procedure](#) on page 248-18

### Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler, and the handler is using LOB assembly (the `assemble_lob` parameter is set to `TRUE` for the handler), you can use this member procedure in the handler procedure on a LOB column in a row LCR. If `assemble_lob` is set to `FALSE` for the handler, then you cannot use this member procedure on a LOB column.

To use a DML or error handler to set the value of a LOB column, specify the LOB locator for the `column_value` parameter in the member procedure. The `SET_VALUE` member procedure verifies that an `ANYDATA` encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

---

---

**Note:**

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
  - When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on a LOB column.
  - When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure on a `LONG` or `LONG RAW` column.
- 
- 

### Considerations for XMLType Columns

When processing a row LCR with `XMLType` columns with a procedure DML handler or error handler, any `XMLType` columns and LOB columns in the LCR are always assembled using LOB assembly. You can use this member procedure in the handler procedure on a row LCR that contains one or more `XMLType` columns.

To use a DML or error handler to set the value an `XMLType` column, specify the `XMLType` for the `column_value` parameter. The `SET_VALUE` member procedure verifies that an `ANYDATA` encapsulated `XMLType` is processed with a DML or error handler. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated `XMLType`.
- An attempt is made to set a `XMLType` column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the `XMLType` column is represented by the original (nonassembled) row LCRs.

---



---

**Note:**

- Database compatibility must be 11.1.0 or higher to process row LCRs with `XMLType` columns.
  - When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on `XMLType` columns.
- 
- 

## Syntax

```
MEMBER PROCEDURE SET_VALUE(
  value_type  IN  VARCHAR2,
  column_name IN  VARCHAR2,
  column_value IN ANYDATA);
```

## Parameters

**Table 248–31** *SET\_VALUE Procedure Parameters*

Parameter	Description
<code>value_type</code>	The type of value to set  Specify <code>old</code> to set the old value of the column. Specify <code>new</code> to set the new value of the column.
<code>column_name</code>	The column name  An error is raised if the specified <code>column_value</code> does not exist in the LCR for the specified <code>column_type</code> .
<code>column_value</code>	The new value of the column  If <code>NULL</code> is specified, then this procedure raises an error. To set the value to <code>NULL</code> , encapsulate the <code>NULL</code> in an <code>ANYDATA</code> instance.  If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then specify a LOB locator for LOB columns.

## SET\_VALUES Member Procedure

Replaces all old values or all new values for the LCR, depending on the value type specified.

### Considerations for LOB Columns

You can use this procedure when processing a row LCR with LOB columns with a procedure DML handler or error handler. If the handler is using LOB assembly (the `assemble_lob` parameter is set to `TRUE` for the handler), then you can use this member procedure in the handler procedure. If `assemble_lob` is set to `FALSE` for the handler, then you cannot use this member procedure on a row LCR.

To use a DML or error handler to set the value of one or more LOB columns in a row LCR, specify a LOB locator for each LOB column in the `value_list` parameter. The `SET_VALUES` member procedure verifies that an `ANYDATA` encapsulated LOB locator

is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB columns are represented by the original (nonassembled) row LCRs.

---

---

**Note:**

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
  - When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on LOB columns.
  - When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure on `LONG` or `LONG RAW` columns.
- 
- 

**Considerations for XMLType Columns**

When processing a row LCR with `XMLType` columns with a procedure DML handler or error handler, any `XMLType` and LOB columns in the LCR are always assembled using LOB assembly. You can use this member procedure in the handler procedure on a row LCR that contains one or more `XMLType` columns.

To use a DML or error handler to set the value of one or more `XMLType` columns in a row LCR, specify an `XMLType` for each `XMLType` column in the `value_list` parameter. The `SET_VALUES` member procedure verifies that an `ANYDATA` encapsulated `XMLType` is processed with a DML or error handler. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated `XMLType`.
- An attempt is made to set a `XMLType` incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the `XMLType` columns are represented by the original (nonassembled) row LCRs.

---

---

**Note:**

- Database compatibility must be 11.1.0 or higher to process row LCRs with `XMLType` columns.
  - When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on `XMLType` columns.
- 
- 

**Syntax**

```
MEMBER PROCEDURE SET_VALUES(
```



```
value_type IN VARCHAR2,
value_list IN SYS.LCR$_ROW_LIST);
```

## Parameters

**Table 248–32 SET\_VALUES Procedure Parameters**

Parameter	Description
value_type	The type of values to replace Specify old to replace the old values. Specify new to replace the new values.
value_list	List of values to replace the existing list Use a NULL or an empty list to remove all values. If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then specify one or more LOB locators for LOB columns.

## SET\_XML\_INFORMATION Member Procedure

Sets the XML information for the column.

## Syntax

```
MEMBER PROCEDURE SET_XML_INFORMATION(
    column_name IN VARCHAR2,
    xml_information IN NUMBER);
```

## Parameters

**Table 248–33 SET\_XML\_INFORMATION Procedure Parameters**

Parameter	Description
column_name	The name of the column An exception is raised if the column value does not exist in the LCR.
xml_information	Specify one of the following values: <pre>DBMS_LCR.NOT_XML    CONSTANT NUMBER := 1; DBMS_LCR.XML_DOC    CONSTANT NUMBER := 2; DBMS_LCR.XML_DIFF   CONSTANT NUMBER := 3;</pre> DBMS_LCR.NOT_XML indicates that the column is not an XMLType column. DBMS_LCR.XML_DOC indicates that the column contains an XML document. DBMS_LCR.XML_DIFF indicates that the column contains differences between old and new XML documents for an update operation.

## Common Subprograms for LCR\$\_DDL\_RECORD and LCR\$\_ROW\_RECORD

The following functions and procedures are common to both the LCR\$\_DDL\_RECORD and LCR\$\_ROW\_RECORD type.

**See Also:** For descriptions of the subprograms for these types that are exclusive to each type:

- ["Summary of LCR\\$\\_DDL\\_RECORD Subprograms"](#) on page 248-8
- ["Summary of LCR\\$\\_ROW\\_RECORD Subprograms"](#) on page 248-17

**Table 248–34 Summary of Common Subprograms for DDL and Row LCR Types**

Subprogram	Description
<a href="#">GET_COMMAND_TYPE Member Function</a> on page 248-37	Gets the command type of the logical change record (LCR)
<a href="#">GET_COMMIT_SCN Member Function</a> on page 248-37	Gets the commit system change number (SCN) of the transaction to which the current LCR belongs
<a href="#">GET_COMMIT_SCN_FROM_POSITION Static Function</a> on page 248-37	Gets the commit SCN of a transaction from the input position, which is generated by an XStream outbound server
<a href="#">GET_COMMIT_TIME</a> on page 248-38	Gets the commit time of the transaction to which the current LCR belongs
<a href="#">GET_COMPATIBLE Member Function</a> on page 248-38	Gets the minimal database compatibility required to support the LCR
<a href="#">GET_EXTRA_ATTRIBUTE Member Function</a> on page 248-39	Gets the value for the specified extra attribute in the LCR
<a href="#">GET_OBJECT_NAME Member Function</a> on page 248-40	Gets the name of the object that is changed by the LCR
<a href="#">GET_OBJECT_OWNER Member Function</a> on page 248-40	Gets the owner of the object that is changed by the LCR
<a href="#">GET_POSITION Member Function</a> on page 248-41	Gets the position of the current LCR
<a href="#">GET_SCN Member Function</a> on page 248-41	Gets the SCN of the LCR
<a href="#">GET_SCN_FROM_POSITION Static Function</a> on page 248-41	Gets the SCN from the input position, which is generated by an XStream outbound server
<a href="#">GET_SOURCE_DATABASE_NAME Member Function</a> on page 248-41	Gets the source database name.
<a href="#">GET_SOURCE_TIME Member Function</a> on page 248-42	Gets the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created
<a href="#">GET_TAG Member Function</a> on page 248-42	Gets the tag for the LCR
<a href="#">GET_THREAD_NUMBER Member Function</a> on page 248-42	Gets the thread number of the database instance that made the change that is encapsulated in the LCR

**Table 248–34 (Cont.) Summary of Common Subprograms for DDL and Row LCR Types**

Subprogram	Description
<a href="#">GET_TRANSACTION_ID Member Function</a> on page 248-42	Gets the transaction identifier of the LCR
<a href="#">IS_NULL_TAG Member Function</a> on page 248-42	Returns Y if the tag for the LCR is NULL, or returns N if the tag for the LCR is not NULL
<a href="#">SET_COMMAND_TYPE Member Procedure</a> on page 248-43	Sets the command type in the LCR
<a href="#">SET_EXTRA_ATTRIBUTE Member Procedure</a> on page 248-43	Sets the value for the specified extra attribute in the LCR
<a href="#">SET_OBJECT_NAME Member Procedure</a> on page 248-44	Sets the name of the object that is changed by the LCR
<a href="#">SET_OBJECT_OWNER Member Procedure</a> on page 248-45	Sets the owner of the object that is changed by the LCR
<a href="#">SET_SOURCE_DATABASE_NAME Member Procedure</a> on page 248-45	Sets the source database name of the object that is changed by the LCR
<a href="#">SET_TAG Member Procedure</a> on page 248-45	Sets the tag for the LCR

### GET\_COMMAND\_TYPE Member Function

Gets the command type of the LCR.

**See Also:** The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

#### Syntax

```
MEMBER FUNCTION GET_COMMAND_TYPE ()
RETURN VARCHAR2;
```

### GET\_COMMIT\_SCN Member Function

Gets the commit system change number (SCN) of the transaction to which the current LCR belongs.

The commit SCN for a transaction is available only during apply or during error transaction execution. This function can be used only in a procedure DML handler, DDL handler, or error handler.

The commit SCN might not be available for an LCR that is part of an incomplete transaction. For example, persistent LCRs might not have a commit SCN. If the commit SCN is not available for an LCR, then this function returns NULL.

#### Syntax

```
MEMBER FUNCTION GET_COMMIT_SCN()
RETURN NUMBER;
```

### GET\_COMMIT\_SCN\_FROM\_POSITION Static Function

Gets the commit system change number (SCN) of a transaction from the input position, which is generated by an XStream outbound server.

#### Syntax

```
STATIC FUNCTION GET_COMMIT_SCN_FROM_POSITION(
```

```

    position IN RAW)
RETURN NUMBER;

```

## Parameters

**Table 248–35** *GET\_COMMIT\_SCN\_FROM\_POSITION* Function Parameter

Parameter	Description
position	The position You can obtain the position by using the GET_POSITION member function or by querying the DBA_XSTREAM_OUTBOUND_PROGRESS data dictionary view.

---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---

## GET\_COMMIT\_TIME

Gets the commit time of the transaction to which the current LCR belongs.

The commit time for a transaction is available only during apply or during error transaction execution. This function can be used only in a procedure DML handler, DDL handler, or error handler.

The commit time might not be available for an LCR that is part of an incomplete transaction. For example, persistent LCRs might not have a commit time. If the commit time is not available for an LCR, then this function returns NULL.

## Syntax

```

MEMBER FUNCTION GET_COMMIT_TIME()
RETURN DATE;

```

## GET\_COMPATIBLE Member Function

Gets the minimal database compatibility required to support the LCR. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

The return value for this function can be one of the following:

Return Value	COMPATIBLE Initialization Parameter Equivalent
DBMS_STREAMS.COMPATIBLE_9_2	9.2.0
DBMS_STREAMS.COMPATIBLE_10_1	10.1.0
DBMS_STREAMS.COMPATIBLE_10_2	10.2.0
DBMS_STREAMS.COMPATIBLE_11_1	11.1.0
DBMS_STREAMS.COMPATIBLE_11_2	11.2.0

DDL LCRs always return DBMS\_STREAMS.COMPATIBLE\_9\_2.

You can use the following functions in the DBMS\_STREAMS package for constant compatibility return values:

- The COMPATIBLE\_9\_2 function returns the DBMS\_STREAMS.COMPATIBLE\_9\_2 constant.

- The COMPATIBLE\_10\_1 function returns DBMS\_STREAMS.COMPATIBLE\_10\_1 constant.
- The COMPATIBLE\_10\_2 function returns DBMS\_STREAMS.COMPATIBLE\_10\_2 constant.
- The COMPATIBLE\_11\_1 function returns DBMS\_STREAMS.COMPATIBLE\_11\_1 constant.
- The COMPATIBLE\_11\_2 function returns DBMS\_STREAMS.COMPATIBLE\_11\_2 constant.
- The MAX\_COMPATIBLE function returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database.

You can use these functions with the GET\_COMPATIBLE member function for an LCR in rule conditions and apply handlers.

---

**Note:** You can determine which database objects in a database are not supported by Oracle Streams by querying the DBA\_STREAMS\_UNSUPPORTED data dictionary view.

---

**See Also:**

- *Oracle Streams Concepts and Administration* for examples of rules that discard changes that are not supported by Oracle Streams
- [Chapter 143, "DBMS\\_STREAMS"](#) and [Chapter 144, "DBMS\\_STREAMS\\_ADM"](#)
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the COMPATIBLE initialization parameter

## Syntax

```
MEMBER FUNCTION GET_COMPATIBLE()
RETURN NUMBER;
```

## GET\_EXTRA\_ATTRIBUTE Member Function

Gets the value for the specified extra attribute in the LCR. The returned extra attribute is contained within an ANYDATA instance. You can use the INCLUDE\_EXTRA\_ATTRIBUTE procedure in the DBMS\_CAPTURE\_ADM package to instruct a capture process to capture one or more extra attributes.

**See Also:** [INCLUDE\\_EXTRA\\_ATTRIBUTE Procedure](#) on page 32-33

## Syntax

```
MEMBER FUNCTION GET_EXTRA_ATTRIBUTE(
    attribute_name IN VARCHAR2)
RETURN ANYDATA;
```

## Parameters

**Table 248–36** *GET\_EXTRA\_ATTRIBUTE Function Parameter*

Parameter	Description
<code>attribute_name</code>	<p>The name of the extra attribute to return</p> <p>Valid names are:</p> <ul style="list-style-type: none"> <li>■ <code>row_id</code> The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is <code>UROWID</code>.</li> <li>■ <code>serial#</code> The serial number of the session that performed the change captured in the LCR. The type is <code>NUMBER</code>.</li> <li>■ <code>session#</code> The identifier of the session that performed the change captured in the LCR. The type is <code>NUMBER</code>.</li> <li>■ <code>thread#</code> The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is <code>NUMBER</code>.</li> <li>■ <code>tx_name</code> The name of the transaction that includes the LCR. The type is <code>VARCHAR2</code>.</li> <li>■ <code>username</code> The name of the current user who performed the change captured in the LCR. The type is <code>VARCHAR2</code>.</li> </ul> <p>An error is raised if the specified <code>attribute_name</code> is not valid.</p> <p>If no value exists for the specified extra attribute, then the function returns a <code>NULL</code>.</p> <p><b>See Also:</b> <i>Oracle Database PL/SQL Language Reference</i> for more information about the current user</p>

### GET\_OBJECT\_NAME Member Function

Gets the name of the object that is changed by the LCR.

#### Syntax

```
MEMBER FUNCTION GET_OBJECT_NAME()
RETURN VARCHAR2;
```

### GET\_OBJECT\_OWNER Member Function

Gets the owner of the object that is changed by the LCR.

#### Syntax

```
MEMBER FUNCTION GET_OBJECT_OWNER()
RETURN VARCHAR2;
```

## GET\_POSITION Member Function

Gets the position of the current LCR. The position uniquely identifies each LCR. The position strictly increases within each transaction and across transactions.

LCR position is commonly used in XStream configurations.

### Syntax

```
MEMBER FUNCTION GET_POSITION()
RETURN RAW;
```

---



---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---



---

## GET\_SCN Member Function

Gets the system change number (SCN) of the LCR.

### Syntax

```
MEMBER FUNCTION GET_SCN()
RETURN NUMBER;
```

## GET\_SCN\_FROM\_POSITION Static Function

Gets the system change number (SCN) from the input position, which is generated by an XStream outbound server.

### Syntax

```
STATIC FUNCTION GET_SCN_FROM_POSITION(
    position IN RAW)
RETURN NUMBER;
```

### Parameters

**Table 248–37** GET\_SCN\_FROM\_POSITION Function Parameter

Parameter	Description
position	The position  You can obtain the position by using the GET_POSITION member function or by querying the DBA_XSTREAM_OUTBOUND_PROGRESS data dictionary view.

---



---

**Note:** Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

---



---

## GET\_SOURCE\_DATABASE\_NAME Member Function

Gets the global name of the source database. The source database is the database where the change occurred.

### Syntax

```
MEMBER FUNCTION GET_SOURCE_DATABASE_NAME()
RETURN VARCHAR2;
```

### GET\_SOURCE\_TIME Member Function

Gets the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.

#### Syntax

```
MEMBER FUNCTION GET_SOURCE_TIME()  
RETURN DATE;
```

### GET\_TAG Member Function

Gets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DML or DDL change when apply forwarding is used.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

#### Syntax

```
MEMBER FUNCTION GET_TAG()  
RETURN RAW;
```

### GET\_THREAD\_NUMBER Member Function

Gets the thread number of the database instance that made the change that is encapsulated in the LCR. Typically, the thread number is relevant in an Oracle Real Application Clusters configuration.

**See Also:** *Oracle Real Application Clusters Administration and Deployment Guide*

#### Syntax

```
MEMBER FUNCTION GET_THREAD_NUMBER()  
RETURN NUMBER;
```

### GET\_TRANSACTION\_ID Member Function

Gets the transaction identifier of the LCR.

#### Syntax

```
MEMBER FUNCTION GET_TRANSACTION_ID()  
RETURN VARCHAR2;
```

### IS\_NULL\_TAG Member Function

Returns Y if the tag for the LCR is NULL, or returns N if the tag for the LCR is not NULL.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

#### Syntax

```
MEMBER FUNCTION IS_NULL_TAG()  
RETURN VARCHAR2;
```



## SET\_COMMAND\_TYPE Member Procedure

Sets the command type in the LCR. If the command type specified cannot be interpreted, then this procedure raises an error. For example, changing `INSERT` to `GRANT` would raise an error.

### See Also:

- The description of the `command_type` parameter in [LCR\\$\\_DDL\\_RECORD Constructor Function Parameters](#) on page 248-7
- The description of the `command_type` parameter in [LCR\\$\\_ROW\\_RECORD Type](#) on page 248-15
- The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

## Syntax

```
MEMBER PROCEDURE SET_COMMAND_TYPE(
    command_type IN VARCHAR2);
```

## Parameter

**Table 248–38 SET\_COMMAND\_TYPE Procedure Parameter**

Parameter	Description
<code>command_type</code>	The command type Set this parameter to a non-NULL value.

## SET\_EXTRA\_ATTRIBUTE Member Procedure

Sets the value for the specified extra attribute in the LCR. You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a capture process to capture one or more extra attributes.

**See Also:** [INCLUDE\\_EXTRA\\_ATTRIBUTE Procedure](#) on page 32-33

## Syntax

```
MEMBER PROCEDURE SET_EXTRA_ATTRIBUTE(
    attribute_name IN VARCHAR2,
    attribute_value IN ANYDATA);
```

## Parameters

**Table 248–39 SET\_EXTRA\_ATTRIBUTE Procedure Parameter**

Parameter	Description
attribute_name	<p>The name of the extra attribute to set</p> <p>Valid names are:</p> <ul style="list-style-type: none"> <li>▪ row_id The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is VARCHAR2.</li> <li>▪ serial# The serial number of the session that performed the change captured in the LCR. The type is NUMBER.</li> <li>▪ session# The identifier of the session that performed the change captured in the LCR. The type is NUMBER.</li> <li>▪ thread# The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is NUMBER.</li> <li>▪ tx_name The name of the transaction that includes the LCR. The type is VARCHAR2.</li> <li>▪ username The name of the current user who performed the change captured in the LCR. The type is VARCHAR2.</li> </ul> <p>An error is raised if the specified attribute_name is not valid.</p> <p><b>See Also:</b> <i>Oracle Database PL/SQL Language Reference</i> for more information about the current user</p>
attribute_value	<p>The value to which the specified extra attribute is set</p> <p>If set to NULL, then this procedure removes the specified extra attribute from the LCR. To set to NULL, encapsulate the NULL in an ANYDATA instance.</p>

## SET\_OBJECT\_NAME Member Procedure

Sets the name of the object that is changed by the LCR.

## Syntax

```
MEMBER PROCEDURE SET_OBJECT_NAME (
    object_name IN VARCHAR2);
```

## Parameter

**Table 248–40 SET\_OBJECT\_NAME Procedure Parameter**

Parameter	Description
object_name	The name of the object

**SET\_OBJECT\_OWNER Member Procedure**

Sets the owner of the object that is changed by the LCR.

**Syntax**

```
MEMBER PROCEDURE SET_OBJECT_OWNER(
    object_owner IN VARCHAR2);
```

**Parameter****Table 248–41 SET\_OBJECT\_OWNER Procedure Parameter**

Parameter	Description
object_owner	The schema that contains the object

**SET\_SOURCE\_DATABASE\_NAME Member Procedure**

Sets the source database name of the object that is changed by the LCR.

**Syntax**

```
MEMBER PROCEDURE SET_SOURCE_DATABASE_NAME(
    source_database_name IN VARCHAR2);
```

**Parameter****Table 248–42 SET\_SOURCE\_DATABASE\_NAME Procedure Parameter**

Parameter	Description
source_database_name	The source database of the change  If you do not include the domain name, then the procedure appends the local domain to the database name automatically. For example, if you specify DBS1 and the local domain is EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically. Set this parameter to a non-NULL value.

**SET\_TAG Member Procedure**

Sets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the change when apply forwarding is used.

**See Also:** *Oracle Streams Replication Administrator's Guide* for more information about tags

**Syntax**

```
MEMBER PROCEDURE SET_TAG(
    tag IN RAW);
```

**Parameter****Table 248-43** *SET\_TAG Procedure Parameter*

<b>Parameter</b>	<b>Description</b>
tag	The binary tag for the LCR The size limit for a tag value is two kilobytes.

## LCR\$\_ROW\_LIST Type

Identifies a list of column values for a row in a table.

This type uses the LCR\$\_ROW\_UNIT type and is used in the LCR\$\_ROW\_RECORD type.

**See Also:**

- [LCR\\$\\_ROW\\_UNIT Type](#) on page 248-48
- [LCR\\$\\_ROW\\_RECORD Type](#) on page 248-15

### Syntax

```
CREATE TYPE SYS.LCR$_ROW_LIST AS TABLE OF SYS.LCR$_ROW_UNIT  
/
```

## LCR\$\_ROW\_UNIT Type

Identifies the value for a column in a row.

This type is used in the LCR\$\_ROW\_LIST type.

**See Also:** [LCR\\$\\_ROW\\_LIST Type](#) on page 248-47

### Syntax

```
CREATE TYPE LCR$_ROW_UNIT AS OBJECT (
  column_name      VARCHAR2(4000),
  data             ANYDATA,
  lob_information  NUMBER,
  lob_offset       NUMBER,
  lob_operation_size NUMBER,
  long_information NUMBER);
/
```

### Attributes

**Table 248-44** LCR\$\_ROW\_UNIT Attributes

Attribute	Description												
column_name	The name of the column												
data	The data contained in the column												
lob_information	Contains the LOB information for the column and contains one of the following values: <table border="0" style="margin-left: 20px;"> <tr> <td>DBMS_LCR.NOT_A_LOB</td> <td>CONSTANT NUMBER := 1;</td> </tr> <tr> <td>DBMS_LCR.NULL_LOB</td> <td>CONSTANT NUMBER := 2;</td> </tr> <tr> <td>DBMS_LCR.INLINE_LOB</td> <td>CONSTANT NUMBER := 3;</td> </tr> <tr> <td>DBMS_LCR.EMPTY_LOB</td> <td>CONSTANT NUMBER := 4;</td> </tr> <tr> <td>DBMS_LCR.LOB_CHUNK</td> <td>CONSTANT NUMBER := 5;</td> </tr> <tr> <td>DBMS_LCR.LAST_LOB_CHUNK</td> <td>CONSTANT NUMBER := 6;</td> </tr> </table>	DBMS_LCR.NOT_A_LOB	CONSTANT NUMBER := 1;	DBMS_LCR.NULL_LOB	CONSTANT NUMBER := 2;	DBMS_LCR.INLINE_LOB	CONSTANT NUMBER := 3;	DBMS_LCR.EMPTY_LOB	CONSTANT NUMBER := 4;	DBMS_LCR.LOB_CHUNK	CONSTANT NUMBER := 5;	DBMS_LCR.LAST_LOB_CHUNK	CONSTANT NUMBER := 6;
DBMS_LCR.NOT_A_LOB	CONSTANT NUMBER := 1;												
DBMS_LCR.NULL_LOB	CONSTANT NUMBER := 2;												
DBMS_LCR.INLINE_LOB	CONSTANT NUMBER := 3;												
DBMS_LCR.EMPTY_LOB	CONSTANT NUMBER := 4;												
DBMS_LCR.LOB_CHUNK	CONSTANT NUMBER := 5;												
DBMS_LCR.LAST_LOB_CHUNK	CONSTANT NUMBER := 6;												
lob_offset	The LOB offset specified in the number of characters for CLOB columns and the number of bytes for BLOB columns  Valid values are NULL or a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE.												
lob_operation_size	If lob_information for the LOB is DBMS_LCR.LAST_LOB_CHUNK, then this parameter can be set to either a valid LOB ERASE value or a valid LOB TRIM value. A LOB ERASE value must be a positive integer less than or equal to DBMS_LOB.LOBMAXSIZE. A LOB TRIM value must be a nonnegative integer less than or equal to DBMS_LOB.LOBMAXSIZE.  If lob_information is not DBMS_LCR.LAST_LOB_CHUNK and for all other operations, is NULL.												
long_information	Contains the LONG information for the column and contains one of the following values: <table border="0" style="margin-left: 20px;"> <tr> <td>DBMS_LCR.not_a_long</td> <td>CONSTANT NUMBER := 1;</td> </tr> <tr> <td>DBMS_LCR.null_long</td> <td>CONSTANT NUMBER := 2;</td> </tr> <tr> <td>DBMS_LCR.inline_long</td> <td>CONSTANT NUMBER := 3;</td> </tr> <tr> <td>DBMS_LCR.long_chunk</td> <td>CONSTANT NUMBER := 4;</td> </tr> <tr> <td>DBMS_LCR.last_long_chunk</td> <td>CONSTANT NUMBER := 5;</td> </tr> </table>	DBMS_LCR.not_a_long	CONSTANT NUMBER := 1;	DBMS_LCR.null_long	CONSTANT NUMBER := 2;	DBMS_LCR.inline_long	CONSTANT NUMBER := 3;	DBMS_LCR.long_chunk	CONSTANT NUMBER := 4;	DBMS_LCR.last_long_chunk	CONSTANT NUMBER := 5;		
DBMS_LCR.not_a_long	CONSTANT NUMBER := 1;												
DBMS_LCR.null_long	CONSTANT NUMBER := 2;												
DBMS_LCR.inline_long	CONSTANT NUMBER := 3;												
DBMS_LCR.long_chunk	CONSTANT NUMBER := 4;												
DBMS_LCR.last_long_chunk	CONSTANT NUMBER := 5;												

---

---

## Oracle Multimedia ORDAudio TYPE

The Oracle Multimedia ORDAudio object type supports the storage and management of audio data.

Audio data can have different formats, encoding types, compression types, numbers of channels, sampling rates, sample sizes, and playing times (duration) depending upon how the audio data is digitally recorded. Oracle Multimedia ORDAudio can store and retrieve audio data of any data format. Oracle Multimedia ORDAudio can extract metadata from audio data of a variety of popular audio formats. Oracle Multimedia ORDAudio can also extract application attributes and store them in the `comments` attribute of the object in XML form.

- [Documentation of ORDAudio](#)

---

## Documentation of ORDAudio

For a complete description of this type within the context of Oracle Multimedia, see `ORDAudio` in the *Oracle Multimedia Reference*.



---

---

## Oracle Multimedia ORDDicom TYPE

The Oracle Multimedia ORDDicom object type supports the storage, management, and manipulation of Digital Imaging and Communications in Medicine (DICOM) data.

The DICOM standard is the dominant standard for radiology imaging and communication, to which all major manufacturers of radiological devices must conform. Oracle Multimedia DICOM provides native support for DICOM format medical images and other objects. These include objects such as single frame and multiframe images, waveforms, slices of 3-D volumes, video segments, and structured reports.

Oracle Multimedia ORDDicom provides methods to extract standard and private DICOM metadata from DICOM content into customizable XML documents, to perform image processing operations such as format conversion and thumbnail image generation, and to create new DICOM objects. Oracle Multimedia ORDDicom also provides methods to check DICOM objects for conformance based on a set of user-specified conformance rules, and to make DICOM objects anonymous based on user-defined rules that specify the set of attributes to be made anonymous and the actions to be taken to make those attributes anonymous.

- [Documentation of ORDDicom](#)

---

## Documentation of ORDDicom

For a complete description of this type within the context of Oracle Multimedia, see ORDDicom in the *Oracle Multimedia DICOM Developer's Guide*.

---

---

## Oracle Multimedia ORDDoc TYPE

The Oracle Multimedia ORDDoc object type supports the storage and management of heterogeneous media data including image, audio, and video.

Heterogeneous media data can have different formats depending upon the application generating the media data. Oracle Multimedia can store and retrieve media data of any data format. Oracle Multimedia ORDDoc data type can be used in applications that require you to store different types of heterogeneous media data in the same column so you can build a common metadata index on all the different types of media data. Using this index, you can search across all the different types of heterogeneous media data. However, you cannot use this same search technique if the different types of heterogeneous media data are stored in different types of objects in different columns of relational tables.

- [Documentation of ORDDoc](#)

---

## Documentation of ORDDoc

For a complete description of this type within the context of Oracle Multimedia, see ORDDoc in the *Oracle Multimedia Reference*.

---

---

## Oracle Multimedia ORDIImage TYPE

The Oracle Multimedia ORDIImage object type supports the storage, management, and manipulation of image data.

Digitized images consist of the image data (digitized bits) and attributes that describe and characterize the image data.

The image data (pixels) can have varying depths (bits for each pixel) depending on how the image was captured, and can be organized in various ways. The organization of the image data is known as the data format. Oracle Multimedia ORDIImage can store and retrieve image data of any data format. Oracle Multimedia ORDIImage can process (cut, scale, and generate thumbnails) of images, convert the format of images, extract properties of images of a variety of popular data formats, and extract and embed application metadata in images.

- [Documentation of ORDIImage](#)

---

## Documentation of ORDImage

For a complete description of this type within the context of Oracle Multimedia, see ORDImage in the *Oracle Multimedia Reference*.

---

## Oracle Multimedia SQL/MM Still Image TYPES

Oracle Multimedia provides support for the SQL/MM Still Image Standard, which supports the storage, retrieval, and modification of images in the database and the ability to locate images using visual predicates.

The following object relational types for images and image characteristics are included in this support:

SI\_StillImage  
SI\_AverageColor  
SI\_Color  
SI\_ColorHistogram  
SI\_FeatureList  
SI\_PositionalColor  
SI\_Texture

- [Documentation of SQL/MM Still Image](#)

---

## Documentation of SQL/MM Still Image

For a complete description of this type within the context of Oracle Multimedia, see `SQL/MM Still Image` in the *Oracle Multimedia Reference*.



---

---

## Oracle Multimedia ORDVideO TYPE

The Oracle Multimedia ORDVideO object type supports the storage and management of video data.

Digitized video consists of the video data (digitized bits) and the attributes that describe and characterize the video data. Video applications sometimes associate application-specific information, such as the description of the video training tape, date recorded, instructor's name, producer's name, and so on, within the video data.

The video data can have different formats, compression types, frame rates, frame sizes, frame resolutions, playing times, compression types, number of colors, and bit rates depending upon how the video data was digitally recorded. Oracle Multimedia ORDVideO can store and retrieve video data of any data format. Oracle Multimedia ORDVideO can extract metadata from video data of a variety of popular video formats. Oracle Multimedia ORDVideO can also extract application attributes and store them in the `comments` attribute of the object in XML form.

- [Documentation of ORDVideO](#)

---

## Documentation of ORDVide

For a complete description of this type within the context of Oracle Multimedia, see ORDVide in the *Oracle Multimedia Reference*.

---

---

## MGD\_ID Package Types

The MGD\_ID package provides an extensible framework that supports current radio-frequency ID (RFID) tags with the standard family of EPC bit encodings for the supported encoding types. The MGD\_ID Package also supports new and evolving tag encodings that are not included in the current EPC standard (EPC v1.1 specification). The MGD\_ID package contains several predefined types.

**See Also:** *Oracle Database Advanced Application Developer's Guide* for more information.

This chapter contains the following topics:

- [Using MGD\\_ID Package Object Types](#)
- [Summary of Types](#)
- [Summary of MGD\\_ID Subprograms](#)

The method described in this reference chapter show examples based on the examples shown in the constructor functions.

The examples in this chapter assume that the you have run the following set of commands before running the contents of each script:

```
SQL> connect / as sysdba;
Connected.
SQL> create user mgduser identified by mgduser;
SQL> grant connect, resource to mgduser;
SQL> connect mgduser
Enter password: mgduserpassword
Connected.
SQL> set serveroutput on;
```

## Using MGD\_ID Package Object Types

This section contains topics that relate to using the MGD\_ID package object types.

- [Security Model](#)

## Security Model

You must run the `catmgd.sql` script to load the `DBMS_MGD_ID_UTL` package and create the required Identity Code Package schema objects in the `MGDSYS` schema.

`MGD_ID` is a `MGDSYS`-owned object type. Any `MGD_ID` subprogram called from an anonymous PL/SQL block is run using the privileges of the current user.

A user must be granted connect and resource roles to use the `MGD_ID` object type and its subprograms.

`EXECUTE` privilege is granted to `PUBLIC` for this ADT: `MGD_ID`.

A public synonym, by the same name, is created for this ADT: `MGD_ID`.

## Summary of Types

---

Table 255–1 describes the MGD\_ID Package object types.

**Table 255–1 MGD\_ID Package Object Types**

<b>Object Type Name</b>	<b>Description</b>
<a href="#">MGD_ID_COMPONENT Object Type</a>	Datatype that specifies the name and value pair attributes that define a component
<a href="#">MGD_ID_COMPONENT_VARRAY Object Type</a>	Datatype that specifies a list of up to 128 components as name-value attribute pairs used in two constructor functions for creating an identity code type object
<a href="#">MGD_ID Object Type</a>	Represents an MGD_ID object that specifies the category identifier for the code category for this identity code and its list of components

---

## MGD\_ID\_COMPONENT Object Type

The `MGD_ID_COMPONENT` type is a datatype that specifies the name and value pair attributes that define a component.

### Syntax

```
CREATE OR REPLACE TYPE MGD_ID_COMPONENT as object (name VARCHAR2(256),  
                                                    value VARCHAR2(1024));
```

### Attributes

**Table 255–2** *MGD\_ID\_COMPONENT Attributes*

Attribute	Description
<code>name</code>	Name of component
<code>value</code>	Value of the component as a character

### Usage Notes

None.

### Examples

See the [MGD\\_ID Constructor Function](#) for an example.

## MGD\_ID\_COMPONENT\_VARRAY Object Type

The `MGD_ID_COMPONENT_VARRAY` type is a datatype that specifies a list of up to 128 components as name-value attribute pairs for use in two constructor functions for creating a product code type object with its list of components.

### Syntax

```
CREATE OR REPLACE TYPE MGD_ID_COMPONENT_VARRAY is VARRAY (128) of MGD_ID_
COMPONENT;
```

### Attributes

None.

### Usage Notes

None.

### Examples

See the [MGD\\_ID Constructor Function](#) for an example.



## MGD\_ID Object Type

The MGD\_ID type represents an identity code in an RFID application. This type represents RFID tags with standard EPC bit encoding as well as tag encodings that are not included in the EPC standard.

### Syntax

```
CREATE OR REPLACE TYPE MGD_ID as object (category_id VARCHAR2(256),
                                         components MGD_ID_COMPONENT_VARRAY);
```

### Attributes

**Table 255–3 MGD\_ID Object Type Attributes**

Attribute	Description
category_id	Category identifier for the code category of this code
components	List of components as name-value attributes

### Methods

[Table 255–5](#) describes the methods of the MGD\_ID object type.

**Table 255–4 MGD\_ID Methods**

Method	Description
MGD_ID constructor function	Creates an MGD_ID object based on the parameters passed in and returns self as a result
FORMAT function	Returns the string representation of the MGD_ID in the specified format
GET_COMPONENT function	Returns the string value of the specified MGD_ID component
TO_STRING function	Returns the string value of semicolon (;) separated component name value pairs of the MGD_ID object
TRANSLATE function	Returns the result of the conversion of the identifier from one format to the specified format

### Usage Notes

None.

### Examples

See the [Summary of MGD\\_ID Subprograms](#) section and the section about using the Identity Code package in *Oracle Database Advanced Application Developer's Guide* for examples.

## Summary of MGD\_ID Subprograms

Table 255–5 describes the subprograms in the MGD\_ID object type.

All the values and names passed to the procedures defined in the MGD\_ID object type are case insensitive unless otherwise mentioned. To preserve the case, enclose the values with double quotation marks.

**Table 255–5 MGD\_ID Object Type Subprograms**

<b>Subprogram</b>	<b>Description</b>
<a href="#">MGD_ID Constructor Function</a>	Creates an MGD_ID object based on the parameters passed in and returns self as a result
<a href="#">FORMAT Function</a>	Returns the string representation of the MGD_ID object in the specified format
<a href="#">GET_COMPONENT Function</a>	Returns the string value of the specified MGD_ID component
<a href="#">TO_STRING Function</a>	Returns the string value of semicolon (;) separated component name value pairs of the MGD_ID object
<a href="#">TRANSLATE Function</a>	Returns the result of the conversion of the identifier from one format to the specified format

## MGD\_ID Constructor Function

This constructor function constructs an identity code type object, `MGD_ID`. The constructor function is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Constructs an `MGD_ID` object type based on the category ID and a list of components.

```
constructor function MGD_ID (
    category_id      IN VARCHAR2,
    components       IN MGD_ID_COMPONENT_VARRAY)
RETURN SELF AS RESULT DETERMINISTIC;
```

Constructs an `MGD_ID` object type based on the category ID, the identifier string, and the list of additional parameters required to create it.

```
constructor function MGD_ID (
    category_id      VARCHAR2,
    identifier       VARCHAR2,
    parameter_list   VARCHAR2)
RETURN SELF AS RESULT DETERMINISTIC;
```

Constructs an `MGD_ID` object type based on the category name, category version, and a list of components.

```
constructor function MGD_ID (
    category_name    VARCHAR2,
    category_version VARCHAR2,
    components       MGD_ID_COMPONENT_VARRAY)
RETURN SELF AS RESULT DETERMINISTIC;
```

Constructs an `MGD_ID` object type based on the category name, category version, the identifier string, and the list of additional parameters required to create it.

```
constructor function MGD_ID (
    category_name    VARCHAR2,
    category_version VARCHAR2,
    identifier       VARCHAR2,
    parameter_list   VARCHAR2)
RETURN SELF AS RESULT DETERMINISTIC;
```

### Parameters

**Table 255–6 MGD\_ID Constructor Function Parameters**

Parameter	Description
<code>category_id</code>	Category identifier
<code>components</code>	List of component name value pairs
<code>category_name</code>	Category name, such as EPC
<code>category_version</code>	Category version. If NULL, the latest version for the specified category name will be used.

**Table 255–6 (Cont.) MGD\_ID Constructor Function Parameters**

Parameter	Description
identifier	<p>Identifier string in any format of an encoding scheme in the specified category. For example, for SGTIN-96 encoding, the identifier can be in the format of BINARY, PURE_IDENTITY, TAG_ENCODING, or LEGACY.</p> <p>Express this identifier as a string according to the appropriate grammar or pattern in the tag data translation (TDT) markup file. For example, a binary string consisting of characters 0 and 1, a URI (either tag-encoding or pure-identity formats), or a serialized legacy code expressed as a string format for input, such as <code>gtin=00037000302414;serial=10419703</code> for a SGTIN coding scheme.</p>
parameter_list	<p>List of additional parameters required to create the object in the representation. The list is expressed as a parameter string containing key-value pairs, separated by the semicolon (;) as a delimiter between key-value pairs. For example, for a GTIN code, the parameter string would look as follows:</p> <p><code>filter=3;companyprefixlength=7;taglength=96</code></p>

## Usage Notes

- Use `MGD_ID_UTL.EPC_ENCODING_CATEGORY_ID` as `category_id`.
- If the category is not already registered, an error is raised.
- If the `bit_length` parameter is NULL, the `bit_length` is 8\* the length of `bit_encoding`.
- If the component list does not contain all required components, an exception `MGD_ID_UTL.e_LackComponent` will be thrown.

## Examples

The following examples construct identity code type objects.

Construct an `MGD_ID` object (SGTIN-64) passing in the category ID and a list of components.

```
--Contents of constructor11.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
call DBMS_MGD_ID_UTL.refresh_category('1');
select MGD_ID('1',
            MGD_ID_COMPONENT_VARRAY (
                MGD_ID_COMPONENT('companyprefix','0037000'),
                MGD_ID_COMPONENT('itemref','030241'),
                MGD_ID_COMPONENT('serial','1041970'),
                MGD_ID_COMPONENT('schemes','SGTIN-64')
            )
        ) from dual;
call DBMS_MGD_ID_UTL.remove_proxy();

SQL> @constructor11.sql
.
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('companyprefix', '0037000'),
                                     MGD_ID_COMPONENT('itemref', '030241'),
                                     MGD_ID_COMPONENT('serial', '1041970'),
                                     MGD_ID_COMPONENT('schemes', 'SGTIN-64')))
```

.  
.  
.

Constructs an MGD\_ID object (SGTIN-64) passing in the category ID, the tag identifier, and the list of additional parameters that may be required to create it.

```
--Contents of constructor22.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
call DBMS_MGD_ID_UTL.refresh_category('1');
select MGD_ID('1',
            'urn:epc:id:sgtin:0037000.030241.1041970',
            'filter=3;scheme=SGTIN-64') from dual;
call DBMS_MGD_ID_UTL.remove_proxy();
```

```
SQL> @constructor22.sql
```

.  
.  
.

```
MGD_ID('1', MGD_ID_COMPONENT_VARRAY (MGD_ID_COMPONENT('filter', '3'),
                                     MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
                                     MGD_ID_COMPONENT('companyprefixlength', '7'),
                                     MGD_ID_COMPONENT('companyprefix', '0037000'),
                                     MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
                                     MGD_ID_COMPONENT('serial', '1041970'),
                                     MGD_ID_COMPONENT('itemref', '030241')))
```

.  
.  
.

Construct an MGD\_ID object (SGTIN-64) passing in the category name, category version (if NULL, then the latest version will be used), and a list of components.

```
--Contents of constructor33.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
call DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
select MGD_ID('EPC', NULL,
            MGD_ID_COMPONENT_VARRAY (
                MGD_ID_COMPONENT('companyprefix', '0037000'),
                MGD_ID_COMPONENT('itemref', '030241'),
                MGD_ID_COMPONENT('serial', '1041970'),
                MGD_ID_COMPONENT('schemes', 'SGTIN-64')
            )
        ) from dual;
call DBMS_MGD_ID_UTL.remove_proxy();
```

```
SQL> @constructor33.sql
```

.  
.  
.

```
MGD_ID('1', MGD_ID_COMPONENT_VARRAY (MGD_ID_COMPONENT('companyprefix', '0037000'),
                                     MGD_ID_COMPONENT('itemref', '030241'),
                                     MGD_ID_COMPONENT('serial', '1041970'),
                                     MGD_ID_COMPONENT('schemes', 'SGTIN-64')))
```

.  
.  
.

Constructs an MGD\_ID object (SGTIN-64) passing in the category name and category version, the tag identifier, and the list of additional parameters that may be required to create it.

```
--Contents of constructor4.sql
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
call DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
select MGD_ID('EPC', NULL,
             'urn:epc:id:sgtin:0037000.030241.1041970',
             'filter=3;scheme=SGTIN-64') from dual;
call DBMS_MGD_ID_UTL.remove_proxy();
```

```
SQL> @constructor4.sql
```

```
.
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('filter', '3'),
                                     MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
                                     MGD_ID_COMPONENT('companyprefixlength', '7'),
                                     MGD_ID_COMPONENT('companyprefix', '0037000'),
                                     MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
                                     MGD_ID_COMPONENT('serial', '1041970'),
                                     MGD_ID_COMPONENT('itemref', '030241')))
.
.
.
```

## FORMAT Function

This function returns the string representation of the MGD\_ID object in the specified format.

### Syntax

```
function FORMAT (parameter_list IN VARCHAR2,
                 output_format  IN VARCHAR2)
RETURN VARCHAR2 DETERMINISTIC;
```

### Parameters

**Table 255–7** *FORMAT Function Parameters*

Parameter	Description
parameter_list	List of additional parameters required to create the object in the representation. The list is expressed as a parameter string containing key-value pairs, separated by the semicolon (;) as a delimiter between key-value pairs. For example, for a GTIN code, the parameter string would look as follows:  filter=3;companyprefixlength=7;taglength=96
output_format	One of the supported output formats into which an MGD_ID component is formatted: <ul style="list-style-type: none"> <li>■ BINARY</li> <li>■ LEGACY</li> <li>■ TAG_ENCODING</li> <li>■ PURE_IDENTITY</li> <li>■ ONS_HOSTNAME</li> </ul>

### Usage Notes

None.

### Examples

See the example for the [GET\\_COMPONENT Function](#).

## GET\_COMPONENT Function

This function returns the value of the specified MGD\_ID component.

### Syntax

```
function GET_COMPONENT (
            component_name IN VARCHAR2)
RETURN VARCHAR2 DETERMINISTIC;
```

### Parameters

**Table 255–8 GET\_COMPONENT Function Parameter**

Parameter	Description
component_name	Name of component

### Usage Notes

- If the code is an invalid code, meaning its structure is not defined in the metadata table, an error is raised.
- If the code is valid, but it does not contain the required component, NULL is returned.

### Examples

The following example returns the general manager, object class, and serial number components for this GID-96 identity component:

```
--Contents of get_components.sql file
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
DECLARE
id MGD_ID;
BEGIN
  DBMS_MGD_ID_UTL.set_java_logging_level(DBMS_MGD_ID_UTL.LOGGING_LEVEL_OFF);
  DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
  -----
  --PURE_IDENTIT
  -----
  dbms_output.put_line('..Testing constructor with pure identity');
  -----
  -- PURE_IDENTITY representation can be translated to BINARY and
  -- TAG_ENCODING ONLY when BOTH scheme and filer are provided.
  -----
  id := MGD_ID('EPC', NULL, 'urn:epc:id:sgtin:0037000.030241.1041970',
'scheme=SGTIN-64;filter=3');
  dbms_output.put_line(id.to_string);
  dbms_output.put_line('filter          = ' || id.get_component('filter'));
  dbms_output.put_line('company prefix = ' || id.get_component('companyprefix'));
  dbms_output.put_line('itemref       = ' || id.get_component('itemref'));
  dbms_output.put_line('serial        = ' || id.get_component('serial'));
  dbms_output.put_line('BINARY format = ' || id.format(NULL, 'BINARY'));
  dbms_output.put_line('PURE_IDENTITY format = ' || id.format(NULL, 'PURE_IDENTITY'));
  dbms_output.put_line('TAG_ENCODING format = ' || id.format(NULL, 'TAG_ENCODING'));
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
```



```
SQL> @get_component.sql
.
.
.
..Testing constructor with pure identity
category_id =1;filter = 3;schemes = SGTIN-64;companyprefixlength =
7;companyprefix = 0037000;scheme = SGTIN-64;serial = 1041970;itemref = 030241
filter          = 3
company prefix = 0037000
itemref         = 030241
serial          = 1041970
BINARY format  =1001100000000000001000001110110001000010000011111110011000110010
PURE_IDENTITY format = urn:epc:id:sgtin:0037000.030241.1041970
TAG_ENCODING format = urn:epc:tag:sgtin-64:3.0037000.030241.1041970
PL/SQL procedure successfully completed.
.
.
.
```

## TO\_STRING Function

This function returns the semicolon (;) separated component name value pairs of the MGD\_ID object.

### Syntax

```
function TO_STRING  
    RETURN VARCHAR2;
```

### Parameters

None.

### Usage Notes

None.

### Examples

The following example converts the MGD\_ID object into a string value:

```
-- Contents of tostring3.sql file  
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');  
DECLARE  
id          MGD_ID;  
BEGIN  
    DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));  
    dbms_output.put_line('..Testing to_string');  
    id := mgd_id('EPC', NULL, 'urn:epc:id:gid:0037000.30241.1041970', 'scheme=GID-96');  
    DBMS_OUTPUT.PUT_LINE('mgd_id object as a string');  
    DBMS_OUTPUT.PUT_LINE(id.to_string);  
END;  
/  
SHOW ERRORS;  
call DBMS_MGD_ID_UTL.remove_proxy();  
connect / as sysdba;  
drop user mgduser cascade;  
  
SQL> @tostring3.sql  
. .  
..Testing to_string  
mgd_id object as a string  
category_id =1;schemes = GID-96;objectclass = 30241;generalmanager =  
0037000;scheme = GID-96;1 = 1;serial = 1041970  
PL/SQL procedure successfully completed.  
. .
```

## TRANSLATE Function

This static function translates between different representations directly without first constructing an MGD\_ID object. This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

### Syntax

Converts the identifier in one format to another given the category name, the tag identifier, the parameter list, and the output format.

```
function TRANSLATE (
    category_name    IN VARCHAR2,
    identifier       IN VARCHAR2,
    parameter_list   IN VARCHAR2,
    output_format    IN VARCHAR2)
RETURN VARCHAR2 DETERMINISTIC;
```

Converts the identifier in one format to another given the category name, category version, the tag identifier, the parameter list, and the output format.

```
function TRANSLATE (
    category_name    IN VARCHAR2,
    category_version IN VARCHAR2,
    identifier       IN VARCHAR2,
    parameter_list   IN VARCHAR2,
    output_format    IN VARCHAR2)
RETURN VARCHAR2 DETERMINISTIC;
```

### Parameters

**Table 255–9 TRANSLATE Function Parameters**

Parameter	Description
category_name	Name of category
category_version	Category version. If NULL, the latest version of the specified category name will be used.
identifier	EPC identifier, expressed as a string in accordance with one of the grammars or patterns in the TDT markup file. For example, a binary string consisting of characters 0 and 1, a URI (either tag-encoding or pure-identity formats), or a serialized legacy code expressed as a string format for input, such as gtin=00037000302414;serial=10419703 for a SGTIN coding scheme.
parameter_list	List of additional parameters required to create the object in the representation. The list is expressed as a parameter string containing key-value pairs, separated by the semicolon (;) as a delimiter between key-value pairs. For example, for a GTIN code, the parameter string would look as follows: filter=3;companyprefixlength=7;taglength=96

**Table 255–9 (Cont.) TRANSLATE Function Parameters**

Parameter	Description
output_format	One of the supported output formats into which an MGD_ID component shall be converted: <ul style="list-style-type: none"> <li>▪ BINARY</li> <li>▪ LEGACY</li> <li>▪ TAG_ENCODING</li> <li>▪ PURE_IDENTITY</li> <li>▪ ONS_HOSTNAME</li> </ul>

## Usage Notes

When converting from a pure identity representation to a binary representation, the filter value must be supplied as a value using the `parameter_list` parameter.

## Examples

The following examples translates one GID-96 representation into another:

```
-- Contents of translate1.sql file
call DBMS_MGD_ID_UTL.set_proxy('www-proxy.us.oracle.com', '80');
DECLARE
id          MGD_ID;
BEGIN
  DBMS_MGD_ID_UTL.refresh_category(DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
  dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is BINARY');
  dbms_output.put_line(
    mgd_id.translate('EPC',
NULL, '00110101000000000000100100001000100000000000011101100010000100000000000000011111110011000110
010'
, NULL, 'BINARY'));
  dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is PURE_IDENTITY');
  dbms_output.put_line(
    mgd_id.translate('EPC',
NULL, '00110101000000000000100100001000100000000000011101100010000100000000000000011111110011000110
010'
, NULL, 'PURE_IDENTITY'));
  dbms_output.put_line('Category ID is EPC, Identifier is BINARY, Output format is TAG_ENCODING');
  dbms_output.put_line(
    mgd_id.translate('EPC',
NULL, '00110101000000000000100100001000100000000000011101100010000100000000000000011111110011000110
010'
, NULL, 'TAG_ENCODING'));
  dbms_output.put_line('Category ID is EPC, Identifier is TAG_ENCODING, Output format is BINARY');
  dbms_output.put_line(
    mgd_id.translate('EPC', NULL,
'urn:epc:tag:gid-96:0037000.30241.1041970',
NULL, 'BINARY'));
  dbms_output.put_line('Category ID is EPC, Identifier is TAG_ENCODING, Output format is
PURE_IDENTITY');
  dbms_output.put_line(
    mgd_id.translate('EPC', NULL,
'urn:epc:tag:gid-96:0037000.30241.1041970',
NULL, 'PURE_IDENTITY'));
  dbms_output.put_line('Category ID is EPC, Identifier is TAG_ENCODING, Output format is TAG_
ENCODING');
  dbms_output.put_line(
    mgd_id.translate('EPC', NULL,
```

```

        'urn:epc:tag:gid-96:0037000.30241.1041970',
        NULL, 'TAG_ENCODING'));
dbms_output.put_line('Category ID is EPC, Identifier is PURE_IDENTITY, Output format is BINARY');
dbms_output.put_line(
    mgd_id.translate('EPC', NULL,
        'urn:epc:id:gid:0037000.30241.1041970',
        NULL, 'BINARY'));
dbms_output.put_line('Category ID is EPC, Identifier is PURE_IDENTITY, Output format is PURE_
IDENTITY');
dbms_output.put_line(
    mgd_id.translate('EPC', NULL,
        'urn:epc:id:gid:0037000.30241.1041970',
        NULL, 'PURE_IDENTITY'));
dbms_output.put_line('Category ID is EPC, Identifier is PURE_IDENTITY, Output format is TAG_
ENCODING');
dbms_output.put_line(
    mgd_id.translate('EPC', NULL,
        'urn:epc:id:gid:0037000.30241.1041970',
        NULL, 'TAG_ENCODING'));
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();

SQL> @translate1.sql
.
.
.
Category ID is EPC, Identifier is BINARY, Output format is BINARY
00110101000000000000100100001000100000000000011101100010000100000000000000011111110011000110010
Category ID is EPC, Identifier is BINARY, Output format is PURE_IDENTITY
urn:epc:id:gid:37000.30241.1041970
Category ID is EPC, Identifier is BINARY, Output format is TAG_ENCODING
urn:epc:tag:gid-96:37000.30241.1041970
Category ID is EPC, Identifier is TAG_ENCODING, Output format is BINARY
00110101000000000000100100001000100000000000011101100010000100000000000000011111110011000110010
Category ID is EPC, Identifier is TAG_ENCODING, Output format is PURE_IDENTITY
urn:epc:id:gid:0037000.30241.1041970
Category ID is EPC, Identifier is TAG_ENCODING, Output format is TAG_ENCODING
urn:epc:tag:gid-96:0037000.30241.1041970
Category ID is EPC, Identifier is PURE_IDENTITY, Output format is BINARY
00110101000000000000100100001000100000000000011101100010000100000000000000011111110011000110010
Category ID is EPC, Identifier is PURE_IDENTITY, Output format is PURE_IDENTITY
urn:epc:id:gid:0037000.30241.1041970
Category ID is EPC, Identifier is PURE_IDENTITY, Output format is TAG_ENCODING
urn:epc:tag:gid-96:0037000.30241.1041970
PL/SQL procedure successfully completed.
.
.
.

```



This chapter describes the types used with rules, rule sets, and evaluation contexts.

This chapter contains the following topics:

- [Using Rule Types](#)
  - Overview
  - Security Model
- [Summary of Rule Types](#)

## Using Rule Types

This section contains topics that relate to using the types used with rules, rule sets, and evaluation contexts.

- [Overview](#)
- [Security Model](#)



## Overview

This types in this chapter are used in rules and enable clients to evaluate rules with the rules engine.

**See Also:**

- [Chapter 126, "DBMS\\_RULE"](#)
- [Chapter 127, "DBMS\\_RULE\\_ADM"](#)
- *Oracle Streams Extended Examples* for examples that use rule types

## Security Model

PUBLIC is granted EXECUTE privilege on the types described in this chapter.

**See Also:** *Oracle Database Security Guide* for more information about user group PUBLIC

---

## Summary of Rule Types

**Table 256–1 Rule Types**

Type	Description
<a href="#">RE\$ATTRIBUTE_VALUE Type</a> on page 256-7	Specifies the value of a variable attribute
<a href="#">RE\$ATTRIBUTE_VALUE_LIST Type</a> on page 256-8	Identifies a list of attribute values
<a href="#">RE\$COLUMN_VALUE Type</a> on page 256-9	Specifies the value of a table column
<a href="#">RE\$COLUMN_VALUE_LIST Type</a> on page 256-10	Identifies a list of column values
<a href="#">RE\$NAME_ARRAY Type</a> on page 256-11	Identifies a list of names
<a href="#">RE\$NAME_ARRAY Type</a> on page 256-11	Identifies a list of name-value pairs
<a href="#">RE\$NV_LIST Type</a> on page 256-13	Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context and the action context for a rule
<a href="#">RE\$NV_NODE Type</a> on page 256-15	Identifies a name-value pair
<a href="#">RE\$RULE_HIT Type</a> on page 256-16	Specifies a rule found because of evaluation
<a href="#">RE\$RULE_HIT_LIST Type</a> on page 256-17	Identifies a list of rules found because of evaluation
<a href="#">RE\$TABLE_ALIAS Type</a> on page 256-18	Provides the table corresponding to an alias used in a rule evaluation context
<a href="#">RE\$TABLE_ALIAS_LIST Type</a> on page 256-19	Identifies a list of table aliases used in a rule evaluation context
<a href="#">RE\$TABLE_VALUE Type</a> on page 256-20	Specifies the value of a table row using a ROWID
<a href="#">RE\$TABLE_VALUE_LIST Type</a> on page 256-21	Identifies a list of table values
<a href="#">RE\$VARIABLE_TYPE Type</a> on page 256-22	Provides the type of a variable used in a rule evaluation context
<a href="#">RE\$VARIABLE_TYPE_LIST Type</a> on page 256-24	Identifies a list of variables and their types used in a rule evaluation context
<a href="#">RE\$VARIABLE_VALUE Type</a> on page 256-25	Specifies the value of a variable
<a href="#">RE\$VARIABLE_VALUE_LIST Type</a> on page 256-26	Identifies a list of variable values

Rule types are used with the following Oracle-supplied PL/SQL packages:

- DBMS\_RULE
- DBMS\_RULE\_ADM

You can use the DBMS\_RULE\_ADM package to create and administer rules, rule sets, and evaluation contexts, and you can use the DBMS\_RULE package to evaluate rules.

When you use Oracle Streams, rules determine which changes are captured by a capture process, which messages are propagated by a propagation, which messages are applied by an apply process, and which messages are dequeued by a messaging client. The following Oracle Streams packages use rules:

- DBMS\_APPLY\_ADM
- DBMS\_CAPTURE\_ADM
- DBMS\_PROPAGATION\_ADM
- DBMS\_STREAMS
- DBMS\_STREAMS\_ADM
- DBMS\_STREAMS\_AUTH

**See Also:** *Oracle Streams Concepts and Administration*

## RE\$ATTRIBUTE\_VALUE Type

Specifies the value of a variable attribute.

---



---

**Note:** Enclose the variable name and attribute name in double quotation marks (") if the name contains special characters.

---



---

### Syntax

```
TYPE SYS.RE$ATTRIBUTE_VALUE (
  variable_name  VARCHAR2(32),
  attribute_name VARCHAR2(4000),
  attribute_value ANYDATA);
```

### Attributes

**Table 256–2 RE\$ATTRIBUTE\_VALUE Attributes**

Attribute	Description
variable_name	Specifies the variable used in a rule
attribute_name	Specifies the attribute name. The attribute name can be a multi-component name, such as a1.b2.c3.
attribute_value	Specifies the attribute value

## RE\$ATTRIBUTE\_VALUE\_LIST Type

Identifies a list of attribute values.

### Syntax

```
TYPE SYS.RE$ATTRIBUTE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$ATTRIBUTE_VALUE;
```

## RE\$COLUMN\_VALUE Type

Specifies the value of a table column.

---



---

**Note:** Enclose the column name in double quotation marks (") if the name contains special characters.

---



---

### Syntax

```
TYPE SYS.RE$COLUMN_VALUE (
  table_alias  VARCHAR2(32),
  column_name  VARCHAR2(4000),
  column_value ANYDATA);
```

### Attributes

**Table 256-3 RE\$COLUMN\_VALUE Attributes**

Attribute	Description
table_alias	Specifies the alias used for the table in a rule
column_name	Specifies the column name
column_value	Specifies the column value

## RE\$COLUMN\_VALUE\_LIST Type

Identifies a list of column values.

### Syntax

```
TYPE SYS.RE$COLUMN_VALUE_LIST AS VARRAY(1024) OF SYS.RE$COLUMN_VALUE;
```



## RE\$NAME\_ARRAY Type

Identifies a list of names.

### Syntax

```
TYPE SYS.RE$NAME_ARRAY AS VARRAY(1024) OF VARCHAR2(30);
```

## RE\$NV\_ARRAY Type

Identifies a list of name-value pairs.

### Syntax

```
TYPE SYS.RE$NV_ARRAY AS VARRAY(1024) OF SYS.RE$NV_NODE;
```

## RE\$NV\_LIST Type

Identifies an object containing a list of name-value pairs and methods that operate on this list. This object type is used to represent the event context for rule set evaluation and the action context for a rule.

### Syntax

```
TYPE SYS.RE$NV_LIST AS OBJECT(
  actx_list SYS.RE$NV_ARRAY);
```

### Attributes

**Table 256–4 RE\$NV\_LIST Attributes**

Attribute	Description
actx_list	The list of name-value pairs

### RE\$NV\_LIST Subprograms

This section describes the following member procedures and member functions of the SYS.RE\$NV\_LIST type:

- [ADD\\_PAIR Member Procedure](#)
- [GET\\_ALL\\_NAMES Member Function](#)
- [GET\\_VALUE Member Function](#)
- [REMOVE\\_PAIR Member Procedure](#)

### ADD\_PAIR Member Procedure

Adds a name-value pair to the list of name-value pairs.

---



---

**Note:** Enclose the name in double quotation marks (") if the name contains special characters.

---



---

#### Syntax

```
MEMBER PROCEDURE ADD_PAIR(
  name IN VARCHAR2,
  value IN ANYDATA);
```

#### Parameters

**Table 256–5 ADD\_PAIR Procedure Parameters**

Parameter	Description
name	The name in the name-value pair being added to the list. If the name exists in the list, then this procedure raises an error.
value	The value in the name-value pair being added to the list

### GET\_ALL\_NAMES Member Function

Returns a list of all the names in the name-value pair list.

**Syntax**

```
MEMBER FUNCTION GET_ALL_NAMES()  
RETURN SYS.RE$NAME_ARRAY;
```

**GET\_VALUE Member Function**

Returns the value for the specified name in a name-value pair list.

---

---

**Note:** Enclose the name in double quotation marks (") if the name contains special characters.

---

---

**Syntax**

```
MEMBER FUNCTION GET_VALUE(  
    name IN VARCHAR2)  
RETURN ANYDATA;
```

**Parameters**

**Table 256–6** GET\_VALUE Procedure Parameters

Parameter	Description
name	The name whose value to return

**REMOVE\_PAIR Member Procedure**

Removes the name-value pair with the specified name from the name-value pair list.

---

---

**Note:** Enclose the name in double quotation marks (") if the name contains special characters.

---

---

**Syntax**

```
MEMBER PROCEDURE REMOVE_PAIR(  
    name IN VARCHAR2);
```

**Parameters**

**Table 256–7** REMOVE\_PAIR Procedure Parameters

Parameter	Description
name	The name of the pair to remove

## RE\$NV\_NODE Type

Identifies a name-value pair.

---

---

**Note:** Enclose the name in double quotation marks (") if the name contains special characters.

---

---

### Syntax

```
TYPE SYS.RE$NV_NODE (  
    nvn_name    VARCHAR2(30),  
    nvn_value   ANYDATA);
```

### Attributes

**Table 256–8** RE\$NV\_NODE Attributes

Attribute	Description
nvn_name	Specifies the name in the name-value pair
nvn_value	Specifies the value in the name-value pair

## RE\$RULE\_HIT Type

Specifies a rule found because of an evaluation.

**See Also:**

- [CREATE\\_RULE Procedure](#) on page 127-15
- [ALTER\\_RULE Procedure](#) on page 127-11

### Syntax

```
TYPE SYS.RE$RULE_HIT (  
    rule_name          VARCHAR2(65),  
    rule_action_context RE$NV_LIST);
```

### Attributes

**Table 256–9 RE\$RULE\_HIT Attributes**

Attribute	Description
rule_name	The rule name in the form <i>schema_name.rule_name</i> . For example, a rule named <code>employee_rule</code> in the <code>hr</code> schema is returned in the form <code>"hr"."employee_rule"</code> .
rule_action_context	The rule action context as specified in the <code>CREATE_RULE</code> or <code>ALTER_RULE</code> procedure of the <code>DBMS_RULE_ADM</code> package

## RE\$RULE\_HIT\_LIST Type

Identifies a list of rules found because of an evaluation.

### Syntax

```
TYPE SYS.RE$RULE_HIT_LIST AS VARRAY(1024) OF SYS.RE$RULE_HIT;
```

## RE\$TABLE\_ALIAS Type

Provides the table corresponding to an alias used in a rule evaluation context. A specified table name must satisfy the schema object naming rules.

---

---

**Note:** Enclose the table name in double quotation marks (") if the name contains special characters.

---

---

**See Also:** *Oracle Database SQL Language Reference* for information about schema object naming rules

### Syntax

```
TYPE SYS.RE$TABLE_ALIAS IS OBJECT(  
    table_alias VARCHAR2(32),  
    table_name  VARCHAR2(194));
```

### Attributes

**Table 256–10 RE\$TABLE\_ALIAS Attributes**

Attribute	Description
table_alias	The alias used for the table in a rule
table_name	The table name referred to by the alias. A synonym can be specified. The table name is resolved in the evaluation context schema.  The format is one of the following:  <i>schema_name.table_name</i>  <i>table_name</i>  For example, if the <i>schema_name</i> is <i>hr</i> and the <i>table_name</i> is <i>employees</i> , then enter the following:  <i>hr.employees</i>



## RE\$TABLE\_ALIAS\_LIST Type

Identifies a list of table aliases used in a rule evaluation context.

### Syntax

```
TYPE SYS.RE$TABLE_ALIAS_LIST AS VARRAY(1024) OF SYS.RE$TABLE_ALIAS;
```

## RE\$TABLE\_VALUE Type

Specifies the value of a table row using a ROWID.

### Syntax

```
TYPE SYS.RE$TABLE_VALUE(  
    table_alias VARCHAR2(32),  
    table_rowid VARCHAR2(18));
```

### Attributes

**Table 256–11 RE\$TABLE\_VALUE Attributes**

Attribute	Description
table_alias	Specifies the alias used for the table in a rule
table_rowid	Specifies the rowid for the table row

## RE\$TABLE\_VALUE\_LIST Type

Identifies a list of table values.

---

---

**Note:** Each table alias in the list in the list must be unique.

---

---

### Syntax

```
TYPE SYS.RE$TABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$TABLE_VALUE;
```

## RE\$VARIABLE\_TYPE Type

Provides the type of a variable used in a rule evaluation context. A specified variable name must satisfy the schema object naming rules.

---

**Note:** Enclose the variable name in double quotation marks (") if the name contains special characters.

---

**See Also:** *Oracle Database SQL Language Reference* for information about schema object naming rules

### Syntax

```
TYPE SYS.RE$VARIABLE_TYPE (
    variable_name          VARCHAR2 (32) ,
    variable_type          VARCHAR2 (4000) ,
    variable_value_function VARCHAR2 (228) ,
    variable_method_function VARCHAR2 (228) );
```

### Attributes

**Table 256–12 RE\$VARIABLE\_TYPE Attributes**

Attribute	Description
variable_name	The variable name used in a rule
variable_type	The type that is resolved in the evaluation context schema. Any valid Oracle built-in data type, user-defined type, or Oracle-supplied type can be specified. See the <i>Oracle Database SQL Language Reference</i> for more information about these types.
variable_value_function	A value function that can be specified for implicit variables. A synonym can be specified. The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context.  See the "Usage Notes" for more information.
variable_method_function	Specifies a value function, which can return the result of a method invocation. Specifying such a function can speed up evaluation, if there are many simple rules that invoke the method on the variable. The function can be a synonym or a remote function.  The function name is resolved in the evaluation context schema. It is executed on behalf of the owner of a rule set using the evaluation context or containing a rule that uses the evaluation context.  See the "Usage Notes" for more information.

### Usage Notes

The functions for both the `variable_value_function` parameter and `variable_method_function` parameter have the following format:

```
schema_name.package_name.function_name@dblink
```

Any of the following parts of the format can be omitted: *schema\_name*, *package\_name*, and *@dblink*.

For example, if the *schema\_name* is *hr*, the *package\_name* is *var\_pac*, the *function\_name* is *func\_value*, and the *dblink* is *dbs1.net*, then enter the following:

```
hr.var_pac.func_value@dbs1.net
```

The following sections describe the signature of the functions.

### Signature for *variable\_value\_function*

The function must have the following signature:

```
FUNCTION variable_value_function_name(
  evaluation_context_schema IN VARCHAR2,
  evaluation_context_name   IN VARCHAR2,
  variable_name             IN VARCHAR2,
  event_context             IN SYS.RE$NV_LIST )
RETURN SYS.RE$VARIABLE_VALUE;
```

### Signature for *variable\_method\_function*

This function must have the following signature:

```
FUNCTION variable_method_function_name(
  evaluation_context_schema IN VARCHAR2,
  evaluation_context_name   IN VARCHAR2,
  variable_value            IN SYS.RE$VARIABLE_VALUE,
  method_name              IN VARCHAR2,
  event_context            IN SYS.RE$NV_LIST)
RETURN SYS.RE$ATTRIBUTE_VALUE;
```

## RE\$VARIABLE\_TYPE\_LIST Type

Identifies a list of variables and their types used in a rule evaluation context.

### Syntax

```
TYPE SYS.RE$VARIABLE_TYPE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_TYPE;
```

## RE\$VARIABLE\_VALUE Type

Specifies the value of a variable.

---

---

**Note:** Enclose the variable name in double quotation marks (") if the name contains special characters.

---

---

### Syntax

```
TYPE SYS.RE$VARIABLE_VALUE (  
    variable_name VARCHAR2(32),  
    variable_data ANYDATA);
```

### Attributes

**Table 256–13** RE\$VARIABLE\_VALUE Attributes

Attribute	Description
variable_name	Specifies the variable name used in a rule
variable_data	Specifies the data for the variable value

## RE\$VARIABLE\_VALUE\_LIST Type

Identifies a list of variable values.

### Syntax

```
TYPE SYS.RE$VARIABLE_VALUE_LIST AS VARRAY(1024) OF SYS.RE$VARIABLE_VALUE;
```



---

---

## Rules Manager Types

---

---

**Note:** This functionality is deprecated with Oracle Database Release 11.2 and obsoleted with Release 12.1. For details regarding obsolescence, see My Oracle Support Note ID 1244535.1

---

---

Rules Manager contains one predefined type and a public synonym for this type.

**See Also:** *Oracle Database Rules Manager and Expression Filter Developer's Guide* for more information.

This chapter contains the following topics:

- [Using Rules Manager Types](#)
- [Summary of Rule Manager Types](#)

## Using Rules Manager Types

This section contains topics that relate to using the Rules Manager Types.

- [Security Model](#)

## Security Model

The Oracle Database installation runs the `catrul.sql` script to load the `DBMS_RLMGR` package and create the required Rules Manager schema objects in the `EXFSYS` schema.

## Summary of Rule Manager Types

[Table 257-1](#) describes the Rules Manager object type.

**Table 257-1 Rules Manager Object Types**

Object Type Name	Description
<a href="#">RLM\$EVENTIDS Object Type</a>	Specifies a list of event identifiers to the CONSUME_PRIM_EVENTS procedure

## RLM\$EVENTIDS Object Type

The RLM\$EVENTIDS defines a table of VARCHAR2 values.

### Syntax

```
CREATE OR REPLACE TYPE RLM$EVENTIDS is table of VARCHAR2(38);
```

### Attributes

None.

### Usage Notes

- RLM\$EVENTIDS type passes a list of event identifiers to the CONSUME\_PRIM\_EVENTS procedure. These event identifiers are ROWIDS for the corresponding events in the database and their values are available through the arguments of the action callback procedure and rule class results view columns, when the rule class is configured for RULE consumption policy

### Examples

The following commands show the body of the action callback procedure for a rule class configured for RULE consumption policy. This demonstrates the use of RLM\$EVENTDIDS type to consume the events before executing the action for the matched rules.

```
CREATE OR REPLACE PROCEDURE PromoAction (
    Flt          AddFlight,
    Flt_EvtId ROWID,    --- rowid for the fligt primitive event
    Car          AddRentalCar,
    Car_EvtId ROWID,
    rlm$rule     TravelPromotions%ROWTYPE) is
    evtcnsmd NUMBER;
BEGIN
    evtcnsmd := dbms_rlmgr.consume_prim_events(
        rule_class => 'TravelPromotions',
        event_idents => RLM$EVENTIDS(Flt_EvtId, Car_EvtId));

    IF (evtcnsmd = 1) THEN
        -- consume operation was successful; perform the action ---
        OfferPromotion (Flt.CustID, rlm$rule.PromoType, rlm$rule.OfferedBy);
    END IF;
END;
/
```



---

---

## UTL Streams Types

UTL Streams Types describes abstract types used with Oracle XML functionality.

Four abstract PL/SQL streams are introduced and defined within the 'SYS' schema. The streams may be referenced by PUBLIC and are described in the following sections.

**See Also:** For more information, see A.

This chapter contains the following topics:

- [Summary of UTL Binary Streams Types](#)

## Summary of UTL Binary Streams Types

**Table 258–1 UTL Binary Streams Types**

Type	Description
<a href="#">UTL_BINARYINPUTSTREAM</a> Type on page 258-3	Reads bytes and closes a stream.
<a href="#">UTL_BINARYOUTPUTSTREAM</a> Type on page 258-4	Writes bytes and closes a stream.
<a href="#">UTL_CHARACTERINPUTSTREAM</a> Type on page 258-5	Reads chars and closes a stream.
<a href="#">UTL_CHARACTEROUTPUTSTREAM</a> Type on page 258-6	Writes chars and closes a stream.



## UTL\_BINARYINPUTSTREAM Type

This type is similar to `java.io.InputStream` in that it can only read and close a stream.

### Syntax

```
CREATE OR REPLACE TYPE Utl_BinaryInputStream AS OBJECT (

    MEMBER FUNCTION available (
        self      IN OUT NOCOPY Utl_BinaryInputStream)
        RETURN INTEGER,

    MEMBER FUNCTION read (
        self      IN OUT NOCOPY Utl_BinaryInputStream,
        numBytes IN              INTEGER DEFAULT 1)
        RETURN RAW,

    MEMBER PROCEDURE read (
        self      IN OUT NOCOPY Utl_BinaryInputStream,
        bytes     IN OUT NOCOPY RAW,
        numBytes  IN OUT        INTEGER),

    MEMBER PROCEDURE read (
        self      IN OUT NOCOPY Utl_BinaryInputStream,
        bytes     IN OUT NOCOPY RAW,
        offset    IN INTEGER,
        numBytes  IN OUT        INTEGER),

    member function close (
        self      In Out Nocopy Utl_BinaryInputStream)

) NOT FINAL;
```

### Attributes

**Table 258–2 UTL\_BINARYINPUTSTREAM Type Member Subprograms**

Member Subprogram	Description
AVAILABLE	Returns the number of bytes available to be read
READ	<ul style="list-style-type: none"> <li>■ #1 - Reads the number of bytes specified by <code>numBytes</code> (default is 1) and returns the bytes as a <code>RAW</code>. If there are no remaining bytes a value of <code>NULL</code> is returned.</li> <li>■ #2 - Reads the number of bytes specified in <code>numBytes</code> into the parameter <code>bytes</code>. Additionally, the actual number of bytes read is returned in parameter <code>numBytes</code>. If this parameter is set to 0 then there are no more bytes to be read.</li> <li>■ #3 - Reads the number of bytes specified in <code>numBytes</code> into the parameter <code>bytes</code>, beginning at the <code>offset</code> specified by parameter <code>offset</code>. The actual number of bytes read is returned in parameter <code>numBytes</code>. If this value is 0, then there are no additional bytes to be read.</li> </ul>
CLOSE	Releases all resources held on the node to support the stream

## UTL\_BINARYOUTPUTSTREAM Type

This type is similar to `java.io.OutputStream` in that it can only write and close a stream.

### Syntax

```
CREATE OR REPLACE TYPE Utl_BinaryOutputStream AS OBJECT (

    MEMBER FUNCTION write (                                     -- #1
        self          IN OUT NOCOPY sys.utl_BinaryOutputStream,
        bytes         IN           RAW,
        numBytes      IN           INTEGER DEFAULT 1)
    RETURN INTEGER,

    MEMBER PROCEDURE write (                                  -- #2
        self          IN OUT NOCOPY sys.utl_BinaryOutputStream,
        bytes         IN NOCOPY   RAW,
        numBytes      IN OUT      INTEGER),

    MEMBER PROCEDURE write (                                  -- #3
        self          IN OUT NOCOPY utl_BinaryOutputStream,
        bytes         IN NOCOPY   RAW,
        offset        IN           INTEGER,
        numBytes      IN OUT      INTEGER),

    MEMBER PROCEDURE flush (
        self          IN OUT NOCOPY utl_BinaryOutputStream),

    MEMBER PROCEDURE close (
        self          IN OUT NOCOPY utl_BinaryOutputStream)

) NOT FINAL;
```

### Attributes

**Table 258–3 UTL\_BINARYOUTPUTSTREAM Type Member Subprograms**

Member Subprogram	Description
WRITE	<ul style="list-style-type: none"> <li>■ #1 - Writes the number of bytes specified by <code>numBytes</code> (default is 1) from RAW into the stream. The actual number of bytes written is returned.</li> <li>■ #2 - Writes the number of bytes specified in parameter <code>numBytes</code> from parameter bytes to the stream. The actual number of bytes written is returned in parameter <code>numBytes</code>.</li> <li>■ #3 - Writes the number of bytes specified by <code>numBytes</code> to the stream, beginning at the offset specified by parameter <code>offset</code>. The actual number of bytes written is returned in parameter <code>numBytes</code>.</li> </ul>
FLUSH	Insures that any buffered bytes are copied to the node destination
CLOSE	Frees all resources associated with the stream

## UTL\_CHARACTERINPUTSTREAM Type

This type is similar to `java.io.Reader` in that it can only read characters (chars) and close a stream.

### Syntax

```
CREATE OR REPLACE TYPE Utl_CharacterInputStream AS OBJECT (

    MEMBER FUNCTION available (
        self          IN OUT NOCOPY  utl_CharacterInputStream)
    RETURN INTEGER,

    MEMBER FUNCTION read (
        self          IN OUT NOCOPY  utl_CharacterInputStream,
        numChars     IN              INTEGER DEFAULT 1,
        lineFeed     IN              BOOLEAN DEFAULT FALSE)
    RETURN VARCHAR2,

    MEMBER PROCEDURE read (
        self          IN OUT NOCOPY  utl_CharacterInputStream,
        chars        IN OUT NOCOPY  VARCHAR2,
        numChars     IN OUT         INTEGER,
        lineFeed     IN BOOLEAN DEFAULT FALSE),

    MEMBER PROCEDURE read (
        self          IN OUT NOCOPY  utl_CharacterInputStream,
        chars        IN OUT NOCOPY  VARCHAR2,
        offset       IN              INTEGER,
        numChars     IN OUT         INTEGER,
        lineFeed     IN              BOOLEAN DEFAULT FALSE),

    MEMBER PROCEDURE close (
        self          IN OUT NOCOPY  utl_CharacterInputStream)

) NOT FINAL;
```

### Attributes

**Table 258–4 UTL\_CHARACTERINPUTSTREAM Type Member Subprograms**

Member Subprogram	Description
AVAILABLE	Returns the number of bytes available to be read
READ	<ul style="list-style-type: none"> <li>■ #1 - Returns the number of characters remaining to be read</li> <li>■ #2 - Reads the number of characters specified by <code>numChars</code> (default value is 1) and returns the characters as a <code>VARCHAR2</code>. If the value of <code>lineFeed</code> is true (default value is <code>FALSE</code>) then the reading stops if a linefeed character is found. If there are no remaining characters a value of <code>NULL</code> is returned.</li> <li>■ #3 - Reads reads the number of characters specified by parameter <code>numChars</code> into the parameter <code>chars</code>. Additionally, the actual number of characters read is returned in parameter <code>numChars</code>. If this value is 0, then there are no more characters to be read. If the value of <code>lineFeed</code> is <code>TRUE</code> (default is <code>FALSE</code>), then reading stops if a linefeed character is encountered.</li> </ul>
CLOSE	Releases all resources held by the stream

## UTL\_CHARACTEROUTPUTSTREAM Type

This type is similar to `java.io.Reader` in that it can only read characters (`chars`) and close a stream.

### Syntax

```
CREATE OR REPLACE TYPE utl_CharacterOutputStream AS OBJECT (

  MEMBER FUNCTION write (                                     -- #1
    self      IN OUT NOCOPY utl_CharacterOutputStream,
    chars     IN             VARCHAR2,
    numChars  IN             INTEGER DEFAULT 1,
    lineFeed  IN             BOOLEAN DEFAULT FALSE)
  RETURN INTEGER,

  MEMBER PROCEDURE write (                                   -- #2
    self      IN OUT NOCOPY utl_CharacterOutputStream,
    chars     IN OUT NOCOPY VARCHAR2,
    numChars  IN OUT        INTEGER,
    lineFeed  IN             BOOLEAN DEFAULT FALSE),

  member procedure write (                                   -- #3
    self      IN OUT NOCOPY utl_CharacterOutputStream,
    chars     IN NOCOPY     varchar2,
    offset    IN            integer,
    numChars  IN OUT        integer,
    lineFeed  IN            boolean default false),

  MEMBER PROCEDURE flush (
    self      IN OUT NOCOPY utl_CharacterOutputStream),

  MEMBER PROCEDURE close (
    self      IN OUT NOCOPY utl_CharacterOutputStream)

) NOT FINAL;
```

### Attributes

**Table 258–5 UTL\_CHARACTEROUTPUTSTREAM Type Member Subprograms**

Member Subprogram	Description
WRITE	<ul style="list-style-type: none"> <li>■ #1 - Writes the number of characters specified by <code>numChars</code> (default is 1) from parameter <code>chars</code> into the stream and returns the actual number of characters written. If the value of <code>lineFeed</code> is <code>TRUE</code> (default is <code>FALSE</code>) a <code>lineFeed</code> character is inserted after the last character.</li> <li>■ #2 - writes the number of characters specified by parameter <code>numChars</code>, from parameter <code>chars</code> into the stream. The actual number of characters written is returned in parameter <code>numChars</code>. If the value of <code>lineFeed</code> is <code>true</code> (default is <code>FALSE</code>) a <code>lineFeed</code> character is inserted after the last character.</li> <li>■ #3 - Writes the number of characters specified by parameter <code>numChars</code>, from parameter <code>chars</code>, beginning at <code>offset</code> specified by parameter <code>offset</code>. The actual number of characters written is returned in parameter <code>numChars</code>. If the value of <code>lineFeed</code> is <code>true</code> (default is <code>FALSE</code>) a <code>lineFeed</code> character is inserted after the last character .</li> </ul>
FLUSH	Copies all characters that may be contained within buffers to the node value

**Table 258-5 (Cont.) UTL\_CHARACTEROUTPUTSTREAM Type Member Subprograms**

<b>Member Subprogram</b>	<b>Description</b>
CLOSE	Releases all resources held by the stream



`XMLType` is a system-defined opaque type for handling XML data. It has predefined member functions on it to extract XML nodes and fragments.

You can create columns of `XMLType` and insert XML documents into it. You can also generate XML documents as `XMLType` instances dynamically using the `SYS_XMLGEN` and `SYS_XMLAGG` SQL functions.

This chapter contains the following topics:

- [Summary of XMLType Subprograms](#)

**See Also:**

- *Oracle XML DB Developer's Guide*

## Summary of XMLType Subprograms

Table 259–1 summarizes functions and procedures of the XMLType.

**Table 259–1 XMLTYPE Subprograms**

Method	Description
<a href="#">CREATENONSCHEMABASEDXML</a> on page 259-4	Creates a non schema based XML from the input schema based instance.
<a href="#">CREATESCHEMABASEDXML</a> on page 259-5	Creates a schema based XMLType instance from the non-schema based instance using the input schema URL.
<a href="#">CREATEXML</a> on page 6	Static function for creating and returning an XMLType instance.
<a href="#">EXISTSNODE</a> on page 259-8	Takes a XMLType instance and a XPath and returns 1 or 0 indicating if applying the XPath returns a non-empty set of nodes.
<a href="#">EXTRACT</a> on page 259-9	Takes a XMLType instance and an XPath, applies the XPath expression and returns the results as an XMLType.
<a href="#">GETBLOBVAL</a> on page 259-10	Returns the value of the XMLType instance as a BLOB
<a href="#">GETCLOBVAL</a> on page 259-11	Returns the value of the XMLType instance as a CLOB.
<a href="#">GETNAMESPACE</a> on page 259-12	Returns the namespace for the top level element in a schema based document.
<a href="#">GETNUMBERVAL</a> on page 259-13	Returns the value of the XMLType instance as a NUMBER. This is only valid if the input XMLType instance contains a simple text node and is convertible to a number.
<a href="#">GETROOTELEMENT</a> on page 259-14	Returns the root element of the input instance. Returns NULL if the instance is a fragment
<a href="#">GETSCHEMAURL</a> on page 259-15	Returns the XML schema URL if the input is an XML Schema based.
<a href="#">GETSTRINGVAL</a> on page 259-16	Returns the value of the XMLType instance as a string.
<a href="#">ISFRAGMENT</a> on page 259-17	Checks if the input XMLType instance is a fragment or not. A fragment is a XML instance, which has more than one root element.
<a href="#">ISSCHEMABASED</a> on page 259-18	Returns 1 or 0 indicating if the input XMLType instance is a schema based one or not.
<a href="#">ISSCHEMAVALID</a> on page 259-19	Checks if the input instance is schema valid according to the given schema URL.
<a href="#">ISSCHEMAVALIDATED</a> on page 259-20	Checks if the instance has been validated against the schema.
<a href="#">SCHEMAVALIDATE</a> on page 259-21	Validates the input instance according to the XML Schema. Raises error if the input instance is non-schema based.
<a href="#">SETSCHEMAVALIDATED</a> on page 259-22	Sets the schema valid flag to avoid costly schema validation.
<a href="#">TOOBJECT</a> on page 259-23	Converts the XMLType instance to an object type.
<a href="#">TRANSFORM</a> on page 259-24	Takes an XMLType instance and an associated stylesheet (which is also an XMLType instance), applies the stylesheet and returns the result as XML.



**Table 259-1 (Cont.) XMLTYPE Subprograms**

<b>Method</b>	<b>Description</b>
<a href="#">XMLTYPE</a> on page 259-25	Constructs an instance of the XMLType datatype. The constructor can take in the XML as a CLOB, VARCHAR2 or take in a object type.

## CREATENONSCHEMABASEDXML

Member function. Creates a non-schema based XML document from a schema based instance.

### **Syntax**

```
MEMBER FUNCTION CREATENONSCHEMABASEDXML  
return XMLType deterministic;
```

---

## CREATESCHEMABASEDXML

Member function. Creates a schema based XMLType instance from a non-schema based XMLType value. It uses either the supplied SCHEMA URL, or the SCHEMALOCATION attribute of the instance.

### Syntax

```
MEMBER FUNCTION createSchemaBasedXML(  
schema IN varchar2 := NULL)  
return XMLType deterministic;
```

---

Parameter	Description
schema	Optional XMLSchema URL used to convert the value to the specified schema..

---

## CREATEXML

Static function for creating and returning an `XMLType` instance. The string and clob parameters used to pass in the data must contain well-formed and valid XML documents. The options are described in the following table.

Syntax	Description
<pre> STATIC FUNCTION createXML(     xmlData IN varchar2) RETURN XMLType deterministic; </pre>	Creates the <code>XMLType</code> instance from a string.
<pre> STATIC FUNCTION createXML(     xmlData IN clob) RETURN XMLType deterministic; </pre>	Creates the <code>XMLType</code> instance from a CLOB.
<pre> STATIC FUNCTION createXML (     xmlData IN clob,     schema IN varchar2,     validated IN number := 0,     wellformed IN number := 0 ) RETURN XMLType deterministic; </pre>	This static function creates a schema-based <code>XMLType</code> instance using the specified schema and xml data parameters.
<pre> STATIC FUNCTION createXML (     xmlData IN varchar2,     schema IN varchar2,     validated IN number := 0,     wellformed IN number := 0) RETURN XMLType deterministic; </pre>	This static function creates a schema-based <code>XMLType</code> instance using the specified schema and xml data parameters.
<pre> STATIC FUNCTION createXML (     xmlData IN "&lt;ADT_1&gt;",     schema IN varchar2 := NULL,     element IN varchar2 := NULL,     validated IN NUMBER := 0) RETURN XMLType deterministic; </pre>	Creates an XML instance from an instance of an user-defined type.
<pre> STATIC FUNCTION createXML (     xmlData IN SYS_REFCURSOR,     schema in varchar2 := NULL,     element in varchar2 := NULL,     validated in number := 0) RETURN XMLType deterministic; </pre>	Creates an XML instance from a cursor reference. You can pass in any arbitrary SQL query as a CURSOR.

Syntax	Description
<pre> STATIC FUNCTION createXML (     xmlData IN AnyData,     schema IN varchar2 := NULL,     element IN varchar2 := NULL,     validated IN number := 0) RETURN sys.XMLType deterministic     parallel_enable </pre>	<p>Creates an XML instance from ANYDATA.</p> <p>If the ANYDATAinstance contains an ADT, the XMLType returned is the same as would be returned for a call directly on the ADT. If the ANYDATAcontains a scalar, the XMLType contains a leaf node with the scalar value. The element name for this node is taken from the optional element string if present, and is "ANYDATA" if it is not.</p>
<pre> STATIC FUNCTION createXML (     xmlData IN blob,     csid IN number,     schema IN varchar2,     validated IN number := 0,     wellformed IN number := 0) return sys.XMLType deterministic </pre>	<p>Creates an XML instance from a BLOB.</p>
<pre> STATIC FUNCTION createXML (     xmlData IN bfile,     csid IN number,     Schema IN varchar2,     validated IN number := 0,     wellformed IN number := 0) return sys.XMLType deterministic </pre>	<p>Creates an XML instance from a BFILE.</p>

Parameter	Description
xmlData	The actual data in the form of a BFILE, BLOB, CLOB, REF cursor, VARCHAR2 or object type.
schema	Optional Schema URL to be used to make the input conform to the given schema.
validated	Flag to indicate that the instance is valid according to the given XML Schema. (Default is 0)
wellformed	Flag to indicate that the input is well formed. If set, then the database would not do well formed check on the input instance. (Default is 0)
element	Optional element name in the case of the ADT_1 or REF CURSOR constructors. (Default is NULL)
CSID	The character set id of input XML data.

## EXISTSNODE

Member function. Checks if the node exists. If the XPath string is NULL or the document is empty, then a value of 0 is returned, otherwise returns 1. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION existsNode(     xpath IN varchar2) RETURN number deterministic;</pre>	Given an XPath expression, checks if the XPath applied over the document can return any valid nodes.
<pre>MEMBER FUNCTION existsNode(     xpath in varchar2,     nsmap in varchar2) RETURN number deterministic;</pre>	This member function uses the XPath expression with the namespace information and checks if applying the XPath returns any nodes or not.

Parameter	Description
xpath	The XPath expression to test.
nsmap	Optional namespace mapping.

## EXTRACT

Member function. Extracts an XMLType fragment and returns an XMLType instance containing the result node(s). If the XPath does not result in any nodes, then returns NULL. The options are described in the following table.

Syntax	Description
<pre>MEMBER FUNCTION extract(     xpath IN varchar2) RETURN XMLType deterministic;</pre>	Given an XPath expression, applies the XPath to the document and returns the fragment as an XMLType.
<pre>MEMBER FUNCTION extract(     xpath IN varchar2,     nsmmap IN varchar2) RETURN XMLType deterministic;</pre>	This member function applies the XPath expression and namespace mapping, over the XML data to return a XMLType instance containing the resultant fragment.

Parameter	Description
xpath	The XPath expression to apply.
nsmmap	Optional prefix to namespace mapping information.

## GETBLOBVAL

Member function. Returns a BLOB containing the serialized XML representation; if the BLOB returned is temporary, it must be freed after use.

### Syntax

```
MEMBER FUNCTION getBlobVal()  
RETURN Blob deterministic;
```



## GETCLOBVAL

Member function. Returns a CLOB containing the serialized XML representation; if the CLOB returned is temporary, it must be freed after use.

### Syntax

```
MEMBER FUNCTION getClobVal()  
RETURN clob deterministic;
```

## GETNAMESPACE

Member function. Returns the namespace of the top level element in the instance. Returns `NULL` if the input is a fragment or is a non-schema based instance.

### Syntax

```
MEMBER FUNCTION getNamespace  
return varchar2 deterministic;
```

## GETNUMBERVAL

Member function. Returns a numeric value, formatted from the text value pointed to by the `XMLType` instance. The `XMLType` must point to a valid text node that contains a numerical value. The options are described in the following table.

### Syntax

```
MEMBER FUNCTION getNumberVal()  
RETURN number deterministic;
```

## GETROOTELEMENT

Member function. Gets the root element of the `XMLType` instance. Returns `NULL` if the instance is a fragment.

### Syntax

```
MEMBER FUNCTION getRootElement  
return varchar2 deterministic;
```

## GETSCHEMAURL

Member function. Returns the XML Schema URL corresponding to the `XMLType` instance, if the `XMLType` instance is a schema-based document. Otherwise returns `NULL`.

### Syntax

```
MEMBER FUNCTION getSchemaURL  
return varchar2 deterministic;
```

## GETSTRINGVAL

Member function. Returns the document as a string. Returns a string containing the serialized XML representation, or in case of text nodes, the text itself. If the XML document is bigger than the maximum size of the `VARCHAR2`, which is 4000, then an error is raised at run time.

### Syntax

```
MEMBER FUNCTION getStringVal()  
RETURN varchar2 deterministic;
```

## ISFRAGMENT

Determines if the `XMLType` instance corresponds to a well-formed document, or a fragment. Returns 1 or 0 indicating if the `XMLType` instance contains a fragment or a well-formed document.

### Syntax

```
MEMBER FUNCTION isFragment()  
RETURN number deterministic;
```

## ISSCHEMABASED

Member function. Determines whether the `XMLType` instance is schema-based or not. Returns 1 or 0 depending on whether the `XMLType` instance is schema-based.

### Syntax

```
MEMBER FUNCTION isSchemaBased  
return number deterministic;
```



## ISSCHEMAVALID

Member function. Checks if the input instance is conformant to a specified schema. Does not change the validation status of the XML instance. If a XML Schema URL is not specified and the xml document is schema based, the conformance is checked against the XMLType instance's own schema.

### Syntax

```
member function isSchemaValid(  
schurl IN VARCHAR2 := NULL,  
elem IN VARCHAR2 := NULL)  
return NUMBER deterministic;
```

Parameter	IN / OUT	Description
schurl	(IN)	The URL of the XML Schema against which to check conformance.
elem	(IN)	Element of a specified schema, against which to validate. This is useful when we have a XML Schema which defines more than one top level element, and we want to check conformance against a specific one of these elements.

## ISSCHEMAVALIDATED

Member function. Returns the validation status of the `XMLType` instance -- tells if a schema based instance has been actually validated against its schema. Returns 1 if the instance has been validated against the schema, 0 otherwise.

### Syntax

```
MEMBER FUNCTION isSchemaValidated  
return NUMBER deterministic;
```

## SCHEMAVALIDATE

Member procedure. Validates the XML instance against its schema if it hasn't already been done. For non-schema based documents an error is raised. If validation fails an error is raised; else, the document's status is changed to validated.

### Syntax

```
MEMBER PROCEDURE schemaValidate(  
    self IF OUT NOCOPY XMLType);
```

Parameter	IN / OUT	Description
self	(OUT)	XML instance being validated against the schema.

## SETSCHEMAVALIDATED

Member function. Sets the `VALIDATION` state of the input XML instance.

### Syntax

```
MEMBER PROCEDURE setSchemaValidated(  
self IF OUT NOCOPY XMLType,  
    flag IN BINARY_INTEGER := 1);
```

Parameter	IN / OUT	Description
<code>self</code>	(OUT)	XML instance.
<code>flag</code>	(IN)	0 - NOT VALIDATED; 1 - VALIDATED (Default)

## TOOBJECT

Member procedure. Converts the XML value to an object type using the `XMLSCHEMA` mapping, if available. If a `SCHEMA` is not supplied or the input is a non-schema based XML, the procedure uses canonical mapping between elements and object type attributes.

### See Also:

- An in-depth discussion of this topic inside *Oracle XML DB Developer's Guide*

### Syntax

```
MEMBER PROCEDURE toObject(
  SELF in XMLType,
  object OUT "<ADT_1>",
  schema in varchar2 := NULL,
  element in varchar2 := NULL);
```

Parameter	IN / OUT	Description
<code>SELF</code>	(IN)	Instance to be converted. Implicit if used as a member procedure.
<code>object</code>	(IN)	Converted object. An object instance of the required type may be passed in to this function
<code>schema</code>	(IN)	Schema URL. The mapping of the <code>XMLType</code> instance to the converted object instance may be specified using a schema.
<code>element</code>	(IN)	Top-level element name. An XML Schema document does not specify the top-level element for a conforming XML instance document without this parameter.

## TRANSFORM

Member function. This member function transforms the XML data using the XSL stylesheet argument and the top-level parameters passed as a string of name=value pairs. If any of the arguments other than the parammap is NULL, then a NULL is returned.

### Syntax

```
MEMBER FUNCTION transform(  
xsl IN XMLType,  
parammap in varchar2 := NULL)  
RETURN XMLType deterministic;
```

Parameter	IN / OUT	Description
xsl	(IN)	The XSL stylesheet describing the transformation
parammap	(IN)	Top level parameters to the XSL - string of name=value pairs

## XMLTYPE

XMLType constructor. The options are described in the following table.

Syntax	Description
<pre> constructor function XMLType(   xmlData IN clob,   schema IN varchar2 := NULL,   validated IN number := 0,   wellformed IN Number := 0) return self as result deterministic; </pre>	<p>This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data parameters.</p>
<pre> constructor function XMLType(   xmlData IN varchar2,   schema IN varchar2 := NULL,   validated IN number := 0,   wellformed IN number := 0) return self as result deterministic; </pre>	<p>This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data parameters.</p>
<pre> constructor function XMLType (   xmlData IN "w&lt;ADT_1&gt;",   schema IN varchar2 := NULL,   element IN varchar2 := NULL,   validated IN number := 0) return self as result deterministic; </pre>	<p>This constructor function creates an optionally schema-based XMLType instance from the specified object type parameter.</p>
<pre> constructor function XMLType(   xmlData IN SYS_REFCURSOR,   schema in varchar2 := NULL,   element in varchar2 := NULL,   validated in number := 0) return self as result deterministic; </pre>	<p>This constructor function creates an optionally schema-based XMLType instance from the specified REF CURSOR parameter.</p>
<pre> constructor function XMLType(   xmlData IN AnyData,   schema IN varchar2 := NULL,   element IN varchar2 := NULL,   validated IN number := 0) return self as result deterministic parallel_enable </pre>	<p>This constructor function creates an optionally schema-based XMLType instance from the specified ANYDATA parameter.</p> <p>If the ANYDATA instance contains an ADT, the XMLType returned is the same as would be returned for a call directly on the ADT. If the ANYDATA contains a scalar, the XMLType contains a leaf node with the scalar value. The element name for this node is taken from the optional element string if present, and is "ANYDATA" if it is not.</p>

Syntax	Description
<pre> constructor function XMLType(   xmlData IN blob, csid IN number,   schema IN varchar2 := NULL,   validated IN number := 0,   wellformed IN number := 0) return self as result deterministic </pre>	<p>This constructor function creates an optionally schema-based XMLType instance from the specified BLOB parameter.</p>
<pre> constructor function XMLType(   xmlData IN bfile,   csid IN number,   schema IN varchar2 := NULL,   validated IN number := 0,   wellformed IN number := 0) return self as result deterministic </pre>	<p>This constructor function creates an optionally schema-based XMLType instance from the specified BFILE parameter.</p>

Parameter	Description
xmlData	The data in the form of a BFILE, BLOB, CLOB, REFS, VARCHAR2 or object type.
schema	Optional Schema URL to be used to make the input conform to the given schema.
validated	Indicates that the instance is valid to the given XML Schema.
wellformed	Indicates that the input is well formed. If set, then the database would not do well formed check on the input instance.
element	Optional element name in the case of the ADT_1 or REF CURSOR constructors. (Default is NULL)
CSID	The character set id of input XML data.



### A

ABORT procedure, 122-5  
ABORT\_GLOBAL\_INSTANTIATION procedure, 32-7  
ABORT\_REDEF\_TABLE procedure, 110-8  
ABORT\_SCHEMA\_INSTANTIATION procedure, 32-8  
ABORT\_TABLE\_INSTANTIATION procedure, 32-10  
ABORTED\_REQUEST\_THRESHOLD procedure, 132-6  
ACCEPT\_SQL\_PATCH Procedure, 137-11  
ACCEPT\_SQL\_PROFILE procedure, 139-19  
ACLCHECKPRIVILEGES function, 163-10  
ACTIVATE\_SUBSCRIPTION Procedure, 34-9  
ACTIVE\_INSTANCES procedure, 156-17  
ADD\_COLORED\_SQL Procedure, 161-10  
ADD\_COLUMN member procedure, 248-18  
ADD\_COLUMN procedure, 144-27  
ADD\_COOKIES procedure, 224-38  
ADD\_ELEMENTARY\_ATTRIBUTE procedure, 65-5, 123-6  
ADD\_EVENT procedure, 123-9  
ADD\_FILE procedure, 67-7  
ADD\_FILTER Procedure, 159-6, 160-6  
ADD\_FUNCTIONS procedure, 65-8, 123-11  
ADD\_GLOBAL\_PROPAGATION\_RULES procedure, 144-30  
ADD\_GLOBAL\_RULES procedure, 144-34  
ADD\_MESSAGE\_PROPAGATION\_RULE procedure, 144-39  
ADD\_MESSAGE\_RULE procedure, 144-43  
ADD\_PAIR member procedure, 256-13  
ADD\_PRIVILEGE Procedure, 94-6  
ADD\_RULE procedure, 123-12, 127-6  
ADD\_SCHEMA\_PROPAGATION\_RULES procedure, 144-46  
ADD\_SCHEMA\_RULES procedure, 144-50  
ADD\_SCHEME procedure, 89-7  
ADD\_SQLSET\_REFERENCE function, 139-22  
ADD\_SQLWKLD\_REF Procedure, 18-8  
ADD\_SQLWKLD\_STATEMENT Procedure, 18-9  
ADD\_STMT\_HANDLER procedure, 21-8  
ADD\_STMT\_TO\_HANDLER procedure, 147-11  
ADD\_SUBSCRIBER Procedure, 90-16  
ADD\_SUBSET\_PROPAGATION\_RULES procedure, 144-55  
ADD\_SUBSET\_RULES procedure, 144-59  
ADD\_TABLE\_PROPAGATION\_RULES procedure, 144-65  
ADD\_TABLE\_RULES procedure, 144-70  
ADD\_WARNING\_SETTING\_CAT procedure, 157-5  
ADD\_WARNING\_SETTING\_NUM procedure, 157-6  
ADD2MULTI procedure, 193-5  
ADDATTR member procedure

---

- of ANYTYPE TYPE, 242-6
- ADDHTTPEXPPIREMAPPING Procedure, 163-11
- ADDINSTANCE member procedure
  - of ANYDATASET TYPE, 241-4
- ADDMIMEMAPPING Procedure, 163-12
- ADDREPOSITORYRESCONFIG Procedure, 118-5
- ADDRESCONFIG Function, 118-6
- ADDRESS function
  - of HTF package, 182-16
- ADDRESS procedure
  - of HTP package, 183-15
- ADDSCHEMALOCMAPPING Procedure, 163-13
- ADDSERVLET Procedure, 163-14
- ADDSERVLETMAPPING Procedure, 163-15
- ADDSERVLETSECROLE Procedure, 163-16
- ADDXMLEXTENSION Procedure, 163-17
- ADM\_DROP\_CHUNKS Procedure, 101-12
- ADM\_DROP\_TASK Procedure, 101-13
- ADM\_GET\_TASK\_STATUS Procedure, 101-14
- ADM\_STOP\_TASK Procedure, 101-15
- ADMIN\_TABLES procedure, 112-10
- ADVISE\_COMMIT procedure, 152-5
- ADVISE\_NOTHING procedure, 152-6
- ADVISE\_ROLLBACK procedure, 152-7
- ADVISOR privilege, 41-3
- advisors
  - Oracle Streams, 145-1
- aggregation management, 41-4
- aggregation operators (OLAP), 41-24
- ALLOCATE\_UNIQUE procedure, 83-9
- ALTER\_AGENT Procedure, 90-20
- ALTER\_APPLY procedure, 21-10
- ALTER\_CAPTURE procedure, 32-11
- ALTER\_COMPILE procedure, 53-7
- ALTER\_EVALUATION\_CONTEXT procedure, 127-8
- ALTER\_FILE procedure, 67-9
- ALTER\_FILE\_GROUP procedure, 67-11
- ALTER\_JOB Procedure, 90-22
- ALTER\_MSGSYSTEM\_LINK Procedure for TIB/Rendezvous, 90-24
- ALTER\_MSGSYSTEM\_LINK Procedure for WebSphere MQ, 90-25
- ALTER\_PARAM Procedure, 37-3
- ALTER\_PROPAGATION procedure, 107-6
- ALTER\_PROPAGATION\_SCHEDULE Procedure, 90-26
- ALTER\_REWRITE\_EQUIVALENCE Procedure, 17-5
- ALTER\_RULE procedure, 127-11
- ALTER\_SQL\_PATCH Procedure, 137-13
- ALTER\_SQL\_PLAN\_BASELINE Function, 135-10
- ALTER\_SQL\_PROFILE procedure, 139-23
- ALTER\_STATS\_HISTORY\_RETENTION procedure, 141-20
- ALTER\_SUBSCRIBER Procedure, 90-27
- ALTER\_SYNC\_CAPTURE procedure, 32-16
- ALTER\_TABLE\_NOT\_REFERENCEABLE procedure, 53-8
- ALTER\_TABLE\_REFERENCEABLE procedure, 53-9
- ALTER\_VERSION procedure, 67-13
- AMATCH function, 191-7
- ANALYZE Function, 73-3
- ANALYZE Procedure, 31-4
- ANALYZE\_CURRENT\_PERFORMANCE procedure, 145-10
- ANALYZE\_DATABASE Procedure, 156-18
- ANALYZE\_DB Procedure, 16-5

---

ANALYZE\_INST Procedure, 16-6  
 ANALYZE\_PART\_OBJECT procedure, 156-19  
 ANALYZE\_PARTIAL Procedure, 16-7  
 ANALYZE\_SCHEMA Procedure, 156-20  
 ANCHOR function  
   of HTF package, 182-17  
 ANCHOR procedure  
   of HTP package, 183-16  
 ANCHOR2 function  
   of HTF package, 182-18  
 ANCHOR2 procedure  
   of HTP package, 183-17  
 anonymous PL/SQL blocks  
   dynamic SQL and, 136-3  
 AnyData datatype  
   queues  
     creating, 144-172  
     removing, 144-149  
 ANYDATA TYPE, 240-1  
 ANYDATASET TYPE, 241-1  
 ANYTYPE TYPE, 242-1  
 APEX\_APPLICATION package, 3-1  
 APEX\_APPLICATION package documentation, 3-2  
 APEX\_CUSTOM\_AUTH package, 2-1  
 APEX\_CUSTOM\_AUTH package documentation, 2-2  
 APEX\_ITEM package, 4-1  
 APEX\_ITEM package documentation, 4-2  
 APEX\_UTIL package, 5-1  
 APPEND Procedures, 82-19  
 APPENDCHILD function, 170-42  
 APPENDDATA procedure, 170-43  
 APPENDPATH Procedure, 163-18  
 APPENDRESCONFIG Procedure, 118-7  
 APPENDRESOURCEMETADATA Procedure, 163-19  
 APPLCETCLOSE function  
   of HTF package, 182-19  
 APPLCETCLOSE procedure  
   of HTP package, 183-18  
 APPLCETOPEN function  
   of HTF package, 182-20  
 APPLCETOPEN procedure  
   of HTP package, 183-19  
 apply process  
   altering, 21-10  
   apply user, 144-6  
   change handlers  
     setting, 21-38  
   conflict handlers  
     setting, 21-71  
   creating, 21-18, 144-34, 144-43, 144-50, 144-59, 144-70  
   DBMS\_APPLY\_ADM package, 21-1  
   DDL handler  
     setting, 21-10, 21-18  
   dropping, 21-27  
   enqueueing events, 21-46  
   error handlers  
     setting, 21-41  
   error queue  
     deleting errors, 21-25, 21-26  
     executing errors, 21-30, 21-31

---

- getting error messages, 21-34
- instantiation
  - global SCN, 21-50
  - schema SCN, 21-66
  - table SCN, 21-69
- message handler
  - setting, 21-10, 21-18
- parameters
  - allow\_duplicate\_rows, 21-57, 32-40
  - commit\_serialization, 21-58
  - compare\_key\_only, 21-59
  - disable\_on\_error, 21-59
  - disable\_on\_limit, 21-59
  - grouptransops, 21-59
  - ignore\_transaction, 21-60
  - max\_sga\_size, 21-60
  - maximum\_scn, 21-60
  - parallelism, 21-61
  - preserve\_encryption, 21-61
  - setting, 21-56
  - startup\_seconds, 21-62
  - time\_limit, 21-62
  - trace\_level, 21-62
  - transaction\_limit, 21-62
  - txn\_age\_spill\_threshold, 21-63
  - txn\_lcr\_spill\_threshold, 21-64
- precommit handler
  - setting, 21-10, 21-18
- procedure DML handlers
  - setting, 21-41
- rules
  - defining global, 144-34
  - defining message, 144-43
  - defining schema, 144-50
  - defining subset, 144-59
  - defining table, 144-70
  - for LCRs, 144-12
  - for user messages, 144-13
  - removing, 144-150
- specifying execution, 21-48
- starting, 21-77
- statement DML handlers
  - setting, 21-8, 21-38
- stopping, 21-78
- substitute key columns
  - setting, 21-53
- transparent data encryption, 21-61

apply user

- DBMS\_STREAMS\_ADM package, 144-7
- APPLY\$\_ENQUEUE, 21-46
- APPLY\$\_EXECUTE, 21-48
- AQ\$\_AGENT Type, 243-3
- AQ\$\_AGENT\_LIST\_T Type, 243-4
- AQ\$\_DESCRIPTOR Type, 243-5, 243-6
- AQ\$\_NTFN\_MSGID\_ARRAY Type, 243-7
- AQ\$\_POST\_INFO Type, 243-8
- AQ\$\_POST\_INFO\_LIST Type, 243-9
- AQ\$\_PURGE\_OPTIONS\_T Type, 243-10
- AQ\$\_RECIPIENT\_LIST\_T Type, 243-11
- AQ\$\_REG\_INFO Type, 243-12

---

AQ\$\_REG\_INFO\_LIST Type, 243-15  
AQ\$\_SUBSCRIBER\_LIST\_T Type, 243-16  
AREA function  
    of HTF package, 182-21  
AREA procedure  
    of HTP package, 183-20  
arrays  
    BIND\_ARRAY procedure, 136-9  
    bulk DML using DBMS\_SQL, 136-53  
ASA\_RECO\_ROW Record Type, 133-6  
ASA\_RECO\_ROW\_TB Table Type, 133-7  
ASA\_RECOMMENDATIONS Function, 133-9  
ASH\_GLOBAL\_REPORT\_HTML Function, 161-11  
ASH\_GLOBAL\_REPORT\_TEXT Function, 161-13  
ASH\_REPORT\_HTML Function, 161-15  
ASH\_REPORT\_TEXT Function, 161-18  
ASSIGN\_ACL Procedure, 94-8  
ASSIGN\_ATTRIBUTE\_SET procedure, 65-10  
ASSIGN\_WALLET\_ACL Procedure, 94-10  
ASSM\_SEGMENT\_VERIFY Procedure, 134-8  
ASSM\_TABLESPACE\_VERIFY Procedure, 134-10  
ATTACH\_SESSION procedure, 52-25  
ATTACH\_SIMPLE\_TABLESPACE procedure, 149-11  
ATTACH\_TABLESPACES procedure, 149-13  
attribute sets  
    dropping, 65-22  
AUTH Function and Procedure, 235-14  
AUTHORIZE function, 188-5  
AUTHORIZE\_DAD Procedure, 63-11  
AVAILABLE function, 237-12  
AWR\_BASELINE\_METRIC\_TYPE Object Type, 161-5  
AWR\_BASELINE\_METRIC\_TYPE\_TABLE Table Type, 161-6  
AWR\_DIFF\_REPORT\_HTML Function, 161-21  
AWR\_DIFF\_REPORT\_TEXT Function, 161-22  
AWR\_GLOBAL\_DIFF\_REPORT\_HTML Functions, 161-23  
AWR\_GLOBAL\_DIFF\_REPORT\_TEXT Functions, 161-24  
AWR\_GLOBAL\_REPORT\_HTML Functions, 161-25  
AWR\_GLOBAL\_REPORT\_TEXT Functions, 161-26  
AWR\_REPORT\_HTML function, 161-27  
AWR\_REPORT\_TEXT function, 161-28  
AWR\_SQL\_REPORT\_HTML Function, 161-29  
AWR\_SQL\_REPORT\_TEXT Function, 161-30

## **B**

BASE function  
    of HTF package, 182-22  
BASE procedure  
    of HTP package, 183-21  
BASE64\_DECODE function, 222-3  
BASE64\_ENCODE function, 222-4  
BASEFONT function  
    of HTF package, 182-23  
BASEFONT procedure  
    of HTP package, 183-22  
BEGIN\_PREPARE Procedure, 60-7  
BEGIN\_REQUEST function, 224-39  
BEGIN\_SQL\_BLOCK Procedure, 119-9  
BEGIN\_UPGRADE Procedure, 60-8  
BEGINCREATE static procedure  
    of ANYDATA TYPE, 240-7

---

- of ANYDATASET TYPE, 241-5
- of ANYTYPE TYPE, 242-3
- BFILE\_TABLE Table Type, 136-29
- BGSOUND function
  - of HTF package, 182-24
- BGSOUND procedure
  - of HTP package, 183-23
- BIG function
  - of HTF package, 182-25
- BIG procedure
  - of HTP package, 183-24
- BINARY\_DOUBLE\_TABLE Table Type, 136-30
- BINARY\_FLOAT\_TABLE Table Type, 136-31
- BIND\_ARRAY procedures, 136-52
- BIND\_INOUT\_VARIABLE Procedure, 75-6
- BIND\_INOUT\_VARIABLE\_RAW Procedure, 75-7
- BIND\_OUT\_VARIABLE Procedure, 75-8
- BIND\_OUT\_VARIABLE\_RAW Procedure, 75-9
- BIND\_VARIABLE Procedure, 75-10
- BIND\_VARIABLE procedures, 136-55
- BIND\_VARIABLE\_RAW Procedure, 75-11
- BIND\_VARIABLES function, 194-6
- BIT\_AND function, 232-7
- BIT\_COMPLEMENT function, 232-8
- BIT\_OR function, 232-9
- BIT\_XOR function, 232-10
- BLAS Level 1 (Vector-Vector Operations) Subprograms, 231-7
- BLAS Level 2 (Matrix-Vector Operations) Subprograms, 231-8
- BLAS Level 3 (Matrix-Matrix Operations) Subprograms, 231-10
- BLAS\_ASUM Functions, 231-18
- BLAS\_AXPY Procedures, 231-19
- BLAS\_COPY Procedures, 231-20
- BLAS\_DOT Functions, 231-21
- BLAS\_GBMV Procedures, 231-22
- BLAS\_GEMM Procedures, 231-25
- BLAS\_GEMV Procedures, 231-27
- BLAS\_GER Procedures, 231-29
- BLAS\_IAMAX Functions, 231-31
- BLAS\_NRM2 Functions, 231-32
- BLAS\_ROT Procedures, 231-33
- BLAS\_ROTG Procedures, 231-34
- BLAS\_SBMV Procedures, 231-42
- BLAS\_SCAL Procedure, 231-35
- BLAS\_SPMV Procedures, 231-36
- BLAS\_SPR Procedures, 231-38
- BLAS\_SPR2 Procedures, 231-40
- BLAS\_SWAP Procedure, 231-44
- BLAS\_SYMM Procedures, 231-45
- BLAS\_SYMV Procedures, 231-47
- BLAS\_SYR Procedures, 231-49
- BLAS\_SYR2 Procedures, 231-51
- BLAS\_SYRK Procedures, 231-56
- BLAS\_TBMV Procedures, 231-58
- BLAS\_TBSV Procedures, 231-60
- BLAS\_TPMV Procedures, 231-62
- BLAS\_TPSV Procedures, 231-64
- BLAS\_TRMM Procedures, 231-66
- BLAS\_TRMV Procedures, 231-69
- BLAS\_TRSM Procedures, 231-71
- BLAS\_TRSV Procedures, 231-74

---

BLOB\_TABLE Table Type, 136-32  
BLOCKQUOTECLOSE function  
    of HTP package, 182-26  
BLOCKQUOTECLOSE procedure  
    of HTP package, 183-25  
BLOCKQUOTEOPEN function  
    of HTP package, 182-27  
BLOCKQUOTEOPEN procedure  
    of HTP package, 183-26  
BODYCLOSE function  
    of HTP package, 182-28  
BODYCLOSE procedure  
    of HTP package, 183-27  
BODYOPEN function  
    of HTP package, 182-29  
BODYOPEN procedure  
    of HTP package, 183-28  
BOLD function  
    of HTP package, 182-30  
BOLD procedure  
    of HTP package, 183-29  
BR function  
    of HTP package, 182-31  
BR procedure  
    of HTP package, 183-30  
BREAKPOINT\_INFO Record Type, 52-16  
BROKEN procedure, 78-7  
BUILD procedure, 32-18, 41-19  
BUILD\_CHAIN\_ROWS\_TABLE procedure, 76-3  
BUILD\_EXCEPTIONS\_TABLE procedure, 65-12, 76-4  
BUILD\_PART\_INDEX procedure, 102-8  
BUILD\_SAFE\_REWRITE\_EQUIVALENCE Procedure, 17-6  
BYPASS Procedure, 121-5

## C

CALENDARPRINT procedures, 194-7  
CALIBRATE Function, 160-7  
CALIBRATE\_IO Procedure, 119-10  
CAN\_REDEF\_TABLE procedure, 110-9  
CANCEL\_ANALYSIS\_TASK Procedure, 138-6  
CANCEL\_DIAGNOSIS\_TASK Procedure, 137-14  
CANCEL\_REPLAY Procedure, 160-8  
CANCEL\_TASK Procedure, 18-12  
CANCEL\_TUNING\_TASK procedure, 139-25  
CANONICALIZE procedure, 156-21  
capture process  
    altering, 32-11  
    building a Streams data dictionary, 32-18  
    capture user, 144-5  
    creating, 144-34, 144-50, 144-59, 144-70  
    DBMS\_CAPTURE\_ADM package, 32-1  
    instantiation  
        aborting database preparation, 32-7  
        aborting schema preparation, 32-8  
        aborting table preparation, 32-10  
        preparing a database for, 32-35  
        preparing a schema for, 32-36  
        preparing a table for, 32-38  
    parameters  
        disable\_on\_limit, 32-40

---

- downstream\_real\_time\_mine, 32-41
- ignore\_transaction, 32-43
- ignore\_unsupported\_table, 32-43
- max\_sga\_size, 32-44
- maximum\_scn, 32-44
- message\_limit, 32-45
- message\_tracking\_frequency, 32-45
- parallelism, 32-45
- setting, 32-39
- skip\_autofiltered\_table\_ddl, 32-45
- startup\_seconds, 32-46
- time\_limit, 32-46
- trace\_level, 32-47
- write\_alert\_log, 32-47
- xout\_client\_exists, 32-47
- rules, 144-10
  - defining global, 144-34
  - defining schema, 144-50
  - defining subset, 144-59
  - defining table, 144-70
  - removing, 144-150
- starting, 32-49
- stopping, 32-50
- CAPTURE\_CURSOR\_CACHE\_SQLSET Procedure, 139-26
- CAST\_FROM\_BINARY\_DOUBLE function, 232-11
- CAST\_FROM\_BINARY\_FLOAT function, 232-13
- CAST\_FROM\_BINARY\_INTEGER function, 232-14
- CAST\_FROM\_NUMBER function, 232-15
- CAST\_TO\_BINARY\_DOUBLE function, 232-16
- CAST\_TO\_BINARY\_FLOAT function, 232-18
- CAST\_TO\_BINARY\_INTEGER function, 232-20
- CAST\_TO\_NUMBER function, 232-21
- CAST\_TO\_NVARCHAR2 function, 232-22
- CAST\_TO\_RAW function, 232-23
- CAST\_TO\_VARCHAR2 function, 232-24
- CB\$ table prefix, 41-4
- CELLSPRINT procedures, 194-8
- CENTER function
  - of HTF package, 182-32
- CENTER procedure
  - of HTP package, 183-31
- CENTERCLOSE function
  - of HTF package, 182-33
- CENTERCLOSE procedure
  - of HTP package, 183-32
- CENTEROPEN function
  - of HTF package, 182-34
- CENTEROPEN procedure
  - of HTP package, 183-33
- CFG\_GET function, 163-20
- CFG\_REFRESH procedure, 163-21
- CFG\_UPDATE procedure, 163-22
- Chain Condition Syntax, 128-78
- Change Data Capture
  - DBMS\_CDC\_PUBLISH package, 33-1
- change data capture
  - Oracle Streams
    - configuring, 144-90
- CHANGE functions and procedures, 191-9
- CHANGE procedure, 78-8



---

change tables  
  tablespaces created in, 33-26  
CHANGEOWNER Procedure, 163-23  
CHANGEPRIVILEGES Function, 163-24  
CHARARR Table Type, 100-12  
CHECK\_OBJECT procedure, 112-11  
CHECK\_PRIVILEGE Function, 94-11  
CHECK\_PRIVILEGE\_ACLID Function, 94-12  
CHECKACCESS Function, 47-40, 48-11  
CHECKIN function, 165-5  
CHECKOUT procedure, 165-6  
CHECKPRIVILEGES function, 163-25  
CHECKSPI Functions and Procedures, 47-41  
CHECKSUM functions, 190-6  
CHOOSE\_DATE procedure, 194-10  
CITE function  
  of HTF package, 182-35  
CITE procedure  
  of HTP package, 183-34  
CLEAN\_AUDIT\_TRAIL procedure, 27-12  
CLEANUP\_GATEWAY Procedure, 90-29  
CLEANUP\_INSTANTIATION\_SETUP procedure, 144-76  
CLEANUPUNUSEDBACKUPFILES Procedure, 49-12  
CLEAR Procedure, 169-15  
CLEAR\_ALL\_CONTEXT Procedure, 131-9  
CLEAR\_AUDIT\_TRAIL\_PROPERTY procedure, 27-14  
CLEAR\_CONTEXT Procedure, 131-10  
CLEAR\_COOKIES procedure, 224-41  
CLEAR\_EXPRSET\_STATS procedure, 65-13  
CLEAR\_IDENTIFIER Procedure, 131-11  
CLEAR\_LAST\_ARCHIVE\_TIMESTAMP procedure, 27-16  
CLEAR\_PENDING\_AREA procedure, 119-13  
CLEAR\_PLSQL\_TRACE procedure, 151-11  
CLEAR\_USED procedure, 99-6  
CLEARKEYCOLUMNLIST procedure, 175-7, 177-6  
CLEARUPDATECOLUMNLIST procedure, 175-8, 177-7  
CLIENT\_ID\_STAT\_DISABLE procedure, 92-3  
CLIENT\_ID\_STAT\_ENABLE procedure, 92-4  
CLIENT\_ID\_TRACE\_DISABLE procedure, 92-5  
CLIENT\_ID\_TRACE\_ENABLE procedure, 92-6  
CLOB\_TABLE Table Type, 136-33  
CLOB2FILE procedure, 180-6  
CLONE\_SIMPLE\_TABLESPACE procedure, 149-18  
CLONE\_TABLESPACES procedure, 149-20  
CLONENODE function, 170-44  
CLOSE Procedure, 82-20  
CLOSE\_ALL\_CONNECTIONS procedure, 237-14  
CLOSE\_CONNECTION Procedure, 235-16  
CLOSE\_CONNECTION procedure, 237-15  
CLOSE\_CURSOR Procedure, 75-12  
CLOSE\_CURSOR procedure, 136-57  
CLOSE\_DATA Function and Procedure, 235-17  
CLOSE\_DATABASE\_LINK Procedure, 131-12  
CLOSE\_ITERATOR procedure, 126-6  
CLOSE\_PERSISTENT\_CONN procedure, 224-42  
CLOSE\_PERSISTENT\_CONNS procedure, 224-43  
CLOSECONTEXT procedure, 171-5, 174-8, 175-9, 177-8  
COAD\_ADVICE\_REC type, 42-5  
COAD\_ADVICE\_T type, 42-5  
CODE function

---

- of HTF package, 182-36
- CODE procedure
  - of HTP package, 183-35
- COLLECT\_STATS procedure, 236-17
- collections
  - table items, 136-53
- column masking for VPD, 124-10
- COLUMN\_VALUE procedure, 136-58
- COLUMN\_VALUE\_LONG procedure, 136-61
- column-level VPD, 124-8
- COMMA\_TO\_TABLE procedures, 156-22
- COMMAND function and procedure, 235-18
- COMMAND\_REPLIES function, 235-19
- COMMENT function
  - of HTF package, 182-37
- COMMENT procedure
  - of HTP package, 183-36
- COMMIT procedure, 152-8
- COMMIT\_COMMENT procedure, 152-9
- COMMIT\_FORCE procedure, 152-10
- COMPARE Function, 35-16
- COMPARE function, 232-25
- COMPARE Functions, 82-21
- COMPARE\_OLD\_VALUES procedure, 21-16
- COMPARE\_PERIOD\_REPORT Procedure, 160-9
- COMPARE\_SQLSET\_REPORT Function, 160-10
- COMPARISON\_TYPE type, 35-14
- COMPATIBLE\_10\_1 function, 143-9, 248-38
- COMPATIBLE\_10\_2 function, 143-8, 248-38
- COMPATIBLE\_11\_1 function, 143-7
- COMPATIBLE\_11\_2 function, 143-6
- COMPATIBLE\_9\_2 function, 143-10, 248-38
- COMPILE\_FROM\_REMOTE procedure, 81-6
- COMPILE\_SCHEMA procedure, 156-23
- COMPILESCHEMA procedure, 176-10
- CONCAT function, 232-26
- CONDITION\_REF function, 123-14
- CONFIGURE Procedure, 135-12
- CONFIGURE\_POOL Procedure, 37-4
- CONFIGUREAUTOSYNC procedure, 167-8
- conflicts
  - detection
    - stopping, 21-16
- CONNECTION record type, 235-6
- constants
  - DBMS\_MGWMSG package, 91-4
- CONSUME\_EVENT Function, 123-15
- CONSUME\_PRIM\_EVENT Function, 123-17
- CONTAINS\_HOST Function, 95-5
- CONTINUE function, 52-26
- CONVERGE Procedure, 35-18
- CONVERT function, 83-11, 171-6, 232-27
- CONVERT\_ANYDATA\_TO\_LCR\_DDL function, 143-11
- CONVERT\_ANYDATA\_TO\_LCR\_ROW function, 143-12
- CONVERT\_LCR\_TO\_XML function, 143-13
- CONVERT\_LONG\_TO\_LOB\_CHUNK member procedure, 248-20
- CONVERT\_RAW\_VALUE procedures, 141-21
- CONVERT\_RAW\_VALUE\_NVARCHAR procedure, 141-22
- CONVERT\_RAW\_VALUE\_ROWID procedure, 141-23
- CONVERT\_XML\_TO\_LCR function, 143-14

---

CONVERTTOBLOB procedure, 82-23  
 CONVERTTOCLOB Procedure, 82-26  
 COPIES function, 232-28  
 COPY Procedures, 82-29  
 COPY\_ATTRIBUTE\_SET procedure, 65-14  
 COPY\_DBFS\_LINK Procedures, 82-31  
 COPY\_FILE procedure, 68-5  
 COPY\_FROM\_DBFS\_LINK, 82-32  
 COPY\_TABLE\_DEPENDENTS procedure, 110-10  
 COPY\_TABLE\_STATS Procedure, 141-24  
 COPYDEFAULTTOKENTABLESET Procedure, 40-8  
 COPYEVOLVE procedure, 176-11  
 CQ\_NOTIFICATION\$\_DESCRIPTOR Object Type, 38-16  
 CQ\_NOTIFICATION\$\_QUERY Object Type, 38-17  
 CQ\_NOTIFICATION\$\_QUERY\_ARRAY Object (Array) Type, 38-18  
 CQ\_NOTIFICATION\$\_REG\_INFO Object Type, 38-23  
 CQ\_NOTIFICATION\$\_ROW Object Type, 38-21  
 CQ\_NOTIFICATION\$\_TABLE Object Type, 38-19  
 CQ\_NOTIFICATION\$\_TABLE\_ARRAY Object (Array) Type, 38-20  
 CQ\_NOTIFICATION\_QUERYID Function, 38-27  
 CREATE ANY DIMENSION privilege, 41-3  
 CREATE ANY MATERIALIZED VIEW privilege, 41-3  
 CREATE DIMENSION privilege, 41-3  
 CREATE MATERIALIZED VIEW privilege, 41-3  
 CREATE PACKAGE BODY command, 1-5  
 CREATE PACKAGE command, 1-5  
 CREATE SESSION privilege, 41-3  
 CREATE\_ACL Procedure, 94-13  
 CREATE\_AFFECTED\_TABLE Procedure, 60-9  
 CREATE\_AGENT Procedure, 90-32  
 CREATE\_ALTER\_TYPE\_ERROR\_TABLE procedure, 156-24  
 CREATE\_ANALYSIS\_TASK Functions, 138-7  
 CREATE\_APPLY procedure, 21-18  
 CREATE\_ATTRIBUTE\_SET procedure, 65-15  
 CREATE\_BASELINE Function & Procedure, 161-31  
 CREATE\_BASELINE\_TEMPLATE Procedures, 161-33  
 CREATE\_BIN\_CAT procedure, 45-19  
 CREATE\_BIN\_NUM procedure, 45-21  
 CREATE\_CAPTURE procedure  
     capture process  
         creating, 32-19  
 CREATE\_CATEGORY function, 89-11  
 CREATE\_CATEGORY Procedure, 119-14  
 CREATE\_CHUNKS\_BY\_NUMBER\_COL Procedure, 101-17  
 CREATE\_CHUNKS\_BY\_ROWID Procedure, 101-18  
 CREATE\_CHUNKS\_BY\_SQL Procedure, 101-19  
 CREATE\_CLIP procedure, 45-23  
 CREATE\_COL\_REM procedure, 45-25  
 CREATE\_COMPARISON Procedure, 35-20  
 CREATE\_CONDITIONS\_TABLE procedure, 123-19  
 CREATE\_CONSUMER\_GROUP procedure, 119-15  
 CREATE\_DAD Procedure, 63-12  
 CREATE\_DIAGNOSIS\_TASK Functions, 137-15  
 CREATE\_ERROR\_LOG Procedure, 64-5  
 CREATE\_ERROR\_TABLE Procedure, 60-10  
 CREATE\_EVALUATION\_CONTEXT procedure, 127-13  
 CREATE\_EVENT\_STRUCT procedure, 123-21  
 CREATE\_EXPFIL\_INDEXES procedure, 123-22  
 CREATE\_EXTENDED\_STATS Function, 141-26  
 CREATE\_FILE Procedure, 18-14

---

CREATE\_FILE\_GROUP procedure, 67-15  
CREATE\_FILTER\_SET Procedure, 160-11  
CREATE\_INDEX\_COST procedure, 133-10  
CREATE\_INTERFACE procedure, 123-24  
CREATE\_JOB Procedure, 90-34  
CREATE\_MISS\_CAT procedure, 45-26  
CREATE\_MISS\_NUM procedure, 45-28  
CREATE\_MSGSYSTEM\_LINK Procedure for TIB/Rendezvous, 90-38  
CREATE\_MSGSYSTEM\_LINK Procedure for WebSphere MQ, 90-39  
CREATE\_MVIEW function, 41-8, 41-36  
CREATE\_NORM\_LIN procedure, 45-30  
CREATE\_OBJECT Procedure, 18-15  
CREATE\_OBJECT\_DEPENDENCY procedure, 21-24  
CREATE\_OR\_REPLACE\_VIEW procedure  
    of DBMS\_HS\_PARALLEL package, 74-4  
CREATE\_OUTLINE procedure, 99-7  
CREATE\_PENDING\_AREA procedure, 119-16  
CREATE\_PIPE function, 103-19  
CREATE\_PLAN procedure, 119-18  
CREATE\_PLAN\_DIRECTIVE procedure, 119-19  
CREATE\_PROPAGATION procedure, 107-8  
CREATE\_PURGE\_JOB procedure, 27-17  
CREATE\_REQUEST\_CONTEXT Function, 224-45  
CREATE\_RULE procedure, 127-15  
CREATE\_RULE\_CLASS procedure, 123-26  
CREATE\_RULE\_SET procedure, 127-17  
CREATE\_SERVICE procedure, 130-10  
CREATE\_SIMPLE\_PLAN procedure, 119-23  
CREATE\_SNAPSHOT function and procedure, 161-34  
CREATE\_SQL\_PLAN\_BASELINE Procedure, 139-28  
CREATE\_SQLSET procedure, 139-29  
CREATE\_SQLWKLK Procedure, 18-17  
CREATE\_STAT\_TABLE procedure, 141-28  
CREATE\_STGTAB\_BASELINE Procedure, 135-13  
CREATE\_STGTAB\_SQLPATCH Procedure, 137-17  
CREATE\_STGTAB\_SQLPROF Procedure, 139-30  
CREATE\_STGTAB\_SQLSET Procedure, 139-31  
CREATE\_STMT\_HANDLER procedure, 147-12  
CREATE\_SYNC\_CAPTURE procedure  
    synchronous capture  
        creating, 32-29  
CREATE\_TABLE\_COST procedures, 133-11  
CREATE\_TABLE\_COST\_COLINFO Object Type, 133-5  
CREATE\_TABLE\_TEMPLAGE procedure  
    of DBMS\_HS\_PARALLEL package, 74-6  
CREATE\_TASK Procedure, 101-16  
CREATE\_TASK Procedures, 18-18  
CREATE\_TRANSFORMATION procedure, 153-3  
CREATE\_TRIGGER\_TABLE Procedure, 60-11  
CREATE\_TUNING\_TASK functions, 139-33  
CREATE\_VERSION procedure, 67-17  
CREATE\_WRAPPED Procedure, 53-10  
CREATEATTRIBUTE function, 170-45  
CREATEBUCKET Procedure, 49-13  
CREATEDATASECTION function, 170-46  
CREATECOMMENT function, 170-47  
CREATEDATASTOREPREF procedure, 167-9  
CREATEDATEINDEX Procedure, 172-6  
CREATEDIRECTORY Procedures, 47-42, 48-12  
CREATEDOCUMENT function, 170-48

---

CREATEDOCUMENTFRAGMENT function, 170-49  
CREATEELEMENT function, 170-50  
CREATEENTITYREFERENCE function, 170-51  
CREATEFILE Procedures, 47-43, 48-13  
CREATEFILESYSYEM Procedure, 50-7  
CREATEFILTERPREF procedure, 167-10  
CREATEFOLDER function, 163-26  
CREATEINDEX procedure, 167-11  
CREATELEXERPREF procedure, 167-12  
CREATELINK Procedures, 47-44, 48-14  
CREATENONSCHEMABASEDXML function, 259-4  
CREATENUMBERINDEX Procedure, 172-7  
CREATEOIDPATH function, 163-27  
CREATEPREFERENCES procedure, 167-13  
CREATEPROCESSINGINSTRUCTION function, 170-52  
CREATEREFERENCE Procedures, 47-45, 48-15  
CREATEREPOSITORYXMLINDEX Procedure, 164-6  
CREATERESOURCE function, 163-28  
CREATESCHEMABASEDXML function, 259-5  
CREATESECTIONGROUPPREF procedure, 167-14  
CREATESTOPLISTPREF procedure, 167-15  
CREATESTORAGEPREF procedure, 167-16  
CREATESTORE Procedure, 49-14, 50-9  
CREATETEMPORARY Procedures, 82-33  
CREATETEXTNODE function, 170-53  
CREATEURI function, 245-10, 245-19, 245-28  
CREATEWORLDLISTPREF procedure, 167-17  
CREATEXML function, 259-6  
creating  
    packages, 1-5  
CTX\_ADM package, 6-1  
CTX\_ADM package documentation, 6-2  
CTX\_CLS package, 7-1  
CTX\_CLS package documentation, 7-2  
CTX\_DDL package, 8-1  
CTX\_DDL package documentation, 8-2  
CTX\_DOC package documentation, 9-2  
CTX\_ENTITY package, 10-1  
CTX\_ENTITY package documentation, 10-2  
CTX\_OUTPUT package documentation, 11-2  
CTX\_QUERY package documentation, 12-2  
CTX\_REPORT package documentation, 13-2  
CTX\_THES package documentation, 14-2  
CTX\_ULEXER package documentation, 15-2  
cube aggregation, 41-19  
Cube Build log, 43-8  
Cube Dimension Compile log, 43-10  
cube log verbosity levels, 43-5  
cube logging targets, 43-4  
cube logging types, 43-3  
cube maintenance, 41-1  
cube materialized views, 41-4  
    optimizing, 42-1  
Cube Operations log, 43-11  
cube refresh, 41-19  
Cube Rejected Records Log, 43-12  
CUBE\_BUILD\_LATEST view, 41-27  
CUBE\_BUILD\_REPORT view, 41-27  
CUBE\_BUILD\_REPORT\_LATEST view, 41-27  
CUBE\_UPGRADE\_INFO table, 41-52

---

cubes  
optimizer statistics, 31-1  
CURRENT\_INSTANCE function, 156-25  
cursors  
DBMS\_SQL package, 136-7

## D

data dictionary  
removing Streams information, 144-146  
DATA function and procedure, 235-20  
data mining  
algorithms, 44-3  
anomaly detection, 44-4  
association rules, 44-3  
attribute importance, 44-3  
attributes, 44-5  
automated, 104-1  
Automatic Data Preparation, 44-11  
classification, 44-3, 44-4  
clustering, 44-3, 44-4  
confusion matrix, 44-34  
cost matrix, 44-25  
creating a model, 44-50  
data transformation, 44-11, 44-50, 44-95, 44-97  
feature extraction, 44-4  
lift, 44-40  
mining functions, 44-3, 44-9  
PMML, 44-90  
regression, 44-3, 44-4  
ROC, 44-45  
schema objects, 44-5  
scoring, 44-1, 44-3, 44-30, 44-65  
supervised, 44-3  
transactional data, 44-14  
transformations, 45-1  
unsupervised, 44-3  
DATA\_BLOCK\_ADDRESS\_BLOCK function, 156-26  
DATA\_BLOCK\_ADDRESS\_FILE function, 156-27  
database  
locking  
OWA\_OPT\_LOCK package, 190-3  
database tables  
creating for DBMS\_TRACE, 151-7  
database\_role attribute (Scheduler job), 128-133  
DATABASE\_TRACE\_DISABLE Procedure, 92-7  
DATABASE\_TRACE\_ENABLE Procedure, 92-8  
datatypes  
DBMS DESCRIBE, 57-12  
PL/SQL  
numeric codes for, 57-9  
DATE\_TABLE Table Type, 136-34  
DB\_CONNECT\_INFO Procedure, 90-40  
DB\_VERSION procedure, 156-28  
DBFS\_LINK\_GENERATE\_PATH Functions, 82-34  
DBLINK\_ARRAY Table Type, 156-8  
DBMS, 170-41  
DBMS\_NETWORK\_ACL\_ADMIN package, 94-1, 95-1  
DBMS\_ALERT package, 19-1  
DBMS\_APPLICATION\_INFO package, 20-1  
DBMS\_APPLY\_ADM package, 21-1

---

DBMS\_AQADM Constants, 23-3  
DBMS\_AQELM package, 24-1  
DBMS\_AQIN package, 25-1  
DBMS\_AUDIT\_MGMT package, 27-1  
    CLEAN\_AUDIT\_TRAIL procedure, 27-12  
    CLEAR\_AUDIT\_TRAIL\_PROPERTY procedure, 27-14  
    CLEAR\_LAST\_ARCHIVE\_TIMESTAMP procedure, 27-16  
    CREATE\_PURGE\_JOB procedure, 27-17  
    DEINIT\_CLEANUP procedure, 27-19  
    DROP\_PURGE\_JOB procedure, 27-20  
    GET\_AUDIT\_COMMIT\_DELAY function, 27-21  
    INIT\_CLEANUP procedure, 27-22  
    IS\_CLEANUP\_INITIALIZED function, 27-24  
    SET\_AUDIT\_TRAIL\_LOCATION procedure, 27-25  
    SET\_AUDIT\_TRAIL\_PROPERTY procedure, 27-27  
    SET\_LAST\_ARCHIVE\_TIMESTAMP procedure, 27-30  
    SET\_PURGE\_JOB\_INTERVAL procedure, 27-32  
    SET\_PURGE\_JOB\_STATUS procedure, 27-33  
DBMS\_AUTO\_TASK\_ADMIN package, 29-1  
DBMS\_AW\_STATS package, 31-1  
DBMS\_CAPTURE package, 243-1  
DBMS\_CAPTURE\_ADM package, 32-1  
DBMS\_CDC\_PUBLISH package  
    ALTER\_AUTOLOG\_CHANGE\_SOURCE procedure, 33-8  
    ALTER\_CHANGE\_SET procedure, 33-10  
    ALTER\_CHANGE\_TABLE procedure, 33-13  
    ALTER\_HOTLOG\_CHANGE\_SOURCE procedure, 33-16  
    CREATE\_AUTOLOG\_CHANGE\_SOURCE procedure, 33-18  
    CREATE\_CHANGE\_SET procedure, 33-20  
    CREATE\_CHANGE\_TABLE procedure, 33-23  
    CREATE\_HOTLOG\_CHANGE\_SOURCE procedure, 33-27  
    DROP\_CHANGE\_SET procedure, 33-29  
    DROP\_CHANGE\_SOURCE procedure, 33-30  
    DROP\_CHANGE\_TABLE procedure, 33-31  
    DROP\_SUBSCRIPTION procedure, 33-32  
    Oracle GoldenGate, 33-1  
    PURGE procedure, 33-34  
    PURGE\_CHANGE\_SET procedure, 33-35  
    PURGE\_CHANGE\_TABLE procedure, 33-36  
DBMS\_CDC\_SUBSCRIBE package  
    ACTIVATE\_SUBSCRIPTION procedure, 34-13  
    CREATE\_SUBSCRIPTION procedure, 34-10  
    DROP\_SUBSCRIBER\_VIEW procedure, 34-5  
    DROP\_SUBSCRIPTION procedure, 34-12  
    EXTEND\_WINDOW procedure, 34-13  
    GET\_SUBSCRIPTION\_HANDLE procedure, 34-5  
    Oracle GoldenGate, 34-1  
    PREPARE\_SUBSCRIBER\_VIEW procedure, 34-5  
    PURGE\_WINDOW procedure, 34-15  
    SUBSCRIBE procedure, 34-16  
DBMS\_CHANGE\_NOTIFICATION package, 38-1  
DBMS\_COMPARISON package, 35-1  
    constants, 35-6  
    data structures, 35-13  
    index columns, 35-10  
        lead index column, 35-11  
    requirements, 35-9  
    subprograms, 35-15  
    using, 35-2  
    views, 35-8

---

DBMS\_CONNECTION\_POOL package, 37-1  
 DBMS\_CONTENT\_CONTEXT\_T Object Type, 244-6  
 DBMS\_CSX\_ADMIN package, 40-1  
 DBMS\_CUBE package, 41-1  
 DBMS\_CUBE\_ADVISE package, 42-1  
 DBMS\_CUBE\_LOG package, 43-1  
 DBMS\_DATA\_MINING package, 44-1
 

- ADD\_COST\_MATRIX procedure, 44-25
- algorithms, 44-3
- ALL\_MINING\_MODEL\_ATTRIBUTES data dictionary view, 44-5
- ALL\_MINING\_MODEL\_SETTINGS data dictionary view, 44-6
- ALL\_MINING\_MODELS data dictionary view, 44-5
- ALTER\_REVERSE\_EXPRESSION procedure, 44-27
- APPLY procedure, 44-30
- Automatic Data Preparation, 44-11
- COMPUTE\_CONFUSION\_MATRIX procedure, 44-34
- COMPUTE\_LIFT procedure, 44-40
- COMPUTE\_ROC procedure, 44-45
- CREATE\_MODEL procedure, 44-50
- data transformation, 44-11, 44-50, 44-97
- data types, 44-20
- deprecated subprograms, 44-8
- DROP\_MODEL procedure, 44-54
- EXPORT\_MODEL procedure, 44-55
- GET\_ASSOCIATION\_RULES function, 44-58
- GET\_FREQUENT\_ITEMSETS function, 44-63
- GET\_MODEL\_COST\_MATRIX function, 44-65
- GET\_MODEL\_DETAILS\_AI function, 44-69
- GET\_MODEL\_DETAILS\_GLM function, 44-71
- GET\_MODEL\_DETAILS\_GLOBAL function, 44-74
- GET\_MODEL\_DETAILS\_KM function, 44-77
- GET\_MODEL\_DETAILS\_NB function, 44-80
- GET\_MODEL\_DETAILS\_NMF function, 44-82
- GET\_MODEL\_DETAILS\_OC function, 44-84
- GET\_MODEL\_DETAILS\_SVM function, 44-88
- GET\_MODEL\_DETAILS\_XML function, 44-90
- GET\_MODEL\_TRANSFORMATIONS function, 44-95
- GET\_TRANSFORM\_LIST procedure, 44-97
- IMPORT\_MODEL procedure, 44-100
- mining functions, 44-9
- overview, 44-3
- PMML, 44-90
- RANK\_APPLY procedure, 44-104
- REMOVE\_COST\_MATRIX procedure, 44-107
- RENAME\_MODEL procedure, 44-108
- scoring, 44-1, 44-3, 44-30, 44-39, 44-42, 44-65, 44-107
- security model, 44-7
- settings
  - algorithm names, 44-10
  - decision tree, 44-15
  - GLM, 44-16
  - global, 44-13
  - k-Means, 44-16
  - mining functions, 44-11
  - mining models, 44-10
  - Naive Bayes, 44-17
  - NMF, 44-18
  - O-Cluster, 44-18
  - SVM, 44-18
- subprograms, 44-23



---

- supermodels, 44-11
- transactional data, 44-14
- DBMS\_DATA\_MINING\_TRANSFORM
  - data types, 45-15
  - introduction, 45-2
  - package, 45-1
  - subprograms, 45-17
- DBMS\_DATAPUMP package, 46-1
  - ADD\_FILE procedure, 46-13
  - ATTACH function, 46-16
  - DATA\_FILTER procedure, 46-18
  - DETACH procedure, 46-23
  - GET\_DUMPFILE\_INFO procedure, 46-24
  - GET\_STATUS procedure, 46-26
  - LOG\_ENTRY procedure, 46-29
  - METADATA\_FILTER procedure, 46-30
  - METADATA\_REMAP procedure, 46-33
  - METADATA\_TRANSFORM procedure, 46-36
  - OPEN function, 46-39
  - roles used by, 46-4
  - SET\_PARALLEL procedure, 46-42
  - SET\_PARAMETER procedure, 46-44
  - START\_JOB procedure, 46-53
  - STOP\_JOB procedure, 46-55
  - types used by, 46-6
  - WAIT\_FOR\_JOB procedure, 46-57
- DBMS\_DB\_VERSION package, 51-1
- DBMS\_DBFS\_CONTENT package, 47-1
- DBMS\_DBFS\_CONTENT\_LIST\_ITEM\_T Object Type, 244-7
- DBMS\_DBFS\_CONTENT\_LIST\_ITEMS\_T Table Type, 244-9
- DBMS\_DBFS\_CONTENT\_PROPERTIES\_T Table Type, 244-10
- DBMS\_DBFS\_CONTENT\_PROPERTY\_T Object Type, 244-8
- DBMS\_DBFS\_CONTENT\_RAW\_T Table Type, 244-11
- DBMS\_DBFS\_CONTENT\_SPI package, 48-1
- DBMS\_DBFS\_HS package, 49-1
- DBMS\_DBFS\_SFS package, 50-1
- DBMS\_DDL package, 53-1
- DBMS\_DEBUG package, 52-1
- DBMS\_DEFER package documentation, 54-2
- DBMS\_DEFER\_QUERY package documentation, 55-2
- DBMS\_DEFER\_SYS package documentation, 56-2
- DBMS\_DESCRIBE package, 57-1
- DBMS\_DG Package
  - INITIATE\_FS\_FAILOVER procedure, 58-5
- DBMS\_DG package, 58-1
  - using, 58-2
- DBMS\_DIMENSION package, 59-1
- DBMS\_DISTRIBUTED\_TRUST\_ADMIN package, 61-1
- DBMS\_EDITIONS\_UTILITIES package, 62-1, 62-3
- DBMS\_EPG package, 63-1, 63-3
- DBMS\_EXPFIL package, 65-1
  - ADD\_ELEMENTARY\_ATTRIBUTE, 65-5
  - ADD\_FUNCTIONS, 65-8
  - ASSIGN\_ATTRIBUTE\_SET, 65-10
  - BUILD\_EXCEPTIONS\_TABLE, 65-12
  - CLEAR\_EXPRSET\_STATS, 65-13
  - COPY\_ATTRIBUTE\_SET, 65-14
  - CREATE\_ATTRIBUTE\_SET, 65-15
  - DEFAULT\_INDEX\_PARAMETERS, 65-17
  - DEFAULT\_XPINDEX\_PARAMETERS, 65-19

---

- DEFRAG\_INDEX, 65-21
- DROP\_ATTRIBUTE\_SET, 65-22
- GET\_EXPRSET\_STATS, 65-23
- GRANT\_PRIVILEGE, 65-24
- INDEX\_PARAMETERS, 65-25
- MODIFY\_OPERATOR\_LIST, 65-27
- REVOKE\_PRIVILEGE, 65-28
- SYNC\_TEXT\_INDEXES, 65-29
- UNASSIGN\_ATTRIBUTE\_SET, 65-30
- VALIDATE\_EXPRESSIONS, 65-31
- XPINDEX\_PARAMETERS, 65-32
- DBMS\_FGA package, 66-1
- DBMS\_FILE\_GROUP package, 67-1
  - constants, 67-5
- DBMS\_FILE\_TRANSFER package, 68-1
- DBMS\_FLASHBACK package, 69-1
- DBMS\_FREQUENT\_ITEMSET package, 71-1
- DBMS\_HPROF package, 73-1
- DBMS\_HS\_PARALLEL package, 74-1
  - CREATE\_OR\_REPLACE\_VIEW procedure, 74-4
  - CREATE\_TABLE\_TEMPLATE procedure, 74-6
  - DROP\_VIEW procedure, 74-7
  - LOAD\_TABLE procedure, 74-8
- DBMS\_HS\_PASSTHROUGH package, 75-1
- DBMS\_IOT package, 76-1
- DBMS\_JAVA package documentation, 77-2
- DBMS\_JOB package, 78-1
- DBMS\_LDAP package documentation, 79-2
- DBMS\_LDAP\_UTL package documentation, 80-2
- DBMS\_LIBCACHE package, 81-1
- DBMS\_LOB package, 82-1
- DBMS\_LOCK package, 83-1
- DBMS\_LOGMNR package, 84-1
  - ADD\_LOGFILE procedure, 84-10
  - COLUMN\_PRESENT function, 84-12
  - END\_LOGMNR procedure, 84-14
  - MINE\_VALUE function, 84-15
  - REMOVE\_LOGFILE procedure, 84-17
  - START\_LOGMNR procedure, 84-18
- DBMS\_LOGMNR\_D package, 85-1
  - BUILD procedure, 85-6
  - SET\_TABLESPACE procedure, 85-9
- DBMS\_LOGSTDBY package, 86-1
  - APPLY\_SET procedure, 86-7
  - APPLY\_UNSET procedure, 86-10
  - BUILD procedure, 86-11
  - INSTANTIATE\_TABLE procedure, 86-12
  - MAP\_PRIMARY\_SCN function, 86-15
  - overview of managing SQL Apply, 86-3
  - PREPARE\_FOR\_NEW\_PRIMARY procedure, 86-16
  - PURGE\_SESSION procedure, 86-18
  - SET\_TABLESPACE procedure, 86-20
  - SKIP procedure, 86-21
  - SKIP\_ERROR procedure, 86-29
  - SKIP\_TRANSACTION procedure, 86-33
  - UNSKIP procedure, 86-35
  - UNSKIP\_ERROR procedure, 86-37
  - UNSKIP\_TRANSACTION procedure, 86-39
  - using, 86-2
- DBMS\_METADATA package, 87-1

---

- ADD\_TRANSFORM function, 87-12
- CLOSE procedure, 87-16
- CONVERT functions and procedures, 87-17
- GET\_DDL function, 87-22
- GET\_QUERY function, 87-26
- GET\_XML function, 87-22
- OPEN function, 87-27
- OPENW function, 87-33
- PUT function, 87-34
- security, 87-4
- SET\_COUNT procedure, 87-36
- SET\_FILTER procedure, 87-37
- SET\_PARSE\_ITEM procedure, 87-47
- SET\_REMAP\_PARAM procedure, 87-50
- SET\_TRANSFORM\_PARAM procedure, 87-50
- DBMS\_METADATA\_DIFF package, 88-1, 88-3
  - ADD\_DOCUMENT procedure, 88-9
  - FETCH\_CLOB functions and procedures, 88-10
  - OPENC function, 88-8
- DBMS\_MGD\_ID\_UTL package, 89-1
  - ADD\_SCHEME procedure, 89-7
  - CREATE\_CATEGORY function, 89-11
  - EPC\_TO\_ORACLE\_SCHEME function, 89-12
  - GET\_CATEGORY\_ID function, 89-15
  - GET\_COMPONENTS function, 89-16
  - GET\_ENCODINGS function, 89-18
  - GET\_JAVA\_LOGGING\_LEVEL function, 89-19
  - GET\_PLSQL\_LOGGING\_LEVEL function, 89-20
  - GET\_SCHEMENAMES function, 89-21
  - GET\_TDT\_XML function, 89-22
  - GET\_VALIDATOR function, 89-24
  - REFRESH\_CATEGORY function, 89-28
  - REMOVE\_CATEGORY procedure, 89-29
  - REMOVE\_PROXY procedure, 89-30
  - REMOVE\_SCHEME procedure, 89-31
  - SET\_JAVA\_LOGGING\_LEVEL procedure, 89-32
  - SET\_PLSQL\_LOGGING\_LEVEL procedure, 89-33
  - SET\_PROXY procedure, 89-34
  - VALIDATE\_SCHEME function, 89-35
- DBMS\_MGWADM package
  - summary of subprograms, 90-14
- DBMS\_MGWMSG package
  - constants, 91-4
  - summary of subprograms, 91-21
- DBMS\_MONITOR package
  - statistics tracing and gathering
    - DBMS\_MONITOR package, 92-1
- DBMS\_MVIEW package
  - BEGIN\_TABLE\_REORGANIZATION procedure, 93-7
  - END\_TABLE\_REORGANIZATION procedure, 93-8
  - EXPLAIN\_MVIEW procedure, 93-10
  - EXPLAIN\_REWRITE procedure, 93-11
  - I\_AM\_A\_REFRESH function, 93-13
  - PMARKER function, 93-14
  - PURGE\_DIRECT\_LOAD\_LOG procedure, 93-15
  - PURGE\_LOG procedure, 93-16
  - PURGE\_MVIEW\_FROM\_LOG procedure, 93-17
  - REFRESH procedure, 93-18
  - REFRESH\_ALL\_MVIEWS procedure, 93-20
  - REFRESH\_DEPENDENT procedure, 93-21

---

REGISTER\_MVIEW procedure, 93-23  
 UNREGISTER\_MVIEW procedure, 93-25  
 DBMS\_OBFUSCATION\_TOOLKIT package, 96-1  
 DBMS\_ODCI package, 97-1  
   ESTIMATE\_CPU\_UNITS function, 97-3  
   methods, 97-2  
 DBMS\_OFFLINE\_OG package documentation, 98-2  
 DBMS\_OUTLN package, 99-1  
 DBMS\_OUTPUT package, 100-1  
 DBMS\_PARALLEL\_EXECUTE package, 101-1  
 DBMS\_PCLXUTIL package, 102-1  
 DBMS\_PIPE package, 103-1  
 DBMS\_PREDICTIVE\_ANALYTICS package, 104-1  
   EXPLAIN procedure, 104-6  
   PREDICT procedure, 104-8  
   PROFILE Procedure, 104-10  
 DBMS\_PREPROCESSOR package, 105-1  
 DBMS\_PROFILER package, 106-1  
 DBMS\_PROPAGATION\_ADM package, 107-1  
 DBMS\_RANDOM package, 108-1  
 DBMS\_RECTIFIER\_DIFF package documentation, 109-2  
 DBMS\_REDEFINITION package, 110-1  
 DBMS\_REFRESH package documentation, 111-2  
 DBMS\_REPAIR package, 112-1  
 DBMS\_REPCAT package documentation, 113-2  
 DBMS\_REPCAT\_ADMIN package documentation, 114-2  
 DBMS\_REPCAT\_INSTANTIATE package documentation, 115-2  
 DBMS\_REPCAT\_RGT package documentation, 116-2  
 DBMS\_REPUTIL package documentation, 117-2  
 DBMS\_RESCONFIG package, 118-1  
 DBMS\_RESOURCE\_MANAGER package, 119-1  
 DBMS\_RESOURCE\_MANAGER\_PRIVS package, 120-1  
 DBMS\_RESULT\_CACHE package, 121-1  
 DBMS\_RESUMABLE package, 122-1  
 DBMS\_RLMGR package, 123-1  
   ADD\_ELEMENTARY\_ATTRIBUTE, 123-6  
   ADD\_EVENT, 123-9  
   ADD\_FUNCTIONS, 123-11  
   ADD\_RULE, 123-12  
   CONDITION\_REF, 123-14  
   CONSUME\_EVENT, 123-15  
   CONSUME\_PRIM\_EVENT, 123-17  
   CREATE\_CONDITIONS\_TABLE, 123-19  
   CREATE\_EVENT\_STRUCT, 123-21  
   CREATE\_EXPFIL\_INDEXES, 123-22  
   CREATE\_INTERFACE, 123-24  
   CREATE\_RULE\_CLASS, 123-26  
   DELETE\_RULE, 123-30  
   DROP\_CONDITIONS\_TABLE, 123-31  
   DROP\_EVENT\_STRUCT, 123-32  
   DROP\_EXPFIL\_INDEXES, 123-33  
   DROP\_INTERFACE, 123-34  
   DROP\_RULE\_CLASS, 123-35  
   EXTEND\_EVENT\_STRUCT, 123-36  
   GET\_AGGREGATE\_VALUE, 123-37  
   GRANT\_PRIVILEGE, 123-39  
   PROCESS\_RULES, 123-40  
   PURGE\_EVENTS, 123-42  
   RESET\_SESSION, 123-43  
   REVOKE\_PRIVILEGE, 123-44

---

SYNC\_TEXT\_INDEXES, 123-45

DBMS\_RLS package, 124-1

DBMS\_RLS.ADD\_GROUPED\_POLICY parameters

- enable, 124-7
- function\_schema, 124-7
- long\_predicate, 124-8
- object\_name, 124-7
- object\_schema, 124-7
- policy\_function, 124-7
- policy\_group, 124-7
- policy\_name, 124-7
- policy\_type, 124-8
- sec\_relevant\_cols, 124-8
- statement\_types, 124-7
- static\_policy, 124-7
- update\_check, 124-7

DBMS\_RLS.ADD\_POLICY parameters

- enable, 124-9
- function\_schema, 124-9
- long\_predicate, 124-10
- object\_name, 124-9
- object\_schema, 124-9
- policy\_function, 124-9
- policy\_name, 124-9
- policy\_type, 124-10
- sec\_relevant\_cols, 124-10
- sec\_relevant\_cols\_opt, 124-10
- statement\_types, 124-9
- static\_policy, 124-10
- update\_check, 124-9

DBMS\_RLS.ADD\_POLICY policy types

- CONTEXT\_SENSITIVE, 124-10
- DYNAMIC, 124-10
- SHARED\_CONTEXT\_SENSITIVE, 124-10
- SHARED\_STATIC, 124-10
- STATIC, 124-10

DBMS\_RLS.ADD\_POLICY\_CONTEXT parameters

- attribute, 124-13
- namespace, 124-13
- object\_name, 124-13
- object\_schema, 124-13

DBMS\_RLS.CREATE\_POLICY\_GROUP parameters

- object\_name, 124-14
- object\_schema, 124-14
- policy\_group, 124-14

DBMS\_RLS.DELETE\_POLICY\_GROUP parameters

- object\_name, 124-15
- object\_schema, 124-15
- policy\_group, 124-15

DBMS\_RLS.DISABLE\_GROUPED\_POLICY parameters

- group\_name, 124-16
- object\_name, 124-16
- object\_schema, 124-16
- policy\_name, 124-16

DBMS\_RLS.DROP\_GROUPED\_POLICY parameters

- object\_name, 124-17, 124-18
- object\_schema, 124-17, 124-18
- policy\_group, 124-17
- policy\_name, 124-17, 124-18

---

DBMS\_RLS.DROP\_POLICY\_CONTEXT parameters  
  attribute, 124-19  
  namespace, 124-19  
  object\_name, 124-19  
  object\_schema, 124-19

DBMS\_RLS.ENABLE\_GROUPED\_POLICY parameters  
  enable, 124-20  
  group\_name, 124-20  
  object\_name, 124-20  
  object\_schema, 124-20  
  policy\_name, 124-20

DBMS\_RLS.ENABLE\_POLICY parameters  
  enable, 124-21  
  object\_name, 124-21  
  object\_schema, 124-21  
  policy\_name, 124-21

DBMS\_RLS.REFRESH\_GROUPED\_POLICY parameters  
  group\_name, 124-22  
  object\_name, 124-22  
  object\_schema, 124-22  
  policy\_name, 124-22

DBMS\_RLS.REFRESH\_POLICY parameters  
  object\_name, 124-23  
  object\_schema, 124-23  
  policy\_name, 124-23

DBMS\_ROWID package, 125-1

DBMS\_RULE package, 126-1

DBMS\_RULE\_ADM package, 127-1

DBMS\_SCHEDULER package, 128-1

DBMS\_SERVER\_ALERT package, 129-1

DBMS\_SERVICE package, 130-1

DBMS\_SESSION package, 131-1

DBMS\_SHARED\_POOL package, 132-1

DBMS\_SPACE package, 133-1

DBMS\_SPACE\_ADMIN package, 134-1

DBMS\_SPM package, 135-1

DBMS\_SQL package, 136-1

DBMS\_SQLDIAG package, 137-1

DBMS\_SQLTUNE package, 28-1, 138-1, 139-1

DBMS\_STAT\_FUNCS package, 140-1

DBMS\_STATS package, 141-1

DBMS\_STORAGE\_MAP package, 142-1

DBMS\_STREAMS package, 143-1

DBMS\_STREAMS\_ADM package, 144-1  
  apply user, 144-7  
  deprecated subprograms, 144-4  
  inbound server rules, 144-15  
  inbound servers, 144-7  
  outbound server rules, 144-14

DBMS\_STREAMS\_ADVISOR\_ADM package, 145-1  
  constants, 145-5  
  operational notes, 145-7  
  subprograms, 145-9  
  using, 145-2  
  views, 145-6

DBMS\_STREAMS\_AUTH package, 146-1

DBMS\_STREAMS\_HANDLER\_ADM package, 147-1

DBMS\_STREAMS\_MESSAGING package, 148-1

DBMS\_STREAMS\_TABLESPACE package, 149-1

DBMS\_TDB package, 150-1

---

DBMS\_TDB.CHECK\_DB Function, 150-9  
 DBMS\_TDB.CHECK\_DB procedure, 150-11  
 DBMS\_TRACE package, 151-1  
 DBMS\_TRANSACTION package, 152-1  
 DBMS\_TRANSFORM package, 153-1  
 DBMS\_TTS package, 154-1  
 DBMS\_TYPES package, 155-1  
 DBMS\_UTILITY package, 156-1  
 DBMS\_WARNING package, 157-1  
 DBMS\_WM package documentation, 158-2  
 DBMS\_WORKLOAD\_RECORD package, 159-1  
 DBMS\_WORKLOAD\_REPLAY package, 160-1  
 DBMS\_WORKLOAD\_REPOSITORY package, 161-1  
 DBMS\_XA\_XID Object Type, 162-9  
 DBMS\_XA\_XID\_ARRAY Table Type, 162-10  
 DBMS\_XBD\_VERSION package, 165-1  
 DBMS\_XDB Constants, 163-6  
 DBMS\_XDB Overview, 163-3  
 DBMS\_XDB package, 163-1
 

- ACLCHKPRIVILEGES function, 163-10
- CFG\_GET function, 163-20
- CFG\_REFRESH procedure, 163-21
- CFG\_UPDATE procedure, 163-22
- CHECKPRIVILEGES function, 163-25
- CONFIGUREAUTOSYNC procedure, 167-8
- constants, 163-2
- ConText synchronization settings, 167-6
- CREATEDATASTOREPREF procedure, 167-9
- CREATEFILTERPREF procedure, 167-10
- CREATEFOLDER function, 163-26
- CREATEINDEX procedure, 167-11
- CREATELEXERPREF procedure, 167-12
- CREATEOIDPATH function, 163-27
- CREATEPREFERENCES procedure, 167-13
- CREATERESOURCE function, 163-28
- CREATESECTIONGROUPPREF procedure, 167-14
- CREATESTOPLISTPREF procedure, 167-15
- CREATESTORAGEPREF procedure, 167-16
- CREATEWORLDLISTPREF procedure, 167-17
- DELETERESOURCE procedure, 163-32
- DROPPREFERENCES procedure, 167-18
- EXISTSRESOURCE function, 163-39
- filtering settings, 167-5
- general indexing settings, 167-5
- GETACLDOCUMENT function, 163-40
- GETLOCKTOKEN procedure, 163-50
- GETRESOID function, 163-52
- GETXDB\_TABLESPACE function, 163-53
- LOCKRESOURCE function, 163-60
- methods, 163-7, 167-7
- miscellaneous settings, 167-6
- other index preference settings, 167-6
- RENAMERESOURCE procedure, 163-65
- sectioning and section group settings, 167-5
- SETACL procedure, 163-66
- stoplist settings, 167-5
- SYNC settings, 167-6
- UNLOCKRESOURCE function, 163-77

 DBMS\_XDB\_ADMIN package, 164-1  
 DBMS\_XDB\_VERSION package

---

CHECKIN function, 165-5  
CHECKOUT procedure, 165-6  
GETCONTENTSBOBBYRESID function, 165-7  
GETCONTENTSLOBBYRESID function, 165-8  
GETCONTENTSXMLBYRESID function, 165-9  
GETPREDECESSORS function, 165-10  
GETPREDSBYRESID function, 165-11  
GETRESOURCEBYRESID function, 165-12  
GETSUCCESSORS function, 165-13  
GETSUCCSBYRESID function, 165-14  
MAKEVERSIONED function, 165-15  
UNCHECKOUT function, 165-16  
DBMS\_XDBRESOURCE package, 166-1  
DBMS\_XDBZ package  
  DISABLE\_HIERARCHY procedure, 168-6  
  ENABLE\_HIERARCHY procedure, 168-7  
  GET\_ACLOID function, 168-8  
  GET\_USERID function, 168-9  
  IS\_HIERARCHY\_ENABLED function, 168-10  
  PURGELDAPCACHE function, 168-11  
DBMS\_XMLDOM Constants, 170-7  
DBMS\_XMLDOM package, 170-1  
  APPENDDATA procedure, 170-43  
  CREATEATTRIBUTE function, 170-45  
  CREATEDATASECTION function, 170-46  
  CREATECOMMENT function, 170-47  
  CREATEDOCUMENT function, 170-48  
  CREATEDOCUMENTFRAGMENT, 170-49  
  CREATEELEMENT function, 170-50  
  CREATEENTITYREFERENCE function, 170-51  
  CREATEPROCESSINGINSTRUCTION function, 170-52  
  CREATETEXTNODE function, 170-53  
  DELETEDATA procedure, 170-54  
  description, 170-4  
  exceptions, 170-9  
  FINDENTITY function, 170-55  
  FINDNOTATION function, 170-56  
  FREEDOCFRAG procedure, 170-57  
  FREEDOCUMENT procedure, 170-58  
  GETATTRIBUTE function, 170-62  
  GETATTRIBUTENODE function, 170-63  
  GETBUBLICID function, 170-97  
  GETCHARSET function, 170-69  
  GETCHILDRENBYTAGNAME function, 170-67  
  GETDATA function, 170-68  
  GETDOCTYPE function, 170-69  
  GETDOCUMENTELEMENT function, 170-70  
  GETELEMENTSBYTAGNAME function, 170-71, 170-72  
  GETENTITIES function, 170-72  
  GETEXPANDEDNAME function, 170-74  
  GETIMPLEMENTATION function, 170-75  
  GETLENGTH function, 170-77, 170-78  
  GETNAME function, 170-79, 170-80  
  GETNAMEDITEM function, 170-80  
  GETNAMESPACE function, 170-82  
  GETNAMESPACE procedure, 170-81  
  GETNEXTSIBLING function, 170-82  
  GETNODENAME function, 170-84  
  GETNODETYPE function, 170-83  
  GETNODEVALUE function, 170-85



---

GETNOTATIONNAME function, 170-88  
GETNOTATIONS function, 170-89  
GETOWNERDOCUMENT function, 170-91  
GETOWNERELEMENT function, 170-92  
GETPARENTNODE function, 170-93  
GETPREFIX function, 170-94  
GETPREVIOUSIBLING function, 170-95  
GETPUBLICID function, 170-96, 170-97  
GETQUALIFIEDNAME function, 170-97, 170-98  
GETSCHEMANODE function, 170-98  
GETSPECIFIED function, 170-99  
GETSTANDALONE function, 170-100  
GETSYSTEMID function, 170-101, 170-102  
GETTAGNAME function, 170-102  
GETTARGET function, 170-90  
GETVALUE function, 170-103  
GETVERSION function, 170-104  
GETXMLTYPE function, 170-105  
HASATTRIBUTE function, 170-106  
HASATTRIBUTES function, 170-107  
HASCHILDNODES function, 170-108  
HASFEATURE function, 170-109  
IMPORTNODE function, 170-110  
INSERTBEFORE function, 170-111  
INSERTDATA procedure, 170-112  
ISNULL function, 170-113  
MAKEATTR function, 170-117  
MAKECDATASECTION function, 170-118  
MAKECHARACTERDATA function, 170-119  
MAKECOMMENT function, 170-120  
MAKEDOCUMENT function, 170-121  
MAKEDOCUMENTFRAGMENT function, 170-122  
MAKEDOCUMENTTYPE function, 170-123  
MAKEELEMENT function, 170-124  
MAKEENTITY function, 170-125  
MAKEENTITYREFERENCE function, 170-126  
MAKENODE function, 170-127, 170-130  
MAKENOTATION function, 170-130  
MAKEPROCESSINGINSTRUCTION function, 170-131  
MAKETEXT function, 170-132  
methods  
    APPENDCHILD function, 170-42  
    APPENDDATA procedure, 170-43  
    CLONENODE function, 170-44  
    CREATEATTRIBUTE function, 170-45  
    CREATECDATASECTION function, 170-46  
    CREATECOMMENT function, 170-47  
    CREATEDOCUMENT function, 170-48  
    CREATEDOCUMENTFRAGMENT function, 170-49  
    CREATEELEMENT function, 170-50  
    CREATEENTITYREFERENCE function, 170-51  
    CREATEPROCESSINGINSTRUCTION function, 170-52  
    CREATETEXTNODE function, 170-53  
    DELETEDATA procedure, 170-54  
    DOMAttr interface, 170-13  
    DOMCDATASection interface, 170-14  
    DOMCharacterData interface, 170-15  
    DOMComment interface, 170-16  
    DOMDocument interface, 170-17  
    DOMDocumentFragment interface, 170-19, 170-41

---

DOMDocumentType interface, 170-20, 170-41  
 DOMELEMENT interface, 170-21, 170-41  
 DOMEntity interface, 170-22, 170-41  
 DOMEntityReference interface, 170-23  
 DOMImplementation interface, 170-24  
 DOMNamedNodeMap interface, 170-25  
 DOMNode  
     APPENDCHILD function, 170-42  
     CLONENODE function, 170-44  
     GETATTRIBUTES function, 170-64  
     GETCHILDNODES function, 170-66  
     GETEXPANDEDNAME procedure, 170-73  
     GETFIRSTCHILD function, 170-74  
     GETLASTCHILD function, 170-76  
     GETLOCALNAME procedure, 170-78  
 DOMNodeList interface, 170-26  
 DOMNotation interface, 170-27  
 DOMProcessingInstruction interface, 170-28  
 DOMText interface, 170-11, 170-29  
 FINDENTITY function, 170-55  
 FINDNOTATION function, 170-56  
 FREEDOCFRAG procedure, 170-57  
 FREEDOCUMENT procedure, 170-58  
 FREEELEMENT procedure, 170-59  
 FREENODE procedure, 170-59, 170-60  
 FREENODELIST Procedure, 170-61  
 GETATTRIBUTE function, 170-62  
 GETATTRIBUTENODE function, 170-63  
 GETATTRIBUTES function, 170-64  
 GETBUBLICID function, 170-97  
 GETCHARSET function, 170-69  
 GETCHILDNODES function, 170-66  
 GETCHILDRENBYTAGNAME function, 170-67  
 GETDATA function, 170-68  
 GETDOCTYPE function, 170-69  
 GETDOCUMENTELEMENT function, 170-70  
 GETELEMENTSBYTAGNAME function, 170-71, 170-72  
 GETENTITIES function, 170-72  
 GETEXPANDEDNAME function, 170-74  
 GETEXPANDEDNAME procedure, 170-73  
 GETFIRSTCHILD function, 170-74  
 GETIMPLEMENTATION function, 170-75  
 GETLASTCHILD function, 170-76  
 GETLENGTH function, 170-77, 170-78  
 GETLOCALNAME procedure, 170-78  
 GETNAME function, 170-79, 170-80  
 GETNAMEDITEM function, 170-80  
 GETNAMESPACE function, 170-82  
 GETNAMESPACE procedure, 170-81  
 GETNEXTSIBLING function, 170-82  
 GETNODENAME function, 170-84  
 GETNODETYPE function, 170-83  
 GETNODEVALUE function, 170-85  
 GETNOTATIONNAME function, 170-88  
 GETNOTATIONS function, 170-89  
 GETOWNERDOCUMENT function, 170-91  
 GETOWNERELEMENT function, 170-92  
 GETPARENTNODE function, 170-93

---

GETPREFIX function, 170-94  
GETPREVIOUSIBLING function, 170-95  
GETPUBLICID function, 170-96, 170-97  
GETQUALIFIEDNAME function, 170-97, 170-98  
GETSCHEMANODE function, 170-98  
GETSPECIFIED function, 170-99  
GETSTANDALONE function, 170-100  
GETSYSTEMID function, 170-101, 170-102  
GETTAGNAME function, 170-102  
GETTARGET function, 170-90  
GETVALUE function, 170-103  
GETVERSION function, 170-104  
GETXMLTYPE function, 170-105  
HASATTRIBUTE function, 170-106  
HASATTRIBUTES function, 170-107  
HASCHILDNODES function, 170-108  
HASFEATURE function, 170-109  
IMPORTNODE function, 170-110  
INSERTBEFORE function, 170-111  
INSERTDATA procedure, 170-112  
ISNULL function, 170-113  
MAKEATTR function, 170-117  
MAKECDATASECTION function, 170-118  
MAKECHARACTERDATA function, 170-119  
MAKECOMMENT function, 170-120  
MAKEDOCUMENT function, 170-121  
MAKEDOCUMENTFRAGMENT function, 170-122  
MAKEDOCUMENTTYPE function, 170-123  
MAKEELEMENT function, 170-124  
MAKEENTITY function, 170-125  
MAKEENTITYREFERENCE function, 170-126  
MAKENODE function, 170-127, 170-130  
MAKENOTATION function, 170-130  
MAKEPROCESSINGINSTRUCTION function, 170-131  
MAKETEXT function, 170-132  
NEWDOMDOCUMENT function, 170-133  
NORMALIZE procedure, 170-134  
REMOVEATTRIBUTE procedure, 170-135  
REMOVEATTRIBUTENODE function, 170-136  
REMOVENAMEDITEM function, 170-138  
REPLACECHILD function, 170-139  
REPLACEDATA procedure, 170-140  
RESOLVENAMESPACEPREFIX function, 170-141  
SETATTRIBUTE procedure, 170-142  
SETATTRIBUTENODE function, 170-143  
SETCHARSET procedure, 170-152  
SETDATA procedure, 170-145  
SETNAMEDITEM function, 170-147  
SETNODEVALUE procedure, 170-148  
SETPREFIX procedure, 170-151  
SETSTANDALONE procedure, 170-152  
SETVALUE procedure, 170-153  
SETVERSION procedure, 170-154  
SPLITTEXT function, 170-155  
SUBSTRINGDATA function, 170-156  
WRITETOBUFFER procedure, 170-158  
WRITETOCLOB procedure, 170-159  
WRITETOFILE procedure, 170-160  
NEWDOMDOCUMENT function, 170-133  
NORMALIZE procedure, 170-134

---

REMOVEATTRIBUTE procedure, 170-135  
 REMOVEATTRIBUTENODE function, 170-136  
 REMOVENAMEDITEM function, 170-138  
 REPLACECHILD function, 170-139  
 REPLACEDATA procedure, 170-140  
 RESOLVENAMESPACEPREFIX function, 170-141  
 SETATTRIBUTE procedure, 170-142  
 SETATTRIBUTENODE function, 170-143  
 SETCHARSET procedure, 170-152  
 SETDATA procedure, 170-145  
 SETNAMEDITEM function, 170-147  
 SETNODEVALUE procedure, 170-148  
 SETPREFIX procedure, 170-151  
 SETSTANDALONE procedure, 170-152  
 SETVALUE procedure, 170-153  
 SETVERSION procedure, 170-154  
 SPLITTEXT function, 170-155  
 SUBSTRINGDATA function, 170-156  
 types, 69-5, 170-8  
 WRITETOBUFFER procedure, 170-158  
 WRITETOCLOB procedure, 170-159  
 WRITETOFILE procedure, 170-160

DBMS\_XMLGEN package, 171-1  
 CLOSECONTEXT procedure, 171-5  
 CONVERT function, 171-6  
 GETNUMROWSPROCESSED Function, 171-7  
 GETXML function, 171-8  
 GETXMLTYPE function, 171-9  
 NEWCONTEXT function, 171-10  
 RESTARTQUERY procedure, 171-12  
 SETCONVERTSPECIALCHARS procedure, 171-13  
 SETMAXROWS procedure, 171-14  
 SETROWSETTAG procedure, 171-16  
 SETROWTAG procedure, 171-17  
 SETSKIPROWS procedure, 171-18  
 USEITEMTAGSFORCOLL procedure, 171-19  
 USENULLATTRIBUTEINDICATOR procedure, 171-20

DBMS\_XMLINDEX package, 172-1

DBMS\_XMLPARSER package, 173-1  
 FREEPARSER procedure, 173-5  
 GETDOCTYPE function, 173-6  
 GETDOCUMENT function, 173-7  
 GETRELEASEVERSION function, 173-8  
 GETVALIDATIONMODE function, 173-9  
 NEWPARSER function, 173-10  
 PARSE function, 173-11  
 PARSE procedure, 173-11  
 PARSEBUFFER procedure, 173-12  
 PARSECLOB procedure, 173-13  
 PARSEDTD procedure, 173-14  
 PARSEDTDBUFFER procedure, 173-15  
 PARSEDTDCLOB procedure, 173-16  
 SETBASEDIR procedure, 173-17  
 SETDOCTYPE procedure, 173-18  
 SETERRORLOG procedure, 173-19  
 SETPRESERVEWHITESPACE procedure, 173-20  
 SETVALIDATIONMODE procedure, 173-21  
 SHOWWARNINGS procedure, 173-22

DBMS\_XMLQUERY package, 174-1  
 CLOSECONTEXT procedure, 174-8

---

constants, 174-4  
 GETDTD function, 174-9  
 GETDTD procedure, 174-9  
 GETEXCEPTIONCONTENT procedure, 174-10  
 GETNUMROWSPROCESSED procedure, 174-11  
 GETVERSION procedure, 174-12  
 GETXML function, 174-13  
 GETXML procedure, 174-13  
 NEWCONTEXT function, 174-14  
 PROPAGATEORIGINALEXCEPTION procedure, 174-15  
 REMOVEXSLTPARAM procedure, 174-16  
 SETBINDVALUE procedure, 174-17  
 SETCOLLIDATTRNAME procedure, 174-18  
 SETDATAHEADER procedure, 174-19  
 SETDATEFORMAT procedure, 174-20  
 SETENCODINGTAG procedure, 174-21  
 SETERRORTAG procedure, 174-22  
 SETMAXROWS procedure, 174-23  
 SETMETAHEADER procedure, 174-24  
 SETRAISEEXCEPTION procedure, 174-25  
 SETRAISENOROWSEXCEPTION procedure, 174-26  
 SETROWIDATTRNAME procedure, 174-27  
 SETROWIDATTRVALUE procedure, 174-28  
 SETROWSETTAG procedure, 174-29  
 SETROWTAG procedure, 174-30  
 SETSKIPROWS procedure, 174-31  
 SETSQLTOXMLNAMEESCAPING procedure, 174-32  
 SETSTYLESHEETHEADER procedure, 174-33  
 SETTAGCASE procedure, 174-34  
 SETXSLT procedure, 174-35  
 SETXSLTPARAM procedure, 174-36  
 types, 174-4  
 USENULLATTRIBUTEINDICATOR procedure, 174-37  
 USETYPEFORCOLLELEMTAG procedure, 174-38  
 DBMS\_XMLSAVE package, 175-1  
   CLEARKEYCOLUMNLIST procedure, 175-7  
   CLEARUPDATECOLUMNLIST procedure, 175-8  
   CLOSECONTEXT procedure, 175-9  
   constants, 175-4  
   DELETXML function, 175-10  
   GETEXCEPTIONCONTENT procedure, 175-11  
   INSERTXML function, 175-12  
   NEWCONTEXT function, 175-13  
   PROPAGATEORIGINALEXCEPTION procedure, 175-14  
   REMOVEXSLTPARAM procedure, 175-15  
   SETBATCHSIZE procedure, 175-16  
   SETCOMMITBATCH procedure, 175-17  
   SETDATEFORMAT procedure, 175-18  
   SETIGNORECASE procedure, 175-19  
   SETKEYCOLUMN procedure, 175-20  
   SETPRESERVEWHITESPACE procedure, 175-21  
   SETROWTAG procedure, 175-22  
   SETSQLTOXMLNAMEESCAPING procedure, 175-23  
   SETUPDATECOLUMN procedure, 175-24  
   SETXSLT procedure, 175-25  
   SETXSLTPARAM procedure, 175-26  
   UPDATEXML function, 175-27  
 DBMS\_XMLSCHEMA Constants, 176-5  
 DBMS\_XMLSCHEMA Operational Notes, 176-8  
 DBMS\_XMLSCHEMA package, 176-1

---

DBMS\_XMLSCHEMA Views, 176-7  
 DBMS\_XMLSTORE package, 177-1  
     CLEARKEYCOLUMNLIST procedure, 177-6  
     CLEARUPDATECOLUMNLIST procedure, 177-7  
     CLOSECONTEXT procedure, 177-8  
     DELETXML function, 177-9  
     INSERTXML function, 177-10  
     NEWCONTEXT function, 177-11  
     SETKEYCOLUMN procedure, 177-12  
     SETROWTAG procedure, 177-13  
     SETUPDATECOLUMN procedure, 177-14  
     types, 177-4  
     UPDATEXML function, 177-15  
 DBMS\_XMLTRANSLATIONS package, 178-1  
 DBMS\_XPLAN package, 179-1  
 DBMS\_XSLPROCESSOR Package, 180-1, 180-3  
 DBMS\_XSLPROCESSOR package  
     CLOB2FILE procedure, 180-6  
     FREEPROCESSOR procedure, 180-7  
     FREESTYLESHEET procedure, 180-8  
     NEWPROCESSOR function, 180-9  
     NEWSTYLESHEET function, 180-10  
     PROCESSXSL function, 180-11  
     READ2CLOB function, 180-13  
     REMOVEPARAM procedure, 180-14  
     RESETPARAMS procedure, 180-15  
     SELECTNODES function, 180-16  
     SELECTSINGLENODE function, 180-17  
     SETERRORLOG procedure, 180-18  
     SETPARAM procedure, 180-19  
     SHOWWARNINGS procedure, 180-20  
     TRANSFORMNODE function, 180-21  
     VALUEOF procedure, 180-22  
 DBMSOUTPUT\_LINESARRAY Object Type, 100-13  
 DBUriType, 245-18  
 DBURITYPE function, 245-20  
 DBUriType subtype, 245-18  
     CREATEURI function, 245-19  
     DBURITYPE function, 245-20  
     GETBLOB function, 245-21  
     GETCLOB function, 245-22  
     GETCONTENTTYPE function, 245-23  
     GETEXTERNALURL function, 245-24  
     GETURL function, 245-25  
     GETXML function, 245-26  
     methods, 245-18  
 DEAUTHORIZE\_DAD Procedure, 63-13  
 DEBUG\_EXPTOC package, 181-1  
 DEBUG\_ON procedure, 52-28  
 DECLARE\_REWRITE\_EQUIVALENCE Procedures, 17-7  
 DECODEFEATURES Function, 47-46  
 DEFAULT\_INDEX\_PARAMETERS procedure, 65-17  
 DEFAULT\_NAME function, 43-15  
 DEFAULT\_XPINDEX\_PARAMETERS procedure, 65-19  
 DEFINE\_ARRAY procedure, 136-62  
 DEFINE\_COLUMN procedure, 136-65  
 DEFINE\_COLUMN\_CHAR Procedure, 136-67  
 DEFINE\_COLUMN\_LONG procedure, 136-68  
 DEFINE\_COLUMN\_RAW Procedure, 136-69  
 DEFINE\_COLUMN\_ROWID Procedure, 136-70

---

DEFRAG\_INDEX procedure, 65-21  
DEINIT\_CLEANUP procedure, 27-19  
DELETE Procedure, 16-8  
DELETE\_ALL\_ERRORS procedure, 21-25  
DELETE\_BREAKPOINT function, 52-29  
DELETE\_CAPTURE\_INFO Procedure, 159-7  
DELETE\_CATEGORY Procedure, 119-25  
DELETE\_COLUMN member procedure, 248-20  
DELETE\_COLUMN procedure, 144-80  
DELETE\_COLUMN\_STATS procedure, 141-29  
DELETE\_CONSUMER\_GROUP procedure, 119-26  
DELETE\_DAD\_ATTRIBUTE Procedure, 63-14, 63-15  
DELETE\_DATABASE\_PREFS Procedure, 141-31  
DELETE\_DATABASE\_STATS procedure, 141-34  
DELETE\_DICTIONARY\_STATS procedure, 141-35  
DELETE\_ERROR procedure, 21-26  
DELETE\_FILTER Procedure, 159-8, 160-12  
DELETE\_FINDING\_DIRECTIVE Procedure, 16-9  
DELETE\_FIXED\_OBJECTS\_STATS procedure, 141-36  
DELETE\_INDEX\_STATS procedure, 141-37  
DELETE\_OBJECT procedure, 234-8  
DELETE\_OER\_BREAKPOINT function, 52-30  
DELETE\_PARAMETER\_DIRECTIVE Procedure, 16-10  
DELETE\_PENDING\_STATS Procedure, 141-38  
DELETE\_PLAN procedure, 119-27  
DELETE\_PLAN\_CASCADE procedure, 119-28  
DELETE\_PLAN\_DIRECTIVE procedure, 119-29  
DELETE\_PRIVILEGE Procedure, 94-14  
DELETE\_REPLAY\_INFO Procedure, 160-13  
DELETE\_RULE procedure, 123-30  
DELETE\_SCHEMA\_PREFS Procedure, 141-39  
DELETE\_SCHEMA\_STATS Procedure, 141-42  
DELETE\_SEGMENT\_DIRECTIVE Procedure, 16-11  
DELETE\_SERVICE procedure, 130-12  
DELETE\_SQL\_DIRECTIVE Procedure, 16-12  
DELETE\_SQLSET procedure, 139-37  
DELETE\_SQLWKLD Procedure, 18-20  
DELETE\_SQLWKLD\_REF Procedure, 18-21  
DELETE\_SQLWKLD\_STATEMENT Procedure, 18-22  
DELETE\_SYSTEM\_STATS procedure, 141-43  
DELETE\_TABLE\_PREFS Procedure, 141-44  
DELETE\_TABLE\_STATS procedure, 141-47  
DELETE\_TASK Procedure, 18-24  
DELETECONTENT Procedure, 47-47, 48-16  
DELETEDATA procedure, 170-54  
DELETEDIRECTORY Procedure, 47-48, 48-17  
DELETEFILE Procedure, 47-49, 48-18  
DELETEHTTPEXPIREMAPPING Procedure, 163-30  
DELETEDMIMEMAPPING Procedure, 163-31  
DELETEREPOSITORYRESCONFIG Procedure, 118-8  
DELETERESCONFIG Procedure, 118-9  
DELETERESOURCE procedure, 163-32  
DELETERESOURCEMETADATA Procedures, 163-33  
DELETESchema procedure, 176-13  
DELETESCHEMALOCMAPPING Procedure, 163-34  
DELETESERVLET Procedure, 163-35  
DELETESERVLETMAPPING Procedure, 163-36  
DELETESERVLETSECROLE Procedure, 163-37  
DELETXML function, 175-10, 177-9  
DELETXMLEXTENSION Procedure, 163-38

---

DEQUEUE procedure, 148-6  
 DEQUEUE\_ARRAY Function, 22-14  
 DEQUEUE\_OPTIONS\_T Type, 243-17  
 DEREGISTER Procedure, 38-28  
 DEREGSTORECOMMAND Function, 49-16  
 DERIVE\_FROM\_MVIEW function, 41-41  
 DESC\_REC2 Record Type, 136-27  
 DESC\_REC3 Record Type, 136-28  
 DESC\_RESC Record Type, 136-26  
 DESC\_TAB Table Type, 136-35  
 DESC\_TAB2 Table Type, 136-36  
 DESC\_TAB3 Table Type, 136-37  
 DESCRIBE\_COLUMNS procedure, 136-71  
 DESCRIBE\_COLUMNS2 procedure, 136-72  
 DESCRIBE\_COLUMNS3 Procedure, 136-73  
 DESCRIBE\_DIMENSION procedure, 59-5  
 DESCRIBE\_PROCEDURE procedure, 57-11  
 DESCRIBE\_STACK procedure, 45-32  
 DESDecrypt procedure, 96-8, 96-13  
 DESEncrypt procedure, 96-15  
 DESTROY\_REQUEST\_CONTEXT Procedure, 224-47  
 DETACH\_SESSION procedure, 52-31  
 DETACH\_SIMPLE\_TABLESPACE procedure, 149-24  
 DETACH\_TABLESPACES procedure, 149-26  
 DFN function  
     of HTF package, 182-38  
 DFN procedure  
     of HTP package, 183-37  
 DIFF\_TABLE\_STATS\_IN\_HISTORY Function, 141-49  
 DIFF\_TABLE\_STATS\_IN\_PENDING Function, 141-50  
 DIFF\_TABLE\_STATS\_IN\_STATTAB Function, 141-51  
 dimension maintenance (cube), 41-1  
 dimensions  
     optimizer statistics, 31-1  
 DIRECTORY\_OBJECT\_SET type, 149-6  
 DIRLISTCLOSE function  
     of HTF package, 182-39  
 DIRLISTCLOSE procedure  
     of HTP package, 183-38  
 DIRLISTOPEN function  
     of HTF package, 182-40  
 DIRLISTOPEN procedure  
     of HTP package, 183-39  
 DISABLE procedure, 43-16  
     of DBMS\_FLASHBACK package, 69-12  
     of DBMS\_OUTPUT package, 100-15  
     of OWA\_CACHE package, 186-5  
 DISABLE Procedures, 29-5  
 DISABLE\_BREAKPOINT function, 52-32  
 DISABLE\_HIERARCHY procedure, 168-6  
 DISABLE\_JOB Procedure, 90-41  
 DISABLE\_PROPAGATION\_SCHEDULE Procedure, 90-42  
 DISABLETRANSLATION Procedure, 178-5  
 DISCONNECT\_SESSION procedure, 130-13  
 DISPLAY function, 179-10  
 DISPLAY\_AWR function, 179-13  
 DISPLAY\_CURSOR function, 179-16  
 DISPLAY\_PLAN Function, 179-19  
 DISPLAY\_SQL\_PLAN\_BASELINE Function, 179-22  
 DISPLAY\_SQLSET Function, 179-24



---

DIST\_TXN\_SYNC Procedure, 162-12  
DIV function  
    of HTF package, 182-41  
DIV procedure  
    of HTP package, 183-40  
DLISTCLOSE function  
    of HTF package, 182-42  
DLISTCLOSE procedure  
    of HTP package, 183-41  
DLISTDEF function  
    of HTF package, 182-43  
DLISTDEF procedure  
    of HTP package, 183-42  
DLISTOPEN function  
    of HTF package, 182-44  
DLISTOPEN procedure  
    of HTP package, 183-43  
DLISTTERM function  
    of HTF package, 182-45  
DLISTTERM procedure  
    of HTP package, 183-44  
DOMAIN\_LEVEL Function, 95-6  
DOMAINS Function, 95-7  
DOMAttr methods, 170-13  
DOMCDataSection methods, 170-14  
DOMCharacterData methods, 170-15  
DOMComment methods, 170-16  
DOMDocument methods, 170-17  
DOMDocumentType methods, 170-20, 170-41  
DOMEntity methods, 170-41  
DOMNamedNodeMap methods, 170-25  
DOMText methods, 170-11  
DOWNGRADE procedure, 154-7  
DOWNLOAD\_FILE procedures, 239-5  
DROFILESYSTEM Procedures, 50-10  
DROP\_ACL Procedure, 94-15  
DROP\_ALL function, 142-6  
DROP\_ANALYSIS\_TASK Procedure, 138-10  
DROP\_APPLY procedure, 21-27  
DROP\_ATTRIBUTE\_SET procedure, 65-22  
DROP\_BASELINE procedure, 161-35  
DROP\_BY\_CAT procedure, 99-8  
DROP\_CAPTURE procedure  
    capture process  
    dropping, 32-31  
DROP\_CHUNKS Procedure, 101-21  
DROP\_COMPARISON Procedure, 35-24  
DROP\_CONDITIONS\_TABLE procedure, 123-31  
DROP\_DAD Procedure, 63-16  
DROP\_DIAGNOSIS\_TASK Procedure, 137-18  
DROP\_EMPTY\_SEGMENTS Procedure, 134-11  
DROP\_EVALUATION\_CONTEXT procedure, 127-18  
DROP\_EVENT\_STRUCT procedure, 123-32  
DROP\_EXPFIL\_INDEXES procedure, 123-33  
DROP\_EXTENDED\_STATS Procedure, 141-52  
DROP\_FILE function, 142-8  
DROP\_FILE\_GROUP procedure, 67-18  
DROP\_INTERFACE procedure, 123-34  
DROP\_MVIEW procedure, 41-10, 41-43  
DROP\_OBJECT\_DEPENDENCY procedure, 21-29

---

DROP\_PROPAGATION procedure, 107-11  
DROP\_PURGE\_JOB procedure, 27-20  
DROP\_REWRITE\_EQUIVALENCE Procedure, 17-9  
DROP\_RULE procedure, 127-19  
DROP\_RULE\_CLASS procedure, 123-35  
DROP\_RULE\_SET procedure, 127-20  
DROP\_SNAPSHOT\_RANGE procedure, 161-37  
DROP\_SQL\_PATCH Function & Procedure, 137-19  
DROP\_SQL\_PLAN\_BASELINE, 135-14  
DROP\_SQL\_PROFILE procedure, 139-38  
DROP\_SQLSET procedure, 139-39  
DROP\_STAT\_TABLE procedure, 141-53  
DROP\_STMT\_HANDLER procedure, 147-13  
DROP\_TASK Procedure, 101-20  
DROP\_TRANSFORMATION procedure, 153-5  
DROP\_TUNING\_TASK procedure, 139-40  
DROP\_UNUSED procedure, 99-9  
DROP\_VERSION procedure, 67-19  
DROP\_VIEW procedure  
    of DBMS\_HS\_PARALLEL package, 74-7  
DROPPARAMETER Procedure, 172-8  
DROPPREFERENCES procedure, 167-18  
DROPREPOSITORYXMLINDEX Procedure, 164-7  
DROPSTORE Procedure, 49-17  
DUMP\_ORPHAN\_KEYS procedure, 112-13  
dynamic SQL  
    anonymous blocks and, 136-3  
    DBMS\_SQL functions, using, 136-2  
    execution flow in, 136-7

## E

EDIT\_DISTANCE Function, 230-6  
EDIT\_DISTANCE\_SIMILARITY Function, 230-7  
EHLO function and procedure, 235-21  
EM function  
    of HTF package, 182-46  
EM procedure  
    of HTP package, 183-45  
e-mail from PL/SQL (email), 237-9  
EMPHASIS function  
    of HTF package, 182-47  
EMPHASIS procedure  
    of HTP package, 183-46  
ENABLE procedure, 43-17, 100-16  
ENABLE Procedures, 29-6  
ENABLE\_AT\_SYSTEM\_CHANGE\_NUMBER procedure, 69-13  
ENABLE\_AT\_TIME procedure, 69-14  
ENABLE\_BREAKPOINT function, 52-33  
ENABLE\_HIERARCHY procedure, 168-7  
ENABLE\_JOB Procedure, 90-43  
ENABLE\_PROPAGATION\_SCHEDULE Procedure, 90-44  
ENABLETRANSLATION Procedure, 178-6  
END\_PREPARE Procedure, 60-12  
END\_REQUEST procedure, 224-48  
END\_RESPONSE procedure, 224-49  
END\_SQL\_BLOCK Procedure, 119-30  
END\_UPGRADE Procedure, 60-13  
ENDCREATE member procedure  
    of ANYDATA TYPE, 240-8  
    of ANYDATASET TYPE, 241-6

---

- of ANYTYPE TYPE, 242-7
- ENQUEUE procedure, 148-8
- ENQUEUE\_ARRAY Function, 22-18
- ENQUEUE\_OPTIONS\_T Type, 243-20
- ENQUOTE\_LITERAL Function, 26-5
- ENQUOTE\_NAME Function, 26-6
- EPC\_TO\_ORACLE\_SCHEME function, 89-12
- EQUALS\_HOST Function, 95-8
- ERASE Procedures, 82-35
- error queue
  - deleting errors, 21-25, 21-26
  - executing errors, 21-30, 21-31
  - getting error messages, 21-34
- ESCAPE function, 238-7
- ESCAPE\_SC function
  - of HTF package, 182-48
- ESCAPE\_SC procedure
  - of HTP package, 183-47
- ESCAPE\_URL function
  - of HTF package, 182-49
- ESCAPEURI function, 245-38
- ESTIMATE\_CPU\_UNITS function, 97-3
- ESTIMATE\_MVIEW\_SIZE Procedure, 93-9
- ETINSTANCE member function
  - of ANYDATASET TYPE, 241-11
- EVALUATE procedure, 126-7
- EVOLVE\_SQL\_PLAN\_BASELINE Function, 135-15
- EXACT\_TEXT\_SIGNATURES procedure, 99-10
- EXEC\_DDL\_STATEMENT procedure, 156-29
- EXECUTE function, 136-75
- EXECUTE member procedure, 248-9, 248-20
- EXECUTE procedure, 52-34
- EXECUTE\_ALL\_ERRORS procedure, 21-30
- EXECUTE\_ANALYSIS\_TASK, 138-11
- EXECUTE\_AND\_FETCH function, 136-76
- EXECUTE\_DIAGNOSIS\_TASK Procedure, 137-20
- EXECUTE\_ERROR procedure, 21-31
- EXECUTE\_IMMEDIATE Procedure, 75-13
- EXECUTE\_NON\_QUERY Function, 75-14
- EXECUTE\_TASK Procedure, 18-25
- EXECUTE\_TUNING\_TASK Function & Procedure, 28-6, 139-41
- execution flow
  - in dynamic SQL, 136-7
- EXF\$ATTRIBUTE object type, 247-5
- EXF\$ATTRIBUTE\_LIST object type, 247-6
- EXF\$INDEXOPER object type, 247-7
- EXF\$TABLE\_ALIAS object type, 247-9
- EXF\$TEXT object type, 247-10
- EXF\$XPATH\_TAG object type, 247-11
- EXF\$XPATH\_TAGS object type, 247-13
- EXISTSNODE function, 259-8
- EXISTSRESOURCE function, 163-39
- EXPAND\_MESSAGE function, 129-13
- EXPLAIN\_SQL\_TESTCASE Function, 137-21
- EXPONENTIAL\_DIST\_FIT procedure, 140-3
- EXPORT\_AWR Procedure, 159-9, 160-14
- EXPORT\_COLUMN\_STATS procedure, 141-54
- EXPORT\_DATABASE\_PREFS Procedure, 141-55
- EXPORT\_DATABASE\_STATS procedure, 141-56
- EXPORT\_DICTIONARY\_STATS procedure, 141-57

---

EXPORT\_FIXED\_OBJECTS\_STATS procedure, 141-58  
EXPORT\_INDEX\_STATS procedure, 141-59  
EXPORT\_PENDING\_STATS Procedure, 141-60  
EXPORT\_SCHEMA\_PREFS Procedure, 141-61  
EXPORT\_SCHEMA\_STATS procedure, 141-62  
EXPORT\_SQL\_TESTCASE Procedures, 137-22  
EXPORT\_SQL\_TESTCASE\_DIR\_BY\_INC Function, 137-25  
EXPORT\_SQL\_TESTCASE\_DIR\_BY\_TXT Function, 137-26  
EXPORT\_SYSTEM\_STATS procedure, 141-63  
EXPORT\_TABLE\_PREFS Procedure, 141-64  
EXPORT\_TABLE\_STATS procedure, 141-65  
EXPORT\_XML procedure, 41-45, 41-47  
EXPORT\_XML\_TO\_FILE procedure, 41-16  
Expression Filter object types, 247-1  
extend window  
    to create a new view, 34-3  
EXTEND\_EVENT\_STRUCT procedure, 123-36  
EXTRACT function, 259-9  
EXTRACTXLIFF Function & Procedure, 178-7

## F

FCLOSE procedure, 223-13  
FCLOSE\_ALL procedure, 223-14  
FCOPY procedure, 223-15  
FEATURE\_T Record Type, 47-24  
FEATURENAME Function, 47-50  
FEATURES\_T Table Type, 47-31  
FETCH\_ROW Function, 75-15  
FETCH\_ROWS function, 136-77  
FFLUSH procedure, 223-16  
FGETATTR procedure, 223-17  
FGETPOS function, 223-18  
FI\_HORIZONTAL function, 71-3  
FI\_TRANSACTIONAL function, 71-5  
file groups, 67-1  
    adding files, 67-7  
    altering, 67-11  
    altering files, 67-9  
    altering versions, 67-13  
    creating, 67-15  
    creating versions, 67-17  
    dropping, 67-18  
    dropping versions, 67-19  
    granting object privileges, 67-20  
    granting system privileges, 67-21  
    purging, 67-22  
    removing files, 67-23  
    revoking object privileges, 67-24  
    revoking system privileges, 67-25  
FILE type, 149-7  
FILE\_SET type, 149-8  
FILECLOSE Procedure, 82-37  
FILECLOSEALL Procedure, 82-38  
FILEEXISTS Function, 82-39  
FILEGETNAME Procedure, 82-40  
FILEISOPEN Function, 82-41  
FILEOPEN Procedure, 82-42  
FILETYPE Record Type, 223-10  
FIND\_AFFECTED\_TABLE Procedure, 60-14  
FINDENTITY function, 170-55

---

FINDNOTATION function, 170-56  
fine-grained access control  
    DBMS\_RLS package, 124-1  
FINISH\_CAPTURE Procedure, 159-10  
FINISH\_REDEF\_TABLE procedure, 110-12  
FIX\_CORRUPT\_BLOCKS procedure, 112-14  
FLUSH Function & Procedure, 121-7  
FLUSH procedure, 43-19, 237-16  
FLUSH\_DATA function and procedure, 106-10  
FLUSH\_DATABASE\_MONITORING\_INFO procedure, 141-66  
FLUSHCACHE Procedure, 49-18  
FONTCLOSE function  
    of HTF package, 182-50  
FONTCLOSE procedure  
    of HTP package, 183-48  
FONTOPEN function  
    of HTF package, 182-51  
FONTOPEN procedure  
    of HTP package, 183-49  
FOPEN function, 223-19  
FOPEN\_NCHAR function, 223-21  
FORCE parameter  
    and job-to-instance affinity, 78-4  
FORMAT function, 255-13  
FORMAT\_CALL\_STACK Function, 156-30  
FORMAT\_CELL function  
    of HTF package, 182-52  
FORMAT\_ERROR\_BACKTRACE function, 156-31  
FORMAT\_ERROR\_STACK function, 156-34  
FORMCHECKBOX function  
    of HTF package, 182-53  
FORMCHECKBOX procedure  
    of HTP package, 183-50  
FORMCLOSE function  
    of HTF package, 182-54  
FORMCLOSE procedure  
    of HTP package, 183-51  
FORMFILE function  
    of HTF package, 182-55  
FORMFILE procedure  
    of HTP package, 183-53  
FORMHIDDEN function  
    of HTF package, 182-56  
FORMHIDDEN procedure  
    of HTP package, 183-54  
FORMIMAGE function  
    of HTF package, 182-57  
FORMIMAGE procedure  
    of HTP package, 183-55  
FORMOPEN function  
    of HTF package, 182-58  
FORMOPEN procedure  
    of HTP package, 183-52  
FORMPASSWORD function  
    of HTF package, 182-59  
FORMPASSWORD procedure  
    of HTP package, 183-56  
FORMRADIO function  
    of HTF package, 182-60  
FORMRADIO procedure

---

of HTP package, 183-57

FORMRESET function

- of HTF package, 182-61

FORMRESET procedure

- of HTP package, 183-58

FORMSELECTCLOSE function

- of HTF package, 182-62

FORMSELECTCLOSE procedure

- of HTP package, 183-59

FORMSELECTOPEN function

- of HTF package, 182-63

FORMSELECTOPEN procedure

- of HTP package, 183-60

FORMSELECTOPTION function

- of HTF package, 182-64

FORMSELECTOPTION procedure

- of HTP package, 183-61

FORMSUBMIT function

- of HTF package, 182-65

FORMSUBMIT procedure

- of HTP package, 183-62

FORMTEXT function

- of HTF package, 182-66

FORMTEXT procedure

- of HTP package, 183-63

FORMTEXTAREA function

- of HTF package, 182-67

FORMTEXTAREA procedure

- of HTP package, 183-64

FORMTEXTAREA2 function

- of HTF package, 182-68

FORMTEXTAREA2 procedure

- of HTP package, 183-65

FORMTEXTAREACLOSE function

- of HTF package, 182-69

FORMTEXTAREACLOSE procedure

- of HTP package, 183-66

FORMTEXTAREAOPEN function

- of HTF package, 182-70

FORMTEXTAREAOPEN procedure

- of HTP package, 183-67

FORMTEXTAREAOPEN2 function

- of HTF package, 182-71

FORMTEXTAREAOPEN2 procedure

- of HTP package, 183-68

FRAGMENT\_DELETE Procedure, 82-43

FRAGMENT\_INSERT Function, 82-44

FRAGMENT\_MOVE Procedure, 82-45

FRAGMENT\_REPLACE Procedure, 82-46

FRAME function

- of HTF package, 182-72

FRAME procedure

- of HTP package, 183-69

FRAMESETCLOSE function

- of HTF package, 182-73

FRAMESETCLOSE procedure

- of HTP package, 183-70

FRAMESETOPEN function

- of HTF package, 182-74

FRAMESETOPEN procedure

---

- of HTP package, 183-71
- FREE\_BLOCKS procedure, 133-13
- FREE\_UNUSED\_USER\_MEMORY Procedure, 131-13
- FREEDOCFRAG procedure, 170-57
- FREEDOCUMENT procedure, 170-58
- FREENODE procedure, 170-59, 170-60
- FREEPARSER procedure, 173-5
- FREEPROCESSOR procedure, 180-7
- FREERESOURCE Procedure, 166-8
- FREESTYLESHEET procedure, 180-8
- FREETEMPORARY Procedures, 82-48
- FREMOVE procedure, 223-22
- FRENAME procedure, 223-23
- FSEEK procedure, 223-24
- functions
  - adding to attribute sets, 65-8, 123-11

## G

- GATHER\_DATABASE\_STATS procedures, 141-67
- GATHER\_DICTIONARY\_STATS procedure, 141-71
- GATHER\_FIXED\_OBJECTS\_STATS procedure, 141-75
- GATHER\_INDEX\_STATS Procedure, 141-76
- GATHER\_OPTIMIZER\_STATS Procedure, 30-3
- GATHER\_SCHEMA\_STATS procedures, 141-78
- GATHER\_SYSTEM\_STATS procedure, 141-82
- GATHER\_TABLE\_STATS procedure, 141-84
- GENERATE\_STATS procedure, 141-88
- GENERATE\_TASK\_NAME Function, 101-22
- GENERATESCHEMAS function, 176-15
- GET function
  - of OWA\_COOKIE package, 187-7
- GET\* member functions
  - of ANYDATA TYPE, 240-9
  - of ANYDATASET TYPE, 241-7
- GET\_ACLOID function, 168-8
- GET\_AGGREGATE\_VALUE function, 123-37
- GET\_ALL procedure, 187-8
- GET\_ALL\_DAD\_ATTRIBUTES Procedure, 63-17
- GET\_ALL\_DAD\_MAPPINGS Procedure, 63-18
- GET\_ALL\_GLOBAL\_ATTRIBUTES Procedure, 63-19
- GET\_ALL\_NAMES member function, 256-13
- GET\_ASH\_QUERY Function, 16-13
- GET\_AUDIT\_COMMIT\_DELAY function, 27-21
- GET\_AUTHENTICATION procedure, 224-50
- GET\_BASE\_TABLE\_NAME member function, 248-9
- GET\_BASE\_TABLE\_OWNER member function, 248-10
- GET\_BODY\_CHARSET procedure, 224-51
- GET\_CAPTURE\_INFO Function, 159-11
- GET\_CATEGORY function, 157-7
- GET\_CATEGORY\_ID function, 89-15
- GET\_CGI\_ENV function, 194-11
- GET\_CLIENT\_ATTRIBUTES Procedure, 29-7
- GET\_CLIENT\_HOSTNAME function, 192-5
- GET\_CLIENT\_IP function, 192-6
- GET\_COLUMN\_STATS procedures, 141-89
- GET\_COMMAND\_TYPE member function, 248-37
- GET\_COMMIT\_SCN member function, 248-37
- GET\_COMMIT\_SCN\_FROM\_POSITION static function, 248-37
- GET\_COMMIT\_TIME member function, 248-38
- GET\_COMMON\_TIME\_ZONES Function, 225-10

---

GET\_COMPATIBLE member function, 248-38  
GET\_COMPONENT function, 255-14  
GET\_COMPONENTS function, 89-16  
GET\_COMPRESSION\_RATIO Procedure, 36-8  
GET\_COMPRESSION\_TYPE Function, 36-10  
GET\_COOKIE\_COUNT function, 224-52  
GET\_COOKIE\_SUPPORT procedure, 224-53  
GET\_COOKIES function, 224-54  
GET\_CPU\_TIME function, 156-35  
GET\_CURRENT\_SCHEMA member function, 248-10  
GET\_DAD\_ATTRIBUTE Function, 63-20  
GET\_DAD\_LIST Procedure, 63-21  
GET\_DBFS\_LINK Functions, 82-49  
GET\_DBFS\_LINK\_STATE Procedure, 82-50  
GET\_DEFAULT\_ISO\_CURRENCY Function, 225-12  
GET\_DEFAULT\_LINGUISTIC\_SORT Function, 225-13  
GET\_DEPENDENCY procedure, 156-36  
GET\_DETAILED\_EXCP\_SUPPORT procedure, 224-55  
GET\_DETAILED\_SQLCODE function, 224-56  
GET\_DETAILED\_SQLERRM function, 224-57  
GET\_DIVERGING\_STATEMENT Function, 160-15  
GET\_EDITION\_NAME member function, 248-11  
GET\_ENCODINGS function, 89-18  
GET\_ENDIANNES Function, 156-37  
GET\_ERROR\_MESSAGE function, 21-34  
GET\_ETAG function, 186-6  
GET\_EXPRESSION function, 45-34  
GET\_EXPRSET\_STATS procedure, 65-23  
GET\_EXTRA\_ATTRIBUTE member function, 248-39  
GET\_FILE procedure, 68-7  
GET\_FIX\_CONTROL Function, 137-27  
GET\_FOLLOW\_REDIRECT procedure, 224-58  
GET\_GLOBAL\_ATTRIBUTE Function, 63-22  
GET\_HASH\_VALUE function, 156-38  
GET\_HEADER procedure, 224-59  
GET\_HEADER\_BY\_NAME procedure, 224-60  
GET\_HEADER\_COUNT function, 224-61  
GET\_HOST\_ADDRESS function, 226-7  
GET\_HOST\_NAME function, 226-8  
GET\_INDEX\_STATS procedures, 141-91  
GET\_INDEXES function, 52-36  
GET\_INFORMATION function, 143-15  
GET\_JAVA\_LOGGING\_LEVEL function, 89-19  
GET\_LEVEL function, 186-7  
GET\_LINE function, 237-17  
GET\_LINE procedure, 100-17, 223-25  
GET\_LINE\_MAP function, 52-38  
GET\_LINE\_NCHAR Function, 237-18  
GET\_LINE\_NCHAR procedure, 223-26  
GET\_LINES procedure, 100-18  
GET\_LOB\_INFORMATION member function, 248-22  
GET\_LOB\_OFFSET member function, 248-22  
GET\_LOB\_OPERATION\_SIZE member procedure, 248-23  
GET\_LOCAL\_LINGUISTIC\_SORTS Function, 225-15  
GET\_LOG procedure, 43-20  
GET\_LOG\_SPEC function, 43-22  
GET\_LOGON\_USER member function, 248-11  
GET\_LONG\_INFORMATION member function, 248-23  
GET\_MESSAGE\_TRACKING function, 144-83  
GET\_MORE\_SOURCE procedure, 52-37



---

GET\_NEXT\_HIT function, 126-11  
GET\_NUMBER\_COL\_CHUNK Procedure, 101-23  
GET\_OBJECT\_NAME member function, 248-40  
GET\_OBJECT\_OWNER member function, 248-40  
GET\_OBJECT\_TYPE member function, 248-11  
GET\_OWA\_SERVICE\_PATH function, 194-12  
GET\_P1\_RESOURCES Procedure, 29-8  
GET\_PACKAGE\_MEMORY\_UTILIZATION Procedure, 131-15  
GET\_PARAM function, 141-94  
GET\_PARAMETER function, 43-23  
GET\_PARAMETER\_VALUE function, 156-39  
GET\_PASSWORD function, 192-7  
GET\_PERSISTENT\_CONN\_COUNT function, 224-62  
GET\_PERSISTENT\_CONN\_SUPPORT procedure, 224-63  
GET\_PERSISTENT\_CONNS procedure, 224-64  
GET\_PLSQL\_LOGGING\_LEVEL function, 89-20  
GET\_PLSQL\_TRACE\_LEVEL function, 151-12  
GET\_POSITION member function, 248-41  
GET\_POST\_PROCESSED\_SOURCE Functions, 105-8  
GET\_PREFS Function, 141-95  
GET\_PROCEDURE function, 194-13  
GET\_PROXY procedure, 224-65  
GET\_RAW function, 223-27, 237-19  
GET\_REC\_ATTRIBUTES Procedure, 18-27  
GET\_REPLAY\_INFO Function, 160-16  
GET\_REPLAY\_TIMEOUT Procedure, 160-17  
GET\_REPORT Function, 16-14  
GET\_RESPONSE function, 224-66  
GET\_RESPONSE\_ERROR\_CHECK procedure, 224-67  
GET\_ROW\_TEXT member procedure, 248-24  
GET\_ROWID function, 190-7  
GET\_ROWID\_CHUNK Procedure, 101-24  
GET\_RUN\_REPORT Function, 72-5  
GET\_RUNTIME\_INFO function, 52-39  
GET\_SCHEME\_NAMES function, 89-21  
GET\_SCN member function, 248-41  
GET\_SCN\_FROM\_POSITION static function, 248-41  
GET\_SCN\_MAPPING procedure, 144-84  
GET\_SESSION\_TIMEOUT function, 122-6  
GET\_SOURCE\_DATABASE\_NAME member function, 248-41  
GET\_SOURCE\_TIME member function, 248-42  
GET\_SQL Function, 137-28  
GET\_SQL\_HASH Function, 156-41  
GET\_STATS\_HISTORY\_AVAILABILITY function, 141-98  
GET\_STATS\_HISTORY\_RETENTION function, 141-99  
GET\_STREAMS\_NAME function, 143-16  
GET\_STREAMS\_TYPE function, 143-17  
GET\_SYSTEM\_STATS procedure, 141-100  
GET\_TABLE\_STATS procedure, 141-102  
GET\_TAG function, 143-18, 144-86  
GET\_TAG member function, 248-42  
GET\_TASK\_REPORT Procedure, 18-28  
GET\_TASK\_SCRIPT Procedure, 18-29  
GET\_TDT\_XML function, 89-22  
GET\_TEXT function, 237-20  
GET\_TEXT\_NCHAR Function, 237-21  
GET\_THREAD\_NUMBER member function, 248-42  
GET\_THRESHOLD procedure, 129-14  
GET\_TIME function, 156-42  
GET\_TIMEOUT function, 122-7

---

GET\_TIMEOUT\_BEHAVIOUR function, 52-40  
GET\_TRANSACTION\_ID member function, 248-42  
GET\_TRANSFER\_TIMEOUT procedure, 224-68  
GET\_TZ\_TRANSITIONS Procedure, 156-43  
GET\_USER\_ID function, 192-8  
GET\_USERID function, 168-9  
GET\_VALIDATOR function, 89-24  
GET\_VALUE function, 52-41  
GET\_VALUE member function, 248-25, 256-14  
GET\_VALUE Procedure, 75-16  
GET\_VALUE\_RAW Procedure, 75-17  
GET\_VALUES member function, 248-26  
GET\_VERSION procedure, 106-11  
GET\_WARNING\_SETTING\_CAT function, 157-8  
GET\_WARNING\_SETTING\_NUM function, 157-9  
GET\_WARNING\_SETTING\_STRING function, 157-10  
GET\_WHERE\_CLAUSE member procedure, 248-26  
GET\_X function, 189-8  
GET\_XML\_INFORMATION member function, 248-28  
GET\_Y function, 189-9  
GETACL Function, 166-9  
GETACLDODFROMRES Function, 166-10  
GETACLDODDOCUMENT function, 163-40  
GETAPPLICATIONDATA Function, 169-16  
GETATTRELEMINFO member function  
  of ANYTYPE TYPE, 242-10  
GETATTRIBUTE function, 170-62  
GETATTRIBUTENODE function, 170-63  
GETATTRIBUTES function, 170-64  
GETAUTHOR Function, 166-11  
GETBASEDOCUMENT Function, 178-10  
GETBLOB function, 245-3, 245-11, 245-21, 245-29  
GETBLOBVAL function, 259-10  
GETBUBLICID function, 170-97  
GETCHARACTERSET Function, 166-12  
GETCHARSET Function, 170-65  
GETCHILDNODES function, 170-66  
GETCHILDROID Function, 169-17  
GETCHILDRENBYTAGNAME function, 170-67  
GETCHILDRESPATHS Function, 163-41  
GETCHUNKSIZE Functions, 82-53  
GETCLOB function, 245-4, 245-12, 245-22, 245-30  
GETCLOBVAL function, 259-11  
GETCOMMENT Function, 166-13  
GETCONTENTBLOB Function, 163-42, 166-14  
GETCONTENTCLOB Function, 163-43, 166-15  
GETCONTENTREF Function, 166-16  
GETCONTENTSBLOBBYRESID function, 165-7  
GETCONTENTSCLOBBYRESID function, 165-8  
GETCONTENTSEXMLBYRESID function, 165-9  
GETCONTENTTYPE Function, 166-17  
GETCONTENTTYPE function, 245-5, 245-13, 245-23, 245-31  
GETCONTENTTYPE Functions, 82-51  
GETCONTENTVARCHAR2 Function, 163-44, 166-19  
GETCONTENTXML Function, 166-18  
GETCONTENTXMLREF Function, 163-45  
GETCONTENTXMLTYPE Function, 163-46  
GETCOUNT member function  
  of ANYDATASET TYPE, 241-10  
GETCREATIONDATE Function, 166-20

---

GETCREATOR Function, 166-21  
GETCURRENTUSER Function, 169-18  
GETCUSTOMMETADATA Function, 166-22  
GETDATA function, 170-68  
GETDCHARSET function, 170-69  
GETDEFAULTACL Procedure, 47-52  
GETDEFAULTASOF Procedure, 47-53  
GETDEFAULTOWNER Procedure, 47-55  
GETDEFAULTPRINCIPAL Procedure, 47-56  
GETDISPLAYNAME Function, 166-23  
GETDOCTYPE function, 170-69, 173-6  
GETDOCUMENT function, 173-7  
GETDOCUMENTELEMENT function, 170-70  
GETDTD function, 174-9  
GETDTD procedure, 174-9  
GETELEMENTSBYTAGNAME function, 170-71, 170-72  
GETENTITIES function, 170-72  
GETEVENT Function, 169-19  
GETEXCEPTIONCONTENT procedure, 174-10, 175-11  
GETEXPANDEDNAME function, 170-74  
GETEXPANDEDNAME procedure, 170-73  
GETEXTERNALURL function, 245-6, 245-14, 245-24, 245-32  
GETFEATURES Function, 48-19  
GETFEATURESBYMOUNT Function, 47-57  
GETFEATURESBYNAME Function, 47-58  
GETFEATURESBYPATH Function, 47-59  
GETFIRST Function, 169-20  
GETFIRSTCHILD function, 170-74  
GETFTPPORT Function, 163-47  
GETHANDLERLIST Function, 169-21  
GETHTTPPORT Function, 163-48  
GETIMPLEMENTATION function, 170-75  
GETINFO member function  
    of ANYTYPE TYPE, 242-9  
GETINTERFACE Function, 169-22  
GETLANGUAGE Function, 166-24, 169-23, 169-27  
GETLASTCHILD function, 170-76  
GETLASTMODIFIER Function, 166-25  
GETLENGTH function, 170-77, 170-78  
GETLENGTH Functions, 82-55  
GETLINK Function, 169-24  
GETLINKNAME Function, 169-25  
GETLISTENERENDPOINT Procedure, 163-49  
GETLISTENERS Function, 118-10  
GETLOCK Function, 169-26  
GETLOCKTOKEN procedure, 163-50  
GETMODIFICATIONDATE Function, 166-26  
GETNAME Function, 169-28  
GETNAME function, 170-79, 170-80  
GETNAMEDITEM function, 170-80  
GETNAMESPACE function, 170-82  
GETNAMESPACE procedure, 170-81  
GETNEXTSIBLING function, 170-82  
GETNODENAME function, 170-84  
GETNODETYPE function, 170-83  
GETNODEVALUE function, 170-85  
GETNODEVALUEASBINARYSTREAM Function & Procedure, 170-86  
GETNODEVALUEASCHARACTERSTREAM Function & Procedure, 170-87  
GETNOTATIONNAME function, 170-88  
GETNOTATIONS function, 170-89

---

GETNUMBERVAL function, 259-13  
GETNUMROWSPROCESSED function, 171-7  
GETNUMROWSPROCESSED procedure, 174-11  
GETOLDRESOURCE Function, 169-30  
GETOPENACCESSMODE Function, 169-31  
GETOPENDENYMODE Function, 169-32  
GETOPTIONS Functions, 82-56  
GETOUTPUTSTREAM Function, 169-33  
GETOWNER Function, 166-27  
GETOWNERDOCUMENT function, 170-91  
GETOWNERELEMENT function, 170-92  
GETPARAMETER Function, 169-34  
GETPARENT Function, 169-35  
GETPARENTNAME Function, 169-36  
GETPARENTNODE function, 170-93  
GETPARENTOID Function, 169-37  
GETPARENTPATH Function, 169-38  
GETPAT procedure, 191-11  
GETPATH Function, 169-39  
GETPATH Procedures, 47-60, 48-20  
GETPATHBYMOUNTID Function, 47-63  
GETPATHBYREPOSID Function, 48-22  
GETPATHBYSTOREID Function, 47-64  
GETPATHNOWAIT Procedure, 48-23  
GETPATHNOWAIT Procedures, 47-65  
GETPERSISTENT static function  
    of ANYTYPE TYPE, 242-8  
GETPREDECESSORS function, 165-10  
GETPREDSBYRESID function, 165-11  
GETPREFIX function, 170-94  
GETPREVIOUSIBLING function, 170-95  
GETPRIVILEGES function, 163-51  
GETPUBLICID function, 170-96, 170-97  
GETQUALIFIEDNAME function, 170-97, 170-98  
GETREFCOUNT Function, 166-28  
GETRELEASEVERSION function, 173-8  
GETREPOSITORYRESCONFIG Function, 118-11  
GETREPOSITORYRESCONFIGPATHS Function, 118-12  
GETRESCONFIG Function, 118-13  
GETRESCONFIGPATHS Function, 118-14  
GETRESOID function, 163-52  
GETRESOURCE Function, 169-40  
GETRESOURCEBYRESID function, 165-12  
GETROOTELEMENT function, 259-14  
GETRUL function, 245-15  
GETSCHEMA Function, 169-41  
GETSCHEMANODE function, 170-98  
GETSCHEMAURL function, 259-15  
GETSOURCE Function, 169-42  
GETSPECIFIED function, 170-99  
GETSTANDALONE function, 170-100  
GETSTATS Procedure, 47-70  
GETSTOREBYMOUNT Function, 47-67  
GETSTOREBYNAME Function, 47-68  
GETSTOREBYPATH Function, 47-69  
GETSTOREID Function, 48-24  
GETSTOREPROPERTY Function, 49-19  
GETSTRINGVAL function, 259-16  
GETSUCCESSORS function, 165-13  
GETSUCCSBYRESID function, 165-14

---

GETSYSTEMID function, 170-101, 170-102  
GETTAGNAME function, 170-102  
GETTARGET function, 170-90  
GETTDEFAULTCONTEXT Procedure, 47-54  
GETTOKENTABLEINFO Procedure & Function, 40-9  
GETTOKENTABLEINFOBYTABLESPACE Procedure, 40-10  
GETTRACE Function, 47-71  
GETTYPE member function  
    of ANYDATA TYPE, 240-12  
    of ANYDATASET TYPE, 241-12  
GETTYPENAME member function  
    of ANYDATA TYPE, 240-13  
    of ANYDATASET TYPE, 241-13  
GETUPDATEBYTECOUNT Function, 169-43  
GETUPDATEBYTEOFFSET Function, 169-44  
GETURL function, 245-7, 245-25, 245-33, 245-37  
GETVALIDATIONMODE function, 173-9  
GETVALUE function, 170-103  
GETVERSION Function, 47-72, 48-25  
GETVERSION function, 170-104  
GETVERSION procedure, 174-12  
GETVERSIONID Function, 166-29  
GETXDB\_TABLESPACE function, 163-53  
GETXDBEVENT Function, 169-45  
GETXML function, 171-8, 174-13, 245-8, 245-16, 245-26, 245-34  
GETXML procedure, 174-13  
GETXMLTYPE function, 170-105, 171-9  
GRANT\_ADMIN\_PRIVILEGE procedure, 146-6  
GRANT\_OBJECT\_PRIVILEGE procedure, 127-21  
GRANT\_PRIVILEGE procedure, 65-24, 123-39  
GRANT\_REMOTE\_ADMIN\_ACCESS procedure, 146-9  
GRANT\_SWITCH\_CONSUMER\_GROUP procedure, 120-3  
GRANT\_SYSTEM\_PRIVILEGE procedure, 67-21, 120-4, 127-23  
GRANTING\_OBJECT\_PRIVILEGE procedure, 67-20

## H

HASACLCHANGED Function, 166-30  
HASATTRIBUTE function, 170-106  
HASAUTHORCHANGED Function, 166-31  
HASBLOBCONTENT Function, 163-54  
HASCHANGED Function, 166-32  
HASCHARACTERSETCHANGED Function, 166-33  
HASCHARCONTENT Function, 163-55  
HASCILDNODES function, 170-108  
HASCOMMENTCHANGED Function, 166-34  
HASCONTENTCHANGED Function, 166-35  
HASCONTENTTYPECHANGED Function, 166-36  
HASCREATIONDATECHANGED Function, 166-37  
HASCREATORCHANGED Function, 166-38  
HASCUSTOMMETADACHANGED Function, 166-39  
HASDISPLAYNAMECHANGED Function, 166-40  
HASFEATURE function, 170-109  
HASLANGUAGECHANGED Function, 166-41  
HASLASTMODIFIERCHANGED Function, 166-42  
HASMODIFICATIONDATECHANGED Function, 166-43  
HASOWNERCHANGED Function, 166-44  
HASREFCOUNTCHANGED Function, 166-45  
HASVERSIONIDCHANGED Function, 166-46  
HASXMLCONTENT Function, 163-56  
HASXMLREFERENCE Function, 163-57

---

HEADCLOSE function  
  of HTF package, 182-75

HEADCLOSE procedure  
  of HTP package, 183-72

HEADER function  
  of HTF package, 182-76

HEADER procedure  
  of HTP package, 183-73

HEADOPEN function  
  of HTF package, 182-77

HEADOPEN procedure  
  of HTP package, 183-74

HELO function and procedure, 235-22

HELP function, 235-23

HR function  
  of HTF package, 182-78

HR procedure  
  of HTP package, 183-75

HTF package, 182-1

HTML tags  
  applet tags  
    functions, 182-7  
    procedures, 183-6  
  atags tags  
    procedures, 183-7  
  character formatting tags  
    functions, 182-9  
    procedures, 183-8  
  form tags  
    functions, 182-7  
    procedures, 183-6  
  frame tags  
    functions, 182-9  
    procedures, 183-8  
  list tags  
    functions, 182-7  
    procedures, 183-6  
  paragraph formatting tags  
    functions, 182-8  
    procedures, 183-7  
  table tags  
    functions, 182-8

HTMLCLOSE function  
  of HTF package, 182-79

HTMLCLOSE procedure  
  of HTP package, 183-76

HTMLDB\_UTIL package documentation, 5-2

HTMLOPEN function  
  of HTF package, 182-80

HTMLOPEN procedure  
  of HTP package, 183-77

HTP package, 183-1

HTTP\_HEADER\_CLOSE procedure, 194-14

HttpUriType, 245-9

HTTPURITYPE function, 245-17

HttpUriType subtype, 245-9  
  CREATEURI function, 245-10  
  GETBLOB function, 245-11  
  GETCLOB function, 245-12  
  GETCONTENTTYPE function, 245-13

---

GETEXTERNALURL function, 245-14  
GETRUL function, 245-15  
GETXML function, 245-16  
HTTPPURITYPE function, 245-17  
methods, 245-9

**I**

IMG function  
  of HTF package, 182-81

IMG procedure  
  of HTP package, 183-78

IMG2 procedure  
  of HTP package, 183-79

IMPLEMENT\_TASK Procedure, 18-31

IMPLEMENT\_TUNING\_TASK Function, 139-42

IMPORT\_AWR Function, 160-18

IMPORT\_AWR Procedure, 159-12

IMPORT\_COLUMN\_STATS procedure, 141-104

IMPORT\_DATABASE\_PREFS Procedure, 141-106

IMPORT\_DATABASE\_STATS procedure, 141-107

IMPORT\_DICTIONARY\_STATS procedure, 141-108

IMPORT\_FIXED\_OBJECTS\_STATS procedure, 141-110

IMPORT\_INDEX\_STATS procedure, 141-111

IMPORT\_SCHEMA\_PREFS Procedure, 141-112

IMPORT\_SCHEMA\_STATS procedure, 141-113

IMPORT\_SQL\_TESTCASE Procedures, 137-29

IMPORT\_SQLWKLD\_SCHEMA Procedure, 18-32

IMPORT\_SQLWKLD\_SQLCACHE Procedure, 18-34

IMPORT\_SQLWKLD\_STS Procedure, 18-36

IMPORT\_SQLWKLD\_SUMADV Procedure, 18-38

IMPORT\_SQLWKLD\_USER Procedure, 18-40

IMPORT\_SYSTEM\_STATS procedure, 141-115

IMPORT\_TABLE\_PREFS Procedure, 141-116

IMPORT\_TABLE\_STATS procedure, 141-117

IMPORT\_XML procedure, 41-16, 41-49

IMPORTNODE function, 170-110

inbound servers  
  DBMS\_STREAMS\_ADM package, 144-7  
  rules, 144-15

INCIDENTID\_2\_SQL Procedure, 137-31

INCLUDE\_EXTRA\_ATTRIBUTES procedure, 32-33

INDEX\_PARAMETERS procedure, 65-25

INDEX\_TABLE\_TYPE Table Type, 156-9

INIT\_CLEANUP procedure, 27-22

INITFS Procedure, 50-11

INITIALIZE function, 52-43

INITIALIZE procedure, 108-7

INITIALIZE\_CUBE\_UPGRADE procedure, 41-15, 41-16, 41-52

INITIALIZE\_REPLAY Procedure, 160-19

INPLACEEVOLVE Procedure, 176-16

INSERT\_AUTOBIN\_NUM\_EQWIDTH procedure, 45-35

INSERT\_BIN\_CAT\_FREQ procedure, 45-39

INSERT\_BIN\_NUM\_EQWIDTH procedure, 45-44

INSERT\_BIN\_NUM\_QTILE procedure, 45-48

INSERT\_BIN\_SUPER procedure, 45-51

INSERT\_CLIP\_TRIM\_TAIL procedure, 45-55

INSERT\_CLIP\_WINSOR\_TAIL procedure, 45-58

INSERT\_FINDING\_DIRECTIVE Procedure, 16-15

INSERT\_MISS\_CAT\_MODE procedure, 45-61

INSERT\_MISS\_NUM\_MEAN procedure, 45-64

---

INSERT\_NORM\_LIN\_MINMAX procedure, 45-67  
INSERT\_NORM\_LIN\_SCALE procedure, 45-70  
INSERT\_NORM\_LIN\_ZSCORE procedure, 45-73  
INSERT\_PARAMETER\_DIRECTIVE Procedure, 16-16  
INSERT\_SEGMENT\_DIRECTIVE Procedure, 16-17  
INSERT\_SQL\_DIRECTIVE Procedure, 16-19  
INSERTBEFORE function, 170-111  
INSERTDATA procedure, 170-112  
INSERTXML function, 175-12, 177-10  
INSTANCE procedure, 78-9  
INSTANCE\_RECORD Record Type, 156-7  
INSTANCE\_TABLE Table Type, 156-10  
instantiation  
    aborting database preparation, 32-7  
    aborting schema preparation, 32-8  
    aborting table preparation, 32-10  
    global SCN, 21-50  
    preparing a database for, 32-35  
    preparing a schema for, 32-36  
    preparing a table for, 32-37, 32-38  
    schema SCN, 21-66  
    table SCN, 21-69  
INSTR Functions, 82-57  
INTEGER\_ARRAY Table Type, 131-6  
INTERNAL\_VERSION\_CHECK function, 106-12  
internet addressing  
    using UTL\_INADDR, 226-1  
INTERRUPT\_ANALYSIS\_TASK Procedure, 138-14  
INTERRUPT\_DIAGNOSIS\_TASK Procedure, 137-32  
INTERRUPT\_TASK Procedure, 18-42  
INTERRUPT\_TUNING\_TASK procedure, 139-43  
INTERVAL procedure, 78-10  
INTERVAL\_DAY\_TO\_SECOND\_TABLE, 136-38  
INTERVAL\_YEAR\_TO\_MONTH\_TABLE Table Type, 136-39  
INVALIDATE Functions & Procedures, 121-8  
INVALIDATE Procedure, 156-44  
INVALIDATE\_OBJECT Functions & Procedures, 121-9  
IS\_BIT\_SET Function, 156-47  
IS\_CLEANUP\_INITIALIZED function, 27-24  
IS\_CLUSTER\_DATABASE function, 156-48  
IS\_HIERARCHY\_ENABLED function, 168-10  
IS\_LOCATOR function, 220-3  
IS\_NULL\_TAG member function, 248-42  
IS\_OPEN function, 136-78, 223-28  
IS\_REPLAY\_PAUSED Procedure, 160-20  
IS\_ROLE\_ENABLED Function, 131-16  
IS\_SESSION\_ALIVE Function, 131-17  
IS\_TRIGGER\_FIRE\_ONCE function, 53-12  
ISFOLDER Function, 163-58, 166-47  
ISFRAGMENT function, 259-17  
ISINDEX function  
    of HTF package, 182-83  
ISINDEX procedure  
    of HTP package, 183-80  
ISNULL Function, 166-48, 169-46  
ISNULL function, 170-113  
ISOPEN function, 221-7  
ISOPEN Functions, 82-59  
ISSCHEMABASED function, 259-18  
ISSCHEMAVALID function, 259-19



---

ISSHEMAVALIDATED function, 259-20  
ISTEMPORARY Functions, 82-60  
ITALIC function  
    of HTF package, 182-84  
ITALIC procedure  
    of HTP package, 183-81  
ITEM Functions, 170-116

## J

JARO\_WINKLER Function, 230-8  
JARO\_WINKLER\_SIMILARITY Function, 230-9  
JOB constructor function, 128-8  
JOB object type, 128-5, 128-6  
JOB\_ARRAY table type, 128-5, 128-9  
JOBARG constructor function, 128-3  
JOBARG object type, 128-3  
JOBARG\_ARRAY table type, 128-4  
JOBATTR constructor function, 128-10  
JOBATTR object type, 128-10  
JOBATTR\_ARRAY table type, 128-11

## K

KBD function  
    of HTF package, 182-85  
KBD procedure  
    of HTP package, 183-82  
KEEP procedure, 132-7  
KEEP\_COLUMNS member procedure, 248-28  
KEEP\_COLUMNS procedure, 144-87  
KEYBOARD function  
    of HTF package, 182-86  
KEYBOARD procedure  
    of HTP package, 183-83

## L

LAPACK Driver Routines (Linear Equations) Subprograms, 231-11  
LAPACK Driver Routines (LLS and Eigenvalue Problems), 231-12  
LAPACK\_GBSV Procedures, 231-76  
LAPACK\_GEES Procedures, 231-78  
LAPACK\_GEEV Procedures, 231-90  
LAPACK\_GELS Procedures, 231-80  
LAPACK\_GESDD Procedures, 231-82  
LAPACK\_GESV Procedures, 231-85  
LAPACK\_GESVD Procedures, 231-87  
LAPACK\_GTSV Procedures, 231-93  
LAPACK\_PBSV Procedures, 231-95  
LAPACK\_POSV Procedures, 231-97  
LAPACK\_PPSV Procedures, 231-99  
LAPACK\_PTSV Procedures, 231-101  
LAPACK\_SBEV Procedures, 231-103  
LAPACK\_SBEVD Procedures, 231-105  
LAPACK\_SPEV Procedures, 231-107  
LAPACK\_SPEVD Procedures, 231-109  
LAPACK\_SPSV Procedures, 231-111  
LAPACK\_STEV Procedures, 231-113  
LAPACK\_STEVD Procedures, 231-115  
LAPACK\_SYEV Procedures, 231-117  
LAPACK\_SYEVD Procedures, 231-119  
LAPACK\_SYSV Procedures, 231-121

---

LAST\_ERROR\_POSITION function, 136-79  
 LAST\_ROW\_COUNT function, 136-80  
 LAST\_ROW\_ID function, 136-81  
 LAST\_SQL\_FUNCTION\_CODE function, 136-82  
 LCR\$\_DDL\_RECORD type, 248-6  
 LCR\$\_ROW\_LIST type, 248-47  
 LCR\$\_ROW\_RECORD type, 248-15  
 LCR\$\_ROW\_UNIT type, 248-48  
     GET\_LOB\_INFORMATION member function, 248-22  
     GET\_LOB\_OPERATION\_SIZE member procedure, 248-23  
     GET\_LONG\_INFORMATION member function, 248-23  
     GET\_POSITION member function, 248-41  
     GET\_ROW\_TEXT member procedure, 248-24  
     GET\_WHERE\_CLAUSE member procedure, 248-26  
     SET\_LOB\_INFORMATION member procedure, 248-29  
     SET\_LOB\_OPERATION\_SIZE member procedure, 248-31  
 LCR\_TO\_XML Function, 91-22  
 LENGTH function, 232-29  
 LINE function  
     of HTF package, 182-87  
 LINE procedure  
     of HTP package, 183-84  
 LINK Procedures, 163-59  
 LINKREL function  
     of HTF package, 182-88  
 LINKREL procedure  
     of HTP package, 183-85  
 LINKREV function  
     of HTF package, 182-89  
 LINKREV procedure  
     of HTP package, 183-86  
 LIST Function, 47-73, 48-26  
 LIST\_CONTEXT Procedures, 131-18  
 LISTALLCONTENT Function, 47-75  
 LISTALLPROPERTIES Function, 47-74  
 LISTHEADER function  
     of HTF package, 182-90  
 LISTHEADER procedure  
     of HTP package, 183-87  
 LISTINGCLOSE function  
     of HTF package, 182-91  
 LISTINGCLOSE procedure  
     of HTP package, 183-88  
 LISTINGOPEN function  
     of HTF package, 182-92  
 LISTINGOPEN procedure  
     of HTP package, 183-89  
 LISTITEM function  
     of HTF package, 182-93  
 LISTITEM procedure  
     of HTP package, 183-90  
 LISTMOUNTS Function, 47-76  
 LISTPRINT procedure, 194-15  
 LISTSTORES Function, 47-77  
 LNAME\_ARRAY Table Type, 131-7, 156-11  
 LOAD\_PLANS\_FROM\_CURSOR\_CACHE Functions, 135-17  
 LOAD\_PLANS\_FROM\_SQLSET Function, 135-19  
 LOAD\_SQLSET procedure, 139-44  
 LOAD\_SQLSET\_FROM\_TCB Function, 137-33  
 LOAD\_TABLE procedure

---

- of DBMS\_HS\_PARALLEL package, 74-8
- LOADBLOBFROMFILE Procedure, 82-61
- LOADCLOBFROMFILE Procedure, 82-63
- LOADFROMFILE Procedure, 82-67
- LOBs
  - DBMS\_LOB package, 82-1
- LOCAL\_TRANSACTION\_ID function, 152-11
- LOCK\_MAP procedure, 142-9
- LOCK\_OBJECT procedure, 234-10
- LOCK\_PARTITION\_STATS Procedure, 141-119
- LOCK\_SCHEMA\_STATS procedure, 141-120
- LOCK\_TABLE\_STATS procedure, 141-121
- LOCKPATH Procedure, 47-78, 48-27
- LOCKRESOURCE function, 163-60
- log apply services
  - managing initialization parameters for logical standby databases, 86-3
- logical change records (LCRs)
  - DDL LCRs, 248-6
    - getting base table name, 248-9
    - getting base table owner, 248-10
    - getting current schema, 248-10
    - getting edition name, 248-11
    - getting logon user name, 248-11
    - getting object type, 248-11
    - setting base table name, 248-11
    - setting base table owner, 248-11
    - setting current schema, 248-12
    - setting DDL text, 248-12
    - setting logon user, 248-13
    - setting object type, 248-13
  - determining if tag is NULL, 248-42
  - executing, 248-9, 248-20
  - extra attributes
    - excluding, 32-33
    - including, 32-33
- GET\_THREAD\_NUMBER member function, 248-42
- getting command type, 248-37
- getting commit SCN, 248-37
- getting commit SCN from position, 248-37
- getting commit time, 248-38
- getting compatibility information, 248-38
- getting extra attributes, 248-39
- getting LCR creation time, 248-42
- getting object name, 248-40
- getting object owner, 248-40
- getting SCN, 248-41
- getting SCN from position, 248-41
- getting source database name, 248-41
- getting tag, 248-42
- getting transaction identifier, 248-42
- LCR\$\_DDL\_RECORD type, 248-6
- LCR\$\_ROW\_LIST type, 248-47
- LCR\$\_ROW\_RECORD type, 248-15
- LCR\$\_ROW\_UNIT type, 248-48
- row LCRs, 248-15
  - adding value to column, 248-18
  - converting LONG to LOB, 248-20
  - deleting value to column, 248-20
  - getting column value, 248-25
  - getting list of column values, 248-26

- getting LOB offset, 248-22
- getting XML information, 248-28
- keeping columns, 248-28
- renaming column, 248-29
- setting column value, 248-31
- setting list of column values, 248-33
- setting LOB offset, 248-30
- setting XML information, 248-35
- setting command type, 248-43
- setting extra attributes, 248-43
- setting object name, 248-44
- setting object owner, 248-45
- setting source database name, 248-45
- setting tag, 248-45
- types, 248-1

logical standby databases

- managing with DBMS\_LOGSTDBY package, 86-3

logs

- Cube Build, 43-8
- Cube Dimension Compile, 43-10
- Cube Operations, 43-11
- Cube Rejected Records, 43-12

LZ\_COMPRESS functions and procedures, 221-8

LZ\_COMPRESS\_ADD procedure, 221-10

LZ\_COMPRESS\_CLOSE procedure, 221-11

LZ\_COMPRESS\_OPEN function, 221-12

LZ\_UNCOMPRESS functions and procedures, 221-13

LZ\_UNCOMPRESS\_CLOSE procedure, 221-16

LZ\_UNCOMPRESS\_EXTRACT procedure, 221-14

LZ\_UNCOMPRESS\_OPEN function, 221-15

## M

MAIL function and procedure, 235-24

MAILTO function

- of HTF package, 182-94

MAILTO procedure

- of HTP package, 183-91

MAINTAIN\_CHANGE\_TABLE procedure, 144-90

MAINTAIN\_GLOBAL procedure, 144-97

MAINTAIN\_SCHEMAS procedure, 144-101

MAINTAIN\_SIMPLE\_TABLESPACE procedure, 144-106

MAINTAIN\_SIMPLE\_TTS procedure, 144-111

MAINTAIN\_TABLES procedure, 144-114

MAINTAIN\_TABLESPACES procedure, 144-119

MAINTAIN\_TTS procedure, 144-126

MAKE\_DATA\_BLOCK\_ADDRESS function, 156-49

MAKEATTR function, 170-117

MAKECDATASECTION function, 170-118

MAKECHARACTERDATA function, 170-119

MAKECOMMENT function, 170-120

MAKEDOCUMENT Function, 166-49

MAKEDOCUMENT function, 170-121

MAKEDOCUMENTFRAGMENT function, 170-122

MAKEDOCUMENTTYPE function, 170-123

MAKEELEMENT function, 170-124

MAKEENTITY function, 170-125

MAKEENTITYREFERENCE function, 170-126

MAKENODE function, 170-127, 170-130

MAKENOTATION function, 170-130

MAKEPROCESSINGINSTRUCTION function, 170-131

---

MAKETEXT function, 170-132  
 MAKEVERSIONED function, 165-15  
 MAP\_ALL function, 142-10  
 MAP\_DAD Procedure, 63-23  
 MAP\_ELEMENT function, 142-11  
 MAP\_FILE function, 142-12  
 MAP\_OBJECT function, 142-13  
 MAPCLOSE function  
     of HTF package, 182-95  
 MAPCLOSE procedure  
     of HTP package, 183-92  
 MAOPEN function  
     of HTF package, 182-96  
 MAOPEN procedure  
     of HTP package, 183-93  
 MARK\_RECOMMENDATION Procedure, 18-43  
 MARKHOT Procedure, 132-9  
 MATCH function, 191-12  
 MATERIALIZE\_DEFERRED\_SEGMENTS Procedure, 134-12  
 materialized view logs  
     master table  
         purging, 93-15, 93-16, 93-17  
 materialized views  
     comparing, 35-1  
     converting from table- to cube-organized, 41-36  
     optimizing cube organized, 42-1  
     refreshing, 93-18, 93-20, 93-21  
 materialized views (cube organized), 41-4  
 MEMORY\_REPORT Procedure, 121-10  
 MENULISTCLOSE function  
     of HTF package, 182-97  
 MENULISTCLOSE procedure  
     of HTP package, 183-94  
 MENULISTOPEN function  
     of HTF package, 182-98  
 MENULISTOPEN procedure  
     of HTP package, 183-95  
 merge streams, 144-130, 144-133  
 MERGE\_COL\_USAGE Procedure, 141-122  
 MERGE\_STREAMS procedure, 144-130  
 MERGE\_STREAMS\_JOB procedure, 144-133  
 MERGEXLIFF Functions, 178-11  
 MESSAGE\_PROPERTIES\_ARRAY\_T Type, 243-28  
 MESSAGE\_PROPERTIES\_T Type, 243-24  
 messaging client  
     messaging client user, 144-8  
     rules  
         for LCRs, 144-14  
         for user messages, 144-14  
 META function  
     of HTF package, 182-99  
 META procedure  
     of HTP package, 183-96  
 MG2 function  
     of HTF package, 182-82  
 MGD\_ID constructor function, 255-9  
 MGD\_ID object type, 255-7  
     FORMAT function, 255-13  
     GET\_COMPONENT function, 255-14  
     MGD\_ID constructor function, 255-9

---

TO\_STRING function, 255-16  
TRANSLATE function, 255-17  
MGD\_ID object types, 255-1  
MGD\_ID\_COMPONENT object type, 255-5  
MGD\_ID\_COMPONENT\_VARRAY object type, 255-6  
MIGRATE\_STORED\_OUTLINE Function, 135-20  
migration  
    post-migration actions, 96-1  
MIME\_HEADER procedure, 194-16  
MIMEHEADER\_DECODE function, 222-5  
MIMEHEADER\_ENCODE function, 222-6  
model transparency, 45-8  
MODIFY\_BASELINE\_WINDOW\_SIZE Procedure, 161-40  
MODIFY\_OPERATOR\_LIST procedure, 65-27  
MODIFY\_PACKAGE\_STATE Procedure, 131-19  
MODIFY\_SERVICE Procedure, 130-14  
MODIFY\_SNAPSHOT\_SETTINGS procedure, 161-38  
MODIFY\_TRANSFORMATION procedure, 153-6  
MODIFYPARAMETER Procedure, 172-9  
MOUNT\_T Record Type, 47-25  
MOUNTS\_T Table Type, 47-32  
MOUNTSTORE Procedure, 47-79  
MOVE\_TO\_DBFS\_LINK Procedures, 82-69  
MOVEXDB\_TABLESPACE Procedure, 164-8  
MSGID\_ARRAY\_T Type, 243-29  
MV\_CUBE\_ADVICE function, 42-5

## **N**

NAME\_ARRAY Table Type, 156-12  
NAME\_RESOLVE procedure, 156-50  
NAME\_TOKENIZE procedure, 156-52  
NAMELIST Table Type, 135-8  
NAMESPACE function, 259-12  
NAMESPACEIDTABLE Procedure, 40-11  
NEW\_LINE procedure, 100-19, 223-29  
NEW\_ROW\_LIST function and procedure, 193-6  
NEWCONTEXT function, 171-10, 174-14, 175-13, 177-11  
NEWCONTEXTFROMHIERARCHY Function, 171-11  
NEWDOMDOCUMENT function, 170-133  
NEWPARSER function, 173-10  
NEWPROCESSOR function, 180-9  
NEWSTYLESHEET function, 180-10  
NEXT\_DATE procedure, 78-11  
NEXT\_ITEM\_TYPE function, 103-21  
NL function  
    of HTF package, 182-100  
NL procedure  
    of HTP package, 183-97  
NOBR function  
    of HTF package, 182-101  
NOBR procedure  
    of HTP package, 183-98  
NOFRAMESCLOSE function  
    of HTF package, 182-102  
NOFRAMESCLOSE procedure  
    of HTP package, 183-99  
NOFRAMESOPEN function  
    of HTF package, 182-103  
NOFRAMESOPEN procedure  
    of HTP package, 183-100

---

NOOP function and procedure, 235-25  
NOOP Functions, 26-7  
NORMAL function, 108-8  
NORMAL\_DIST\_FIT procedure, 140-4  
NORMALIZE procedure, 170-134  
NORMALIZEPATH Functions, 47-81  
NUMBER\_ARRAY Table Type, 156-13  
NUMBER\_TABLE Table Type, 136-40  
NVARRAY\_ADD Procedure, 91-23  
NVARRAY\_FIND\_NAME Function, 91-24  
NVARRAY\_FIND\_NAME\_TYPE Function, 91-25  
NVARRAY\_GET Function, 91-26  
NVARRAY\_GET\_BOOLEAN, 91-27  
NVARRAY\_GET\_BYTE, 91-28  
NVARRAY\_GET\_DATE Function, 91-29  
NVARRAY\_GET\_DOUBLE Function, 91-30  
NVARRAY\_GET\_FLOAT Function, 91-31  
NVARRAY\_GET\_INTEGER, 91-32  
NVARRAY\_GET\_LONG Function, 91-33  
NVARRAY\_GET\_RAW Function, 91-34  
NVARRAY\_GET\_SHORT, 91-35  
NVARRAY\_GET\_TEXT Function, 91-36

## O

OBJECT\_DEPENDENT\_SEGMENTS function, 133-16  
OBJECT\_GROWTH\_TREND function, 133-18  
ogical change records (LCRs)  
    tracking, 144-83, 144-167  
OLAP aggregation operators, 41-24  
OLAP logs  
    Cube Build, 43-8  
    Cube Dimension Compile, 43-10  
    Cube Operations, 43-11  
    Cube Rejected Records, 43-12  
OLAP metadata  
    upgrading to OLAP 11g, 41-12  
OLAP PL/SQL packages  
    DBMS\_CUBE\_LOG, 43-1  
OLAP templates  
    validation, 41-58  
    writing to XML files, 41-47  
OLAP\_DBA role, 41-3  
OLAP\_USER role, 41-3  
OLD\_CURRENT\_SCHEMA Function, 156-53  
OLD\_CURRENT\_USER Function, 156-54  
OLISTCLOSE function  
    of HTF package, 182-104  
OLISTCLOSE procedure  
    of HTP package, 183-101  
OLISTOPEN function  
    of HTF package, 182-105  
OLISTOPEN procedure  
    of HTP package, 183-102  
ONLINE\_INDEX\_CLEAN Function, 112-15  
OPEN Procedures, 82-70  
OPEN\_CONNECTION function, 237-22  
OPEN\_CONNECTION functions, 235-26  
OPEN\_CURSOR Function, 75-18  
OPEN\_CURSOR function, 136-83  
OPEN\_DATA function and procedure, 235-28

---

OPEN\_FBA Procedure, 70-8  
optimizer statistics, 31-1  
OR REPLACE clause  
    for creating packages, 1-5  
Oracle Data Mining, 44-1  
Oracle Streams  
    administrator  
        granting privileges, 146-6  
        revoking privileges, 146-10  
    advisors, 145-1  
    change data capture  
        configuring, 144-90  
    compatibility, 143-6, 143-7, 143-8, 143-9, 143-10, 248-38  
    creating queues, 144-172  
    data dictionary  
        removing information, 144-146  
    messaging  
        notification, 144-161  
    privileges, 146-1  
    replication  
        configuring, 144-16, 144-97, 144-101, 144-111, 144-114, 144-126, 144-136, 144-141  
Oracle-supplied types  
    logical change record (LCR) types, 248-1  
    rule types, 256-1  
ORD\_DICOM package documentation, 184-2  
ORD\_DICOM\_ADMIN package documentation, 185-2  
outbound servers  
    rules, 144-14  
OVERLAY function, 232-30  
OVERRIDE\_PRIORITY Procedures, 29-9  
OWA\_CACHE package, 186-1  
OWA\_COOKIE package, 187-1  
OWA\_CUSTOM package, 188-1  
OWA\_IMAGE package, 189-1  
OWA\_OPT\_LOCK package, 190-1  
OWA\_PATTERN package, 191-1  
OWA\_SEC package, 192-1  
OWA\_TEXT package, 193-1  
OWA\_UTIL package, 194-1

## **P**

PACK\_MESSAGE procedures, 103-22  
PACK\_STGTAB\_BASELINE Function, 135-22  
PACK\_STGTAB\_SQLPATCH Procedure, 137-34  
PACK\_STGTAB\_SQLPROF Procedure, 139-48  
PACK\_STGTAB\_SQLSET Procedure, 139-49  
package  
    DBMS\_EXPFIL, 65-1  
    DBMS\_ODCI, 97-1  
    DBMS\_RLMGR, 123-1  
    DBMS\_XDB, 163-1  
    DBMS\_XMLDOM, 170-1  
    DBMS\_XMLGEN, 171-1  
    DBMS\_XMLPARSER, 173-1  
    DBMS\_XMLQUERY, 174-1  
    DBMS\_XMLSAVE, 175-1  
    DBMS\_XMLSCHEMA, 176-1  
    UriFactory, 245-36  
Package - UriFactory, 245-36  
package overview, 1-2



---

package variables  
  i\_am\_a\_refresh, 93-13

packages  
  creating, 1-5  
  DBMS\_MGD\_ID\_UTL, 89-1  
  referencing, 1-8  
  where documented, 1-9

PARA function  
  of HTF package, 182-106

PARA procedure  
  of HTP package, 183-103

PARAGRAPH function  
  of HTF package, 182-107

PARAGRAPH procedure  
  of HTP package, 183-104

PARAM function  
  of HTF package, 182-108

PARAM procedure  
  of HTP package, 183-105

PARSE Procedure, 75-19

PARSE procedure, 136-84, 173-11

PARSEBUFFER procedure, 173-12

PARSECLOB procedure, 173-13

PARSEDTD procedure, 173-14

PARSEDTDBUFFER procedure, 173-15

PARSEDTDCLOB procedure, 173-16

PATCHREPOSITORYRESCONFIGLIST Procedure, 118-15

PATH\_ITEM\_T Record Type, 47-26

PATH\_ITEMS\_T Table Type, 47-33

PATHIDTABLE Function, 40-12

PAUSE\_PROFILER function and procedure, 106-13

PAUSE\_REPLAY Procedure, 160-21

PIECEWISE member procedure  
  of ANYDATA TYPE, 240-14  
  of ANYDATASET TYPE, 241-14

PING procedure, 52-45

PLAINTEXT function  
  of HTF package, 182-109

PLAINTEXT procedure  
  of HTP package, 183-106

plan stability, 99-3

PL/SQL  
  datatypes, 57-7  
  numeric codes for, 57-9  
  functions  
    DBMS\_MGWADM package subprograms, 90-14  
    DBMS\_MGWMSG package subprograms, 91-21  
  procedures  
    DBMS\_MGWADM package subprograms, 90-14  
    DBMS\_MGWMSG package subprograms, 91-21

PL/SQL package  
  DBMS\_HS\_PARALLEL, 74-1

PLSQL\_TRACE\_VERSION procedure, 151-13

pointer to  
  CTX\_ADM package, 6-1

point-in-time recovery  
  Oracle Streams, 144-84

POISSON\_DIST\_FIT procedure, 140-5

POPULATE\_DIVERGENCE Procedure, 160-22

PORT\_STRING function, 156-55

---

POST\_INSTANTIATION\_SETUP procedure, 144-136  
PRE\_INSTANTIATION\_SETUP procedure, 144-141  
PRECLOSE function  
    of HTF package, 182-110  
PRECLOSE procedure  
    of HTP package, 183-107  
predictive analytics, 104-1  
PREOPEN function  
    of HTF package, 182-111  
PREOPEN procedure  
    of HTP package, 183-108  
PREPARE Procedure, 160-23  
PREPARE\_COLUMN\_VALUES procedures, 141-123  
PREPARE\_COLUMN\_VALUES\_NVARCHAR2 procedure, 141-125  
PREPARE\_COLUMN\_VALUES\_ROWID procedure, 141-127  
PREPARE\_GLOBAL\_INSTANTIATION procedure, 32-35  
PREPARE\_SCHEMA\_INSTANTIATION procedure, 32-36  
PREPARE\_SYNC\_INSTANTIATION function, 32-37  
PREPARE\_TABLE\_INSTANTIATION procedure, 32-38  
PRINT function  
    of HTF package, 182-112  
PRINT procedure  
    of HTP package, 183-109  
PRINT\_BACKTRACE procedure, 52-46  
PRINT\_CGL\_ENV procedure, 194-17  
PRINT\_INSTANTIATIONS procedure, 52-47  
PRINT\_MULTI procedure, 193-7  
PRINT\_POST\_PROCESSED\_SOURCE Procedure, 105-10  
PRINT\_ROW\_LIST procedure, 193-8  
PRINTS procedure  
    of HTP package, 183-110  
privileges  
    granting, 65-24  
    Oracle Streams administrator, 146-6, 146-10  
    revoking, 65-28  
PRN function  
    of HTF package, 182-113  
PRN procedure  
    of HTP package, 183-111  
PROBE\_VERSION procedure, 52-48  
PROCESS\_RECORDING Procedure, 160-27  
PROCESS\_RULES procedure, 123-40  
PROCESSLINKS Procedure, 163-62  
PROCESSXSL function, 180-11  
PROGRAM\_INFO Record Type, 52-17  
PROP\_ITEM\_T Record Type, 47-28  
PROP\_ITEMS\_T Table Type, 47-34  
PROPAGATEORIGINALEXCEPTION procedure, 174-15, 175-14  
propagations  
    altering, 107-6  
    creating, 107-8, 144-30, 144-39, 144-46, 144-55, 144-65  
    DBMS\_PROPAGATION\_ADM package, 107-1  
    dropping, 107-11  
    propagation user, 144-6  
    rules  
        defining global, 144-30  
        defining message, 144-39  
        defining schema, 144-46  
        defining subset, 144-55  
        defining table, 144-65

---

- for LCRs, 144-11
  - for user messages, 144-12
- starting, 107-13
- stopping, 107-14
- PROPANY Functions, 47-82
- PROPERTIES\_T Table Type, 47-35
- PROPERTIESH2T Function, 47-83
- PROPERTIEST2H Function, 47-84
- PROPERTY\_T Record Type, 47-29
- PROPNUMBER Function, 47-85
- PROPRAW Function, 47-86
- PROPTIMESTAMP Function, 47-87
- PROPVARCHAR2 Function, 47-88
- PS procedure
  - of HTP package, 183-112
- PUBLISH\_PENDING\_STATS Procedure, 141-129
- PULL\_SIMPLE\_TABLESPACE procedure, 149-30
- PULL\_TABLESPACES procedure, 149-32
- PURGE Procedure, 132-10
- PURGE procedure, 103-24
- PURGE\_COMPARISON Procedure, 35-25
- PURGE\_EVENTS procedure, 123-42
- PURGE\_FILE\_GROUP procedure, 67-22
- PURGE\_LOST\_DB\_ENTRY procedure, 152-12
- PURGE\_MIXED procedure, 152-14
- PURGE\_PROCESSED\_CHUNKS Procedure, 101-25
- PURGE\_QUEUE\_TABLE Procedure, 23-42
- PURGE\_SOURCE\_CATALOG procedure, 144-146
- PURGE\_STATS procedure, 141-130
- PURGEALL Procedure, 47-89, 48-28
- PURGELDAPCACHE function, 168-11
- PURGEPATH Procedure, 47-90, 48-29
- PURGERESOURCEMETADATA Procedure, 163-63
- PURGESCHEMA Procedure, 176-18
- purging
  - the subscription window, 34-3
- PUT, 223-31
- PUT procedure, 223-30
- PUT procedures, 100-20
- PUT\_FILE procedure, 68-9
- PUT\_LINE Procedure, 223-31
- PUT\_LINE procedures, 100-21
- PUT\_LINE\_NCHAR procedure, 223-32
- PUT\_NCHAR procedure, 223-33
- PUT\_RAW function, 223-37
- PUTF procedure, 223-34
- PUTF\_NCHAR procedure, 223-36
- PUTPATH Procedures, 47-91, 48-30

## Q

- QNAMEIDTABLE Function, 40-13
- QUALIFIED\_SQL\_NAME Function, 26-8
- queues
  - AnyData
    - creating, 144-172
    - removing, 144-149
- QUICK\_TUNE Procedure, 18-44
- QUIT function and procedure, 235-29
- QUOTED\_PRINTABLE\_DECODE function, 222-7
- QUOTED\_PRINTABLE\_ENCODE function, 222-8

---

## R

RANDOM procedure, 108-9  
RCPT function, 235-30  
RE\$ATTRIBUTE\_VALUE type, 256-7  
RE\$ATTRIBUTE\_VALUE\_LIST type, 256-8  
RE\$COLUMN\_VALUE type, 256-9, 256-15  
RE\$COLUMN\_VALUE\_LIST type, 256-10  
RE\$NAME\_ARRAY type, 256-11  
RE\$NV\_ARRAY type, 256-12  
RE\$NV\_LIST type, 256-13  
    ADD\_PAIR member procedure, 256-13  
    GET\_ALL\_NAMES member function, 256-13  
    GET\_VALUE member function, 256-14  
    REMOVE\_PAIR member procedure, 256-14  
RE\$RULE\_HIT type, 256-16  
RE\$RULE\_HIT\_LIST type, 256-17  
RE\$TABLE\_ALIAS type, 256-18  
RE\$TABLE\_ALIAS\_LIST type, 256-19  
RE\$TABLE\_VALUE type, 256-20  
RE\$TABLE\_VALUE\_LIST type, 256-21  
RE\$VARIABLE\_TYPE type, 256-22  
RE\$VARIABLE\_TYPE\_LIST type, 256-24  
RE\$VARIABLE\_VALUE type, 256-25  
RE\$VARIABLE\_VALUE\_LIST type, 256-26  
READ Procedures, 82-72  
READ\_CLIENT\_INFO procedure, 20-7  
READ\_LINE function, 237-25  
READ\_LINE procedure  
    of UTL\_HTTP, 224-69  
READ\_MODULE procedure, 20-8  
READ\_ONLY procedure, 152-15  
READ\_RAW function, 237-27  
READ\_RAW procedure  
    of UTL\_HTTP, 224-70  
READ\_TEXT function, 237-28  
READ\_TEXT procedure  
    of UTL\_HTTP, 224-71  
READ\_WRITE procedure, 152-16  
READ2CLOB function, 180-13  
Real-time SQL Monitoring Subprograms, 139-14  
REASSOCIATE\_FBA Procedure, 70-9  
REBUILD\_FREELISTS procedure, 112-16  
REBUILDHIERARCHICALINDEX Procedure, 164-9  
RECEIVE\_MESSAGE function, 103-25  
RECHECK Function, 35-26  
RECOMP\_PARALLEL procedure, 233-7  
RECOMP\_SERIAL procedure, 233-8  
RECONFIGCACHE Procedure, 49-20  
REDIRECT\_URL procedure, 194-18  
refresh  
    materialized views, 93-18, 93-20, 93-21  
REFRESH\_CATEGORY function, 89-28  
REFRESH\_MVIEW procedure, 41-10, 41-54  
REGISTER Procedure, 19-12  
REGISTER\_DEPENDENT\_OBJECT procedure, 110-13  
REGISTER\_FOREIGN\_QUEUE Procedure, 90-45  
REGISTERPARAMETER Procedure, 172-10  
REGISTERSHEMA procedure, 176-19  
REGISTERSTORE Procedure, 47-93  
REGISTERSTORECOMMAND Procedur, 49-21

---

REGISTERTOKENTABLESET Procedure, 40-14  
 REGISTERURI procedure, 176-23  
 REGISTERURLHANDLER procedure, 245-40  
 RELEASE function, 83-12  
 REMAP\_CONNECTION Procedure, 160-28  
 REMAP\_STGTAB\_SQLPROF Procedure, 139-51  
 REMAP\_STGTAB\_SQLSET Procedure, 139-52  
 REMOVE Procedure, 169-48  
 REMOVE procedure  
     of DBMS\_ALERT package, 19-13  
     of DBMS\_JOB package, 78-12  
     of OWA\_COOKIE package, 187-9  
 REMOVE\_AGENT Procedure, 90-46  
 REMOVE\_CATEGORY procedure, 89-29  
 REMOVE\_COLORED\_SQL Procedure, 161-41  
 REMOVE\_FILE procedure, 67-23  
 REMOVE\_JOB Procedure, 90-47  
 REMOVE\_MSGSYSTEM\_LINK Procedure, 90-48  
 REMOVE\_OPTION Procedure, 90-49  
 REMOVE\_PAIR member procedure, 256-14  
 REMOVE\_PIPE function, 103-28  
 REMOVE\_PROXY procedure, 89-30  
 REMOVE\_QUEUE procedure, 144-149  
 REMOVE\_RULE procedure, 127-25, 144-150  
 REMOVE\_SCHEME procedure, 89-31  
 REMOVE\_SQLSET\_REFERENCE procedure, 139-53  
 REMOVE\_STMT\_FROM\_HANDLER procedure, 147-14  
 REMOVE\_STMT\_HANDLER procedure, 21-36  
 REMOVE\_STREAMS\_CONFIGURATION procedure, 144-152  
 REMOVE\_SUBSCRIBER Procedure, 90-51  
 REMOVEALL procedure, 19-14  
 REMOVEATTRIBUTE procedure, 170-135  
 REMOVEATTRIBUTENODE function, 170-136  
 REMOVENAMEDITEM function, 170-138  
 REMOVEPARAM procedure, 180-14  
 REMOVEXSLTPARAM procedure, 174-16, 175-15  
 RENAME\_BASELINE Procedure, 161-42  
 RENAME\_COLUMN member procedure, 248-29  
 RENAME\_COLUMN procedure, 144-154  
 RENAME\_SCHEMA procedure, 144-157  
 RENAME\_TABLE procedure, 144-159  
 RENAMEPATH Procedure, 48-32  
 RENAMEPATH Procedures, 47-94  
 RENAMERESOURCE procedure, 163-65  
 REPLACECHILD function, 170-139  
 REPLACEDATA procedure, 170-140  
 replication  
     Oracle Streams  
         configuring, 144-16, 144-97, 144-101, 144-111, 144-114, 144-126, 144-136, 144-141  
         split and merge, 144-130, 144-133, 144-175  
 REPLY, REPLIES record types, 235-6  
 REPORT Function, 159-13, 160-29  
 REPORT\_ANALYSIS\_TASK Function, 138-15  
 REPORT\_AUTO\_TUNING\_TASK Function, 139-54  
 REPORT\_DIAGNOSIS\_TASK Function, 137-35  
 REPORT\_SQL\_DETAIL Function, 139-56  
 REPORT\_SQL\_MONITOR Function, 139-59  
 REPORT\_SQL\_MONITOR\_LIST Function, 139-64  
 REPORT\_TUNING\_TASK function, 28-7, 139-66  
 REQUEST function, 83-13, 224-73

---

REQUEST\_PIECES function, 224-75  
RESET\_ANALYSIS\_TASK Procedure, 138-17  
RESET\_BUFFER procedure, 103-27  
RESET\_DIAGNOSIS\_TASK Procedure, 137-36  
RESET\_GLOBAL\_PREFS\_DEFAULTS Procedure, 141-131  
RESET\_JOB Procedure, 90-52  
RESET\_PACKAGE Procedure, 131-25  
RESET\_PARAM\_DEFAULTS Procedure, 141-132  
RESET\_SESSION procedure, 123-43  
RESET\_SQLWKLD Procedure, 18-45  
RESET\_SUBSCRIBER Procedure, 90-53  
RESET\_TASK Procedure, 18-46  
RESET\_TUNING\_TASK procedure, 139-68  
RESETPARAMS procedure, 180-15  
RESOLVENAMESPACEPREFIX function, 170-141  
RESTARTQUERY procedure, 171-12  
RESTORE function, 142-14  
RESTORE\_DATABASE\_STATS procedure, 141-133  
RESTORE\_DEFAULTS Procedure, 37-8  
RESTORE\_DICTIONARY\_STATS procedure, 141-134  
RESTORE\_FIXED\_OBJECTS\_STATS procedure, 141-135  
RESTORE\_SCHEMA\_STATS procedure, 141-136  
RESTORE\_SYSTEM\_STATS procedure, 141-137  
RESTORE\_TABLE\_STATS procedure, 141-138  
RESTOREALL Procedure, 47-95, 48-33  
RESTOREPATH Procedure, 47-96, 48-34  
RESUME\_ANALYSIS\_TASK Procedure, 138-18  
RESUME\_DIAGNOSIS\_TASK Procedure, 137-37  
RESUME\_PROFILER function and procedure, 106-14  
RESUME\_REPLAY Procedure, 160-30  
RESUME\_TASK Procedure, 101-26  
RESUME\_TUNING\_TASK Procedure, 139-69  
REUSE\_REPLAY\_FILTER\_SET Procedure, 160-31  
REVERSE function, 232-32  
REVOKE\_ADMIN\_PRIVILEGE procedure, 146-10  
REVOKE\_OBJECT\_PRIVILEGE procedure, 67-24, 127-27  
REVOKE\_PRIVILEGE procedure, 65-28, 123-44  
REVOKE\_REMOTE\_ADMIN\_ACCESS procedure, 146-12  
REVOKE\_SWITCH\_CONSUMER\_GROUP procedure, 120-5  
REVOKE\_SYSTEM\_PRIVILEGE procedure, 67-25, 120-6, 127-28  
RLM\$EVENTIDS object type, 257-5  
ROLLBACK procedure, 152-17  
ROLLBACK\_FORCE procedure, 152-18  
ROLLBACK\_SAVEPOINT procedure, 152-19  
row migration, 144-55, 144-59  
ROWID datatype  
    extended format, 125-17  
ROWID\_BLOCK\_NUMBER function, 125-10  
ROWID\_CREATE function, 125-11  
ROWID\_INFO procedure, 125-12  
ROWID\_OBJECT function, 125-13  
ROWID\_RELATIVE\_FNO function, 125-14  
ROWID\_ROW\_NUMBER function, 125-15  
ROWID\_TO\_ABSOLUTE\_FNO function, 125-16  
ROWID\_TO\_EXTENDED function, 125-17  
ROWID\_TO\_RESTRICTED function, 125-19  
ROWID\_TYPE function, 125-20  
ROWID\_VERIFY function, 125-21  
rule sets  
    adding rules to, 127-6

---

- creating, 127-17
- dropping, 127-20
- removing rules from, 127-25
- rule-based transformations
  - setting, 144-167
- rules
  - action contexts
    - adding name-value pairs, 256-13
    - getting name-value pairs, 256-13
    - getting value for name, 256-14
    - removing name-value pairs, 256-14
    - transformations, 144-167
  - altering, 127-11
  - creating, 127-15
  - DBMS\_RULE package, 126-1
  - DBMS\_RULE\_ADM package, 127-1
  - dropping, 127-19
  - evaluation, 126-7
    - iterators, 126-6, 126-11
  - evaluation contexts
    - altering, 127-8
    - creating, 127-13
    - dropping, 127-18
  - inbound servers, 144-15
  - object privileges
    - granting, 127-21
    - revoking, 127-27
  - outbound servers, 144-14
  - propagations
    - removing, 144-150
  - RE\$ATTRIBUTE\_VALUE type, 256-7
  - RE\$ATTRIBUTE\_VALUE\_LIST type, 256-8
  - RE\$COLUMN\_VALUE type, 256-9, 256-15
  - RE\$COLUMN\_VALUE\_LIST type, 256-10
  - RE\$NAME\_ARRAY type, 256-11
  - RE\$NV\_ARRAY type, 256-12
  - RE\$NV\_LIST type, 256-13
  - RE\$RULE\_HIT type, 256-16
  - RE\$RULE\_HIT\_LIST type, 256-17
  - RE\$TABLE\_ALIAS type, 256-18
  - RE\$TABLE\_ALIAS\_LIST type, 256-19
  - RE\$TABLE\_VALUE type, 256-20
  - RE\$TABLE\_VALUE\_LIST type, 256-21
  - RE\$VARIABLE\_TYPE type, 256-22
  - RE\$VARIABLE\_TYPE\_LIST type, 256-24
  - RE\$VARIABLE\_VALUE type, 256-25
  - RE\$VARIABLE\_VALUE\_LIST type, 256-26
  - subset
    - defining, 144-55, 144-59
  - system privileges
    - granting, 127-23
    - revoking, 127-28
  - system-created, 144-9
    - global apply, 144-34
    - global capture, 144-34
    - global propagation, 144-30
    - global schema, 144-50
    - message, 144-43
    - message propagation, 144-39
    - removing, 144-150

---

- schema capture, 144-50
- schema propagation, 144-46
- subset apply, 144-59
- subset capture, 144-59
- subset propagation, 144-55
- table apply, 144-70
- table capture, 144-70
- table propagation, 144-65
- types, 256-1

Rules Manager object types, 257-1

RUN procedure, 78-13

RUN\_CHECK Procedure, 72-6

RUN\_TASK Procedure, 101-28

RUNTIME\_INFO Record Type, 52-18

## S

S function

- of HTF package, 182-114

S procedure

- of HTP package, 183-113

SAM (SQL Aggregation Management), 41-4

SAMPLE function

- of HTF package, 182-115

SAMPLE procedure

- of HTP package, 183-114

SAVE function, 142-15

SAVE Procedure, 166-50

SAVEPOINT procedure, 152-20

SCHEDULE\_PROPAGATION Procedure, 23-48, 90-54

Scheduler Chain Condition Syntax, 128-78

SCHEDULER\$\_STEP\_TYPE object type, 128-12

SCHEDULER\$\_STEP\_TYPE\_LIST table type, 128-12

SCHEMA\_NAME Function, 26-9

SCHEMAVALIDATE procedure, 259-21

SCRIPT function

- of HTF package, 182-116

SCRIPT procedure

- of HTP package, 183-115

SCRIPT\_TUNING\_TASK Function, 139-70

SDO\_CS package documentation, 195-2

SDO\_CSW\_PROCESS package documentation, 196-2

SDO\_GCDR package documentation, 197-2

SDO\_GEOM package documentation, 198-2

SDO\_GEOR package documentation, 199-2

SDO\_GEOR\_ADMIN package documentation, 200-2

SDO\_GEOR\_UTL package documentation, 201-2

SDO\_LRS package documentation, 202-2

SDO\_MIGRATE package documentation, 203-2

SDO\_NET package documentation, 204-2

SDO\_NET\_MEM package documentation, 205-2

SDO\_OLS package documentation, 206-2

SDO\_PC\_PKG package documentation, 207-2

SDO\_SAM package documentation, 208-2

SDO\_TIN\_PKG package documentation, 209-2

SDO\_TOPO package documentation, 210-2

SDO\_TOPO\_MAP package documentation, 211-2

SDO\_TUNE package documentation, 212-2

SDO\_UTIL package documentation, 213-2

SDO\_WFS\_LOCK package documentation, 214-2

SDO\_WFS\_PROCESS package documentation, 215-2



---

SEARCH Function, 47-97, 48-35  
SECURE\_CONNECTION Procedure, 237-30  
SEED procedures, 108-10  
SEED\_COL\_USAGE Procedures, 141-139  
SEGMENT\_CORRUPT procedure, 134-13  
SEGMENT\_DROP\_CORRUPT procedure, 134-14  
SEGMENT\_DUMP procedure, 134-15  
SEGMENT\_FIX\_STATUS procedure, 112-17  
SEGMENT\_VERIFY procedure, 134-16  
SELECT\_BASELINE\_METRICS Function, 161-43  
SELECT\_CURSOR\_CACHE Function, 139-72  
SELECT\_OBJECT procedure, 234-11  
SELECT\_SQL\_TRACE Function, 139-76  
SELECT\_SQLSET function, 139-78, 139-80  
SELECT\_WORKLOAD\_REPOSITORY functions, 139-82  
SELECTNODES function, 180-16  
SELECTSINGLENODE function, 180-17  
SELF\_CHECK procedure, 52-49  
SEM\_APIS package documentation, 216-2  
SEM\_PERF package documentation, 217-2  
SEM\_RDFS package documentation, 218-2, 219-2  
SEND procedure, 187-10, 229-7  
SEND\_ATTACH\_RAW procedure, 229-8  
SEND\_ATTACH\_VARCHAR2 procedure, 229-9  
SEND\_MESSAGE function, 103-29  
SENDCOMMAND Procedures, 49-22  
SERV\_MOD\_ACT\_STAT\_DISABLE procedure, 92-9  
SERV\_MOD\_ACT\_STAT\_ENABLE procedure, 92-10  
SERV\_MOD\_ACT\_TRACE\_DISABLE procedure, 92-12  
SERV\_MOD\_ACT\_TRACE\_ENABLE procedure, 92-13  
SESSION\_TRACE\_DISABLE Procedure, 131-23  
SESSION\_TRACE\_ENABLE Procedure, 131-24  
SESSION\_TRACE\_DISABLE procedure, 92-15  
SESSION\_TRACE\_ENABLE procedure, 92-16  
SET\* member procedures  
    of ANYDATA TYPE, 240-15  
    of ANYDATASET TYPE, 241-15  
SET\_ACTION procedure, 20-9  
SET\_ADVANCED\_PARAMETER Procedure, 160-32  
SET\_ANALYSIS\_DEFAULT\_PARAMETER Procedures, 138-21  
SET\_ANALYSIS\_TASK\_PARAMETER Procedure, 138-19  
SET\_AUDIT\_TRAIL\_LOCATION procedure, 27-25  
SET\_AUDIT\_TRAIL\_PROPERTY procedure, 27-27  
SET\_AUTHENTICATION procedure, 224-78  
SET\_AUTHENTICATION\_FROM\_WALLET Procedure, 224-79  
SET\_AUTHORIZATION procedure, 192-9  
SET\_AUTO\_TUNING\_TASK\_PARAMETER Procedures, 28-9  
SET\_BASE\_TABLE\_NAME member procedure, 248-11  
SET\_BASE\_TABLE\_OWNER member procedure, 248-11  
SET\_BODY\_CHARSET procedures, 224-81  
SET\_BREAKPOINT function, 52-50  
SET\_CHANGE\_HANDLER procedure, 21-38  
SET\_CHUNK\_STATUS Procedure, 101-30  
SET\_CLIENT\_INFO procedure, 20-10  
SET\_CLIENT\_SERVICE Procedure, 29-10  
SET\_CNS\_EXCEPTION\_LOG procedure, 42-8  
SET\_COLUMN\_STATS procedures, 141-140  
SET\_COMMAND\_TYPE member procedure, 248-43  
SET\_CONSUMER\_GROUP\_MAPPING procedure, 119-31  
SET\_CONSUMER\_GROUP\_MAPPING\_PRI procedure, 119-32

---

SET\_CONTEXT Procedure, 131-27  
SET\_COOKIE\_SUPPORT procedures, 224-83  
SET\_CURRENT\_SCHEMA member procedure, 248-12  
SET\_DAD\_ATTRIBUTE Procedure, 63-24  
SET\_DATABASE\_PREFS Procedure, 141-142  
SET\_DBFS\_LINK Procedures, 82-74  
SET\_DDL\_TEXT member procedure, 248-12  
SET\_DEFAULT\_SQLWKLD\_PARAMETER Procedure, 18-47  
SET\_DEFAULT\_TASK\_PARAMETER Procedures, 18-48  
SET\_DEFAULTS procedure, 19-15  
SET\_DETAILED\_EXCP\_SUPPORT procedure, 224-85  
SET\_DIAGNOSIS\_TASK\_PARAMETER Procedure, 137-38  
SET\_DML\_HANDLER procedure, 21-41  
SET\_EDITION\_DEFERRED Procedure, 131-29  
SET\_EDITIONING\_VIEWS\_READ\_ONLY Procedure, 62-7  
SET\_ENQUEUE\_DESTINATION procedure, 21-46  
SET\_EXECUTE procedure, 21-48  
SET\_EXPRESSION procedure, 45-76  
SET\_EXTRA\_ATTRIBUTE member procedure, 248-43  
SET\_FOLLOW\_REDIRECT procedures, 224-86  
SET\_GLOBAL\_ATTRIBUTE Procedure, 63-26  
SET\_GLOBAL\_INSTANTIATION procedure, 21-50  
SET\_GLOBAL\_PREFS Procedure, 141-145  
SET\_HEADER procedure, 224-87  
SET\_IDENTIFIER, 131-30  
SET\_INDEX\_STATS procedures, 141-148  
SET\_INITIAL\_CONSUMER\_GROUP procedure, 119-33  
SET\_JAVA\_LOGGING\_LEVEL procedure, 89-32  
SET\_KEY\_COLUMNS procedure, 21-53  
SET\_LAST\_ARCHIVE\_TIMESTAMP procedure, 27-30  
SET\_LOB\_INFORMATION member procedure, 248-29  
SET\_LOB\_OFFSET member procedure, 248-30  
SET\_LOB\_OPERATION\_SIZE member procedure, 248-31  
SET\_LOG\_LEVEL Procedure, 90-56  
SET\_LOG\_SPEC procedure, 43-29  
SET\_LOGON\_USER member procedure, 248-13  
SET\_MAILHOST Procedure, 24-3  
SET\_MAILPORT Procedure, 24-4  
SET\_MESSAGE\_NOTIFICATION procedure, 144-161  
SET\_MESSAGE\_TRACKING procedure, 144-167  
SET\_MODULE procedure, 20-11  
SET-NLS Procedure, 131-31  
SET\_OBJECT\_NAME member procedure, 248-44  
SET\_OBJECT\_OWNER member procedure, 248-45  
SET\_OBJECT\_TYPE member procedure, 248-13  
SET\_OER\_BREAKPOINT function, 52-51  
SET\_OPTION Procedure, 90-57  
SET\_P1\_RESOURCES Procedure, 29-11  
SET\_PARAM procedure, 141-151  
SET\_PARAMETER procedure, 32-39, 43-30  
    apply process, 21-56  
SET\_PERSISTENT\_CONN\_SUPPORT procedure, 224-88  
SET\_PLSQL\_LOGGING\_LEVEL procedure, 89-33  
SET\_PLSQL\_TRACE procedure, 151-14  
SET\_PROTECTION\_REALM procedure, 192-10  
SET\_PROXY procedure, 89-34, 224-90  
SET\_PURGE\_JOB\_INTERVAL procedure, 27-32  
SET\_PURGE\_JOB\_STATUS procedure, 27-33  
SET\_REPLAY\_TIMEOUT Procedure, 160-33  
SET\_RESPONSE\_ERROR\_CHECK procedure, 224-91

---

SET\_ROLE Procedure, 131-32  
SET\_ROWID\_THRESHOLD Procedure, 38-32  
SET\_RULE\_TRANSFORM\_FUNCTION procedure, 144-167  
SET\_SCHEMA\_INSTANTIATION procedure, 21-66  
SET\_SCHEMA\_PREFS Procedure, 141-153  
SET\_SENDFROM Procedure, 24-5  
SET\_SESSION\_LONGOPS procedure, 20-12  
SET\_SESSION\_TIMEOUT procedure, 122-8  
SET\_SOURCE\_DATABASE\_NAME member procedure, 248-45  
SET\_SQL\_TRACE Procedure, 131-33  
SET\_SQLWKLD\_PARAMETER Procedure, 18-49  
SET\_SYSTEM\_STATS procedure, 141-154  
SET\_TABLE\_INSTANTIATION procedure, 21-69  
SET\_TABLE\_PREFS Procedure, 141-156  
SET\_TABLE\_STATS procedure, 141-159  
SET\_TAG member procedure, 248-45  
SET\_TAG procedure, 143-20, 144-171  
SET\_TASK\_PARAMETER Procedure, 18-50  
SET\_THRESHOLD procedure, 129-15  
SET\_TIMEOUT function, 52-52  
SET\_TIMEOUT procedure, 122-9  
SET\_TIMEOUT\_BEHAVIOUR procedure, 52-53  
SET\_TRANSFER\_TIMEOUT procedure, 224-92  
SET\_TRANSFORM procedure, 45-78  
SET\_TRIGGER\_FIRING\_PROPERTY procedure, 53-13  
SET\_TUNING\_TASK\_PARAMETER Function, 139-84  
SET\_UP\_QUEUE procedure, 144-172  
SET\_UPDATE\_CONFLICT\_HANDLER procedure, 21-71  
SET\_VALUE function, 52-54  
SET\_VALUE member procedure, 248-31  
SET\_VALUE\_DEPENDENCY procedure, 21-75  
SET\_VALUES member procedure, 248-33  
SET\_WALLET procedure, 224-93  
SET\_WARNING\_SETTING\_STRING procedure, 157-11  
SET\_WATERMARK Procedure, 23-50  
SET\_XML\_INFORMATION member procedure, 248-35  
SETACL Procedure, 166-51  
SETACL procedure, 163-66  
SETATTRIBUTE procedure, 170-142  
SETATTRIBUTENODE function, 170-143  
SETAUTHOR Procedure, 166-52  
SETBASEDIR procedure, 173-17  
SETBATCHSIZE procedure, 175-16  
SETBINDVALUE procedure, 174-17  
SETCHARACTERSET Procedure, 166-53  
SETCHARSET Procedure, 170-144  
SETCOLLIDATRNAME procedure, 174-18  
SETCOMMENT Procedure, 166-54  
SETCOMMITBATCH procedure, 175-17  
SETCONTENT Procedures, 163-67, 166-55  
SETCONTENTTYPE Procedure, 82-75, 166-56  
SETCONVERTSPECIALCHARS procedure, 171-13  
SETCUSTOMMETADATA Procedure, 166-57  
SETDATA procedure, 170-145  
SETDATAHEADER procedure, 174-19  
SETDATEFORMAT procedure, 174-20, 175-18  
SETDCHARSET procedure, 170-152  
SETDEFAULTACL Procedure, 47-98  
SETDEFAULTASOF Procedur, 47-99  
SETDEFAULTCONTEXT Procedure, 47-100

---

SETDEFAULTOWNER Procedure, 47-101  
SETDEFAULTPRINCIPAL Procedure, 47-102  
SETDISPLAYNAME Procedure, 166-58  
SETDOCTYPE Procedure, 170-146  
SETDOCTYPE procedure, 173-18  
SETDVERSION procedure, 170-154  
SETENCODINGTAG procedure, 174-21  
SETERRORLOG procedure, 173-19, 180-18  
SETERRORTAG procedure, 174-22  
SETFTPSPORT Procedure, 163-69  
SETHTTPSPORT Procedure, 163-70  
SETIGNORECASE procedure, 175-19  
SETINFO member procedure  
    of ANYTYPE TYPE, 242-4  
SETKEYCOLUMN procedure, 175-20, 177-12  
SETLANGUAGE Procedure, 166-59  
SETLISTENERENDPOINT Procedure, 163-71  
SETLISTENERLOCALACCESS Procedure, 163-72  
SETMAXROWS procedure, 171-14, 174-23  
SETMETAHEADER procedure, 174-24  
SETNAMEDITEM function, 170-147  
SETNODEVALUE procedure, 170-148  
SETNODEVALUEASBINARYSTREAM Function & Procedure, 170-149  
SETNODEVALUEASCHARACTERSTREAM Function & Procedure, 170-150  
SETOPTIONS Procedure, 82-76  
SETOWNER Procedure, 166-60  
SETPARAM procedure, 180-19  
SETPATH Procedure, 48-36  
SETPATH Procedures, 47-103  
SETPREFIX procedure, 170-151  
SETPRESERVEWHITESPACE procedure, 173-20, 175-21  
SETRAISEEXCEPTION procedure, 174-25  
SETRAISENOROWSEXCEPTION procedure, 174-26  
SETRENDERPATH Procedure, 169-49  
SETRENDERSTREAM Procedure, 169-50  
SETROWIDATTRNAME procedure, 174-27  
SETROWIDATTRVALUE procedure, 174-28  
SETROWSETTAG procedure, 171-16, 174-29  
SETROWTAG procedure, 174-30, 175-22, 177-13  
SETSCHEMAVALIDATED procedure, 259-22  
SETSKIPROWS procedure, 171-18, 174-31  
SETSOURCELANG Function, 178-14  
SETSQLTOXMLNAMEESCAPING procedure, 174-32, 175-23  
SETSTANDALONE procedure, 170-152  
SETSTATS Procedure, 47-104  
SETSTOREPROPERTY Procedure, 49-23  
SETSTYLESHEETHEADER procedure, 174-33  
SETTAGCASE procedure, 174-34  
SETTRACE Procedure, 47-105  
SETUPDATECOLUMN procedure, 175-24, 177-14  
SETVALIDATIONMODE procedure, 173-21  
SETVALUE procedure, 170-153  
SETXSLT procedure, 174-35, 175-25  
SETXSLTPARAM procedure, 174-36, 175-26  
SHOW\_BREAKPOINTS procedures, 52-56  
SHOW\_EXTENDED\_STATS\_NAME Function, 141-161  
SHOW\_FRAME\_SOURCE procedure, 52-57  
SHOW\_SOURCE procedures, 52-58  
SHOW\_STATS procedure, 236-20  
SHOWPAGE procedure, 194-19

---

SHOWSOURCE procedure, 194-20  
SHOWWARNINGS procedure, 173-22, 180-20  
SHUTDOWN Procedure, 90-59  
SIGNAL procedure, 19-16  
SIGNATURE procedure, 194-21  
SIMPLE\_SQL\_NAME Function, 26-10  
SIZES procedure, 132-12  
SKIP\_CORRUPT\_BLOCKS procedure, 112-18  
SLEEP procedure, 83-14  
SMALL function  
    of HTF package, 182-117  
SMALL procedure  
    of HTP package, 183-116  
snapshot. See DBMS\_MVIEW, 93-1  
SOURCE\_LINES\_T Table Type, 105-6  
SPACE\_ERROR\_INFO function, 122-10  
SPACE\_USAGE procedure, 133-20  
SPACEUSAGE Procedure, 47-106, 48-37  
split streams, 144-175  
SPLIT\_STREAMS procedure, 144-175  
SPLITPATH Procedure, 163-73  
SPLITTEXT function, 170-155  
SQL Apply  
    managing logical standby databases, 86-3  
    managing with DBMS\_LOGSTDBY package, 86-3  
SQL generation, 248-24, 248-26  
SQL Performance Reporting Subprograms, 139-15  
SQL\*Plus  
    creating a sequence, 1-7  
SQL\_OBJECT\_NAME Function, 26-11  
SQLID\_TO\_SQLHASH Function, 156-56  
SQLSET\_ROW Object Type, 139-8  
SQLTEXT\_TO\_SIGNATURE Function, 139-86  
STACK\_BIN\_CAT procedure, 45-80  
STACK\_BIN\_NUM procedure, 45-82  
STACK\_CLIP procedure, 45-85  
STACK\_COL\_REM procedure, 45-87  
STACK\_MISS\_CAT procedure, 45-89  
STACK\_MISS\_NUM procedure, 45-91  
STACK\_NORM\_LIN procedure, 45-93  
staging  
    queues  
        creating, 144-172  
        removing, 144-149  
START\_APPLY procedure, 21-77  
START\_CAPTURE Procedure, 159-14  
START\_CAPTURE procedure, 32-49  
START\_POOL Procedure, 37-6  
START\_PROFILER functions and procedures, 106-15  
START\_PROFILING Procedure, 73-5  
START\_PROPAGATION procedure, 107-13  
START\_REDEF\_TABLE procedure, 110-14  
START\_REPLAY Procedure, 160-34  
START\_SERVICE procedure, 130-16  
STARTTLS Function and Procedure, 235-32  
STARTUP Procedure, 90-60  
STARTUP\_EXTPROC\_AGENT procedure, 181-7  
STATUS Function, 121-11  
STATUS\_LINE procedure, 194-22  
STEP\_ID function, 152-21

---

STOP\_APPLY procedure, 21-78  
STOP\_CAPTURE procedure, 32-50  
STOP\_POOL Procedure, 37-7  
STOP\_PROFILER function and procedure, 106-16  
STOP\_PROFILING Procedure, 73-6  
STOP\_PROPAGATION procedure, 107-14  
STOP\_SERVICE procedure, 130-17  
STOP\_TASK Procedure, 101-31  
STORE\_T Record Type, 47-30  
STORE\_VALUES procedure, 190-8  
stored outlines  
    DBMS\_OUTLN, 99-1  
    OUTLN\_PKG package, 99-1  
STOREPUSH Procedure, 49-26  
STREAM2MULTI procedure, 193-9  
Streams  
    removing configuration, 144-152  
STREAMS\$\_TRANSFORM\_FUNCTION, 144-169  
STRIKE function  
    of HTF package, 182-118  
STRIKE procedure  
    of HTP package, 183-117  
STRING function, 108-11  
STRING\_TO\_RAW Function, 225-31  
STRONG function  
    of HTF package, 182-119  
STRONG procedure  
    of HTP package, 183-118  
STYLE function  
    of HTF package, 182-120  
STYLE procedure  
    of HTP package, 183-119  
SUB procedure  
    of HTP package, 183-120  
SUBMIT procedure, 78-14  
SUBMIT\_PENDING\_AREA procedure, 119-34  
subscribers  
    drop the subscription, 34-3  
    extend the window to create a new view, 34-3  
    purging the subscription window, 34-3  
    retrieve change data from the subscriber views, 34-3  
subscription window  
    purging, 34-3  
SUBSTR function, 232-33  
SUBSTR Functions, 82-77  
SUBSTRINGDATA function, 170-156  
Summary of DBMS\_AQELM Subprograms, 24-2  
Summary of DBMS\_DIMENSION Subprograms, 59-4  
Summary of DBMS\_ERRLOG Subprograms, 64-4  
Summary of DBMS\_MVIEW Subprograms, 93-6  
Summary of DBMS\_XDBZ Subprograms, 168-5  
Summary of DBMS\_XMLDOM Subprograms, 170-30  
Summary of DBMS\_XMLSCHEMA Subprograms, 176-9  
Summary of UTL\_LMS Subprograms, 228-4  
SUMMARY procedure, 140-6  
SUP function  
    of HTF package, 182-122  
SUP procedure  
    of HTP package, 183-121  
SWITCH\_CONSUMER\_GROUP\_FOR\_SESS procedure, 119-35

---

SWITCH\_CONSUMER\_GROUP\_FOR\_USER procedure, 119-36  
SWITCH\_CURRENT\_CONSUMER\_GROUP Procedure, 131-34  
SWITCH\_PLAN procedure, 119-37  
SYNC\_INTERIM\_TABLE procedure, 110-15  
SYNC\_TEXT\_INDEXES procedure, 65-29, 123-45  
SYNCHRONIZE function, 52-60  
synchronous capture  
    altering, 32-16  
    instantiation  
        preparing a table for, 32-37  
    rules, 144-11  
SYNCINDEX Procedure, 172-11  
synonyms  
    comparing, 35-1  
SYS.MGW\_MQSERIES\_PROPERTIES Object Type, 90-8  
SYS.MGW\_PROPERTIES Object Type, 90-10  
SYS.MGW\_PROPERTY Object Type, 90-12  
SYS.MGW\_TIBRV\_PROPERTIES Object Type, 90-13

## T

table alias  
    attributes, 247-9  
TABLE\_CREATE procedure, 43-32  
TABLE\_TO\_COMMA procedures, 156-57  
TABLECAPTION function  
    of HTF package, 182-123  
TABLECAPTION procedure  
    of HTP package, 183-122  
TABLECLOSE function  
    of HTF package, 182-124  
TABLECLOSE procedure  
    of HTP package, 183-123  
TABLEDATA function  
    of HTF package, 182-125  
TABLEDATA procedure  
    of HTP package, 183-124  
TABLEHEADER function  
    of HTF package, 182-126  
TABLEHEADER procedure  
    of HTP package, 183-125  
TABLEOPEN function  
    of HTF package, 182-127  
TABLEOPEN procedure  
    of HTP package, 183-126  
TABLEPRINT function, 194-23  
TABLEROWCLOSE function  
    of HTF package, 182-128  
TABLEROWCLOSE procedure  
    of HTP package, 183-127  
TABLEROWOPEN function  
    of HTF package, 182-129  
TABLEROWOPEN procedure  
    of HTP package, 183-128  
tables  
    comparing, 35-1  
    table items as arrays, 136-53  
tablespace repositories  
    attaching tablespaces, 149-13  
    cloning tablespaces, 149-20  
    detaching tablespaces, 149-26

---

TABLESPACE\_FIX\_BITMAPS procedure, 134-17  
 TABLESPACE\_FIX\_SEGMENT\_STATES procedure, 134-18  
 TABLESPACE\_MIGRATE\_FROM\_LOCAL procedure, 134-19  
 TABLESPACE\_MIGRATE\_TO\_LOCAL procedure, 134-20  
 TABLESPACE\_REBUILD\_BITMAPS procedure, 134-21  
 TABLESPACE\_REBUILD\_QUOTAS procedure, 134-22  
 TABLESPACE\_RELOCATE\_BITMAPS procedure, 134-23  
 TABLESPACE\_SET type, 149-9  
 TABLESPACE\_VERIFY procedure, 134-24  
 tablespaces  
   change tables and, 33-26  
 tags  
   GET\_TAG function, 143-18, 144-86  
   SET\_TAG procedure, 143-20, 144-171  
 TARGET\_FILE function, 43-33  
 TARGET\_LOB function, 43-34  
 TARGET\_PROGRAM\_RUNNING procedure, 52-61  
 TARGET\_TABLE function, 43-35  
 TARGET\_TRACE function, 43-36  
 TELETYPE function  
   of HTF package, 182-130  
 TELETYPE procedure  
   of HTP package, 183-129  
 TERMINATE procedure, 108-12  
 TEXT\_DECODE function, 222-9  
 TEXT\_ENCODE function, 222-10  
 TIME\_TABLE Table Type, 136-41  
 TIME\_WITH\_TIME\_ZONE\_TABLE Table Type, 136-42  
 TIMESTAMP\_TABLE Table Type, 136-43  
 TIMESTAMP\_WITH\_LTZ\_TABLE Table Type, 136-44  
 TIMESTAMP\_WITH\_TIME\_ZONE\_TABLE Table Type, 136-45  
 TITLE function  
   of HTF package, 182-131  
 TITLE procedure  
   of HTP package, 183-130  
 TO\_CURSOR\_NUMBER Function, 136-87  
 TO\_REFCURSOR Function, 136-89  
 TO\_STRING function, 255-16  
 TODATE function, 194-26  
 TOOBJECT procedure, 259-23  
 TOUCHRESOURCE Procedure, 163-74  
 TRACE Procedure, 47-107  
 TRACE procedure, 42-9  
 TRACEENABLED Function, 47-108  
 TRACETAB.SQL, 151-7  
 tracking LCRs, 144-83, 144-167  
 TRANSACTION\_BACKOUT Procedures, 69-16  
 TRANSFORM function, 259-24  
 transformations  
   rule-based  
     adding a column, 144-27  
     custom, 144-167  
     deleting a column, 144-80  
     keeping columns, 144-87  
     renaming a column, 144-154  
     renaming a schema, 144-157  
     renaming a table, 144-159  
     STREAMS\$\_TRANSFORM\_FUNCTION, 144-169  
 TRANSFORMNODE function, 180-21  
 TRANSLATE function, 232-35, 255-17



---

TRANSLATEXML Function, 178-16  
TRANSLITERATE Function, 225-32, 232-37  
transparent data encryption  
    apply process, 21-61  
TRANSPORT\_SET\_CHECK procedure, 154-8  
TRIM Procedures, 82-79  
TUNE\_MVIEW Procedure, 18-63  
TYPE\_BUILD function, 43-37  
TYPE\_DIMENSION\_COMPILE function, 43-38  
TYPE\_OPERATIONS function, 43-39  
TYPE\_REJECTED\_RECORDS function, 43-40  
types  
    Expression Filter, 247-1  
    MGD\_ID, 255-1, 255-7  
    MGD\_ID\_COMPONENT, 255-5  
    MGD\_ID\_COMPONENT\_VARRAY, 255-6  
    Oracle Multimedia  
        ORDAudio, 249-1  
        ORDDicom, 250-1  
        ORDDoc, 251-1  
        ORDImage, 252-1  
        ORDVideo, 254-1  
        SQL/MM Still Image, 253-1  
    Rules Manager, 257-1

## U

ULISTCLOSE function  
    of HTF package, 182-132  
ULISTCLOSE procedure  
    of HTP package, 183-131  
ULISTOPEN function  
    of HTF package, 182-133  
ULISTOPEN procedure  
    of HTP package, 183-132  
UNASSIGN\_ACL Procedure, 94-16  
UNASSIGN\_ATTRIBUTE\_SET procedure, 65-30  
UNASSIGN\_WALLET\_ACL Procedure, 94-17  
UNCHECKOUT function, 165-16  
UNCL\_ARRAY Table Type, 156-14  
UNDERLINE function  
    of HTF package, 182-134  
UNDERLINE procedure  
    of HTP package, 183-133  
UNESCAPE function, 238-9  
UNESCAPEURI function, 245-39  
UNIFORM\_DIST\_FIT procedure, 140-7  
UNIQUE\_SESSION\_ID Function, 131-36  
UNIQUE\_SESSION\_NAME function, 103-31  
UNKEEP procedure, 132-13  
UNLOCK\_MAP procedure, 142-16  
UNLOCK\_PARTITION\_STATS Procedure, 141-162  
UNLOCK\_SCHEMA\_STATS procedure, 141-163  
UNLOCK\_TABLE\_STATS procedure, 141-164  
UNLOCKPATH Procedure, 47-109, 48-38  
UNLOCKRESOURCE function, 163-77  
UNMAP\_DAD Procedure, 63-27  
UNMARKHOT Procedure, 132-14  
UNMOUNTSTORE Procedure, 47-110  
UNPACK\_MESSAGE procedures, 103-32  
UNPACK\_STGTAB\_BASELINE Function, 135-23

---

- UNPACK\_STGTAB\_SQLPATCH Procedure, 137-39
- UNPACK\_STGTAB\_SQLPROF Procedure, 139-87
- UNPACK\_STGTAB\_SQLSET Procedure, 139-88
- UNREGISTER\_DEPENDENT\_OBJECT procedure, 110-16
- UNREGISTER\_FOREIGN\_QUEUE Procedure, 90-61
- UNREGISTERSTORE Procedure, 47-111
- UNREGISTERURLHANDLER procedure, 245-41
- UNSCHEDULE\_PROPAGATION Procedure, 90-62
- UNUSED\_SPACE procedure, 133-23
- UPDATE\_BY\_CAT procedure, 99-11
- UPDATE\_CATEGORY Procedure, 119-38
- UPDATE\_CONSUMER\_GROUP procedure, 119-39
- UPDATE\_OBJECT Procedure, 18-65
- UPDATE\_OBJECT procedure, 234-12
- UPDATE\_PLAN procedure, 119-40
- UPDATE\_PLAN\_DIRECTIVE procedure, 119-41
- UPDATE\_REC\_ATTRIBUTES Procedure, 18-67
- UPDATE\_SIGNATURES procedure, 99-12
- UPDATE\_SQLSET procedures, 139-90
- UPDATE\_SQLWKLD\_ATTRIBUTES Procedure, 18-69
- UPDATE\_SQLWKLD\_STATEMENT Procedure, 18-70
- UPDATE\_TASK\_ATTRIBUTES Procedure, 18-72
- UPDATERESOURCEMETADATA Procedures, 163-75
- UPDATETRANSLATION Function, 178-17
- UPDATEXML function, 175-27, 177-15
- UPGRADE\_DATABASE Procedure, 60-15
- UPGRADE\_SCHEMA Procedure, 60-17
- UPGRADE\_STAT\_TABLE procedure, 141-165
- UPGRADE\_TABLE Procedure, 60-19
- upgrading
  - post-upgrade actions, 96-1
- URI Types
  - description, 245-1
- UriFactory package, 245-36
  - ESCAPEURI function, 245-38
  - GETURL function, 245-37
  - methods, 245-36
  - REGISTERURLHANDLER procedure, 245-40
  - UNESCAPEURI function, 245-39
  - UNREGISTERURLHANDLER procedure, 245-41
- UriType supertype, 245-2
  - GETBLOB function, 245-3
  - GETCLOB function, 245-4
  - GETCONTENTTYPE function, 245-5
  - GETEXTERNALURL function, 245-6
  - GETURL function, 245-7
  - GETXML function, 245-8
  - methods, 245-2
- UROWID\_TABLE Table Type, 136-46
- USE\_FILTER\_SET Procedure, 160-35
- USE\_ROLLBACK\_SEGMENT procedure, 152-22
- USEBINARYSTREAM Function, 170-157
- USEITEMTAGSFORCOLL procedure, 171-19
- USENULLATTRIBUTEINDICATOR procedure, 171-20, 174-37
- USER\_EXPORT procedures, 78-16
- USETYPEFORCOLLELEMTAG procedure, 174-38
- Using DBMS\_ADVISOR, 18-2
- Using DBMS\_AQIN, 25-2
- Using DBMS\_FILE\_GROUP, 67-2
- Using DBMS\_MVIEW, 93-2

---

- Using DBMS\_RULE, 126-2
- Using DBMS\_RULE\_ADM, 127-2
- Using DBMS\_STREAMS, 143-2
- Using DBMS\_STREAMS\_ADM, 144-2
- Using DBMS\_STREAMS\_TABLESPACE\_ADM, 149-2
- Using DBMS\_XMLDOM, 170-3
- Using DBMS\_XMLSCHEMA, 176-2
- Using UTL\_HTTP, 224-2
- UTL Streams Types, 258-1
- UTL\_BINARYINPUTSTREAM Type, 258-3
- UTL\_BINARYOUTPUTSTREAM Type, 258-4
- UTL\_CHARACTERINPUTSTREAM Type, 258-5
- UTL\_CHARACTEROUTPUTSTREAM Type, 258-6
- UTL\_COLL package, 220-1
- UTL\_COMPRESS package, 221-1
- UTL\_ENCODE package, 222-1
- UTL\_FILE package, 223-1
- UTL\_HTTP package, 224-1
- UTL\_I18N package, 225-1
  - ESCAPE\_REFERENCE function, 225-9, 225-14, 225-16, 225-17, 225-19, 225-22, 225-26
  - GET\_DEFAULT\_CHARSET function, 225-11
  - MAP\_CHARSET function, 225-20
  - MAP\_LANGUAGE\_FROM\_ISO function, 225-23
  - MAP\_LOCALE\_TO\_ISO function, 225-24
  - MAP\_TERRITORY\_FROM\_ISO function, 225-25
  - RAW\_TO\_CHAR function, 225-27
  - RAW\_TO\_NCHAR function, 225-29
  - UNESCAPE\_REFERENCE function, 225-34
- UTL\_INADDR package, 226-1
- UTL\_LMS package, 228-1
  - FORMAT\_MESSAGE function, 228-5
  - GET\_MESSAGE function, 228-6
- UTL\_MAIL package, 229-1
- UTL\_MATCH package, 230-1
- UTL\_NLA package, 231-1
- UTL\_RAW package, 232-1
- UTL\_RECOMP package, 233-1
- UTL\_REF package, 234-1
- UTL\_SPADV package, 236-1
- UTL\_TCP package, 237-1
- UTL\_URL package, 238-1
- UUDECODE function, 222-11
- UUENCODE function, 222-12

## V

- V\$STREAMS\_MESSAGE\_TRACKING view, 144-83, 144-167
- v\$vpd\_policies, 124-5
- VALIDATE procedure, 156-58
- VALIDATE\_DIMENSION procedure, 59-6
- VALIDATE\_EXPRESSIONS procedure, 65-31
- VALIDATE\_PENDING\_AREA procedure, 119-45
- VALIDATE\_REWRITE\_EQUIVALENCE Procedure, 17-10
- VALIDATE\_SCHEME function, 89-35
- VALIDATE\_XML procedure, 41-58
- VALUE functions, 108-13
- VALUEOF procedure, 180-22
- VARCHAR2\_TABLE Table Type, 136-47
- VARCHAR2A Table Type, 136-48
- VARCHAR2S Table Type, 136-49
- VARIABLE function

---

- of HTF package, 182-135
- VARIABLE procedure
  - of HTP package, 183-134
- VARIABLE\_VALUE procedures, 136-91
- VERBOSE\_ACTION function, 43-27
- VERBOSE\_DEBUG function, 43-25
- VERBOSE\_INFO function, 43-28
- VERBOSE\_NOTICE function, 43-26
- VERBOSE\_STATS function, 43-24
- VERIFY\_VALUES function, 190-9
- VERSION function, 43-41
- views
  - comparing, 35-1
- virtual dependency definitions
  - object dependencies
    - creating, 21-24
    - dropping, 21-29
  - value dependencies, 21-75
- Virtual Private Database. See VPD
- VPD
  - column masking, 124-10
  - enabling column-level, 124-10
  - viewing current cursors and policy predicates, 124-5
- VPD use of DBMS\_RLS, 124-1
- VPFY function, 235-33

## W

- WAIT\_ON\_PENDING\_DML Procedure, 156-60
- WAITANY procedure, 19-17
- WAITONE procedure, 19-18
- WBR function
  - of HTF package, 182-136
- WBR procedure
  - of HTP package, 183-135
- WEIBULL\_DIST\_FIT procedure, 140-8
- WHAT procedure, 78-17
- WHO\_CALLED\_ME procedure, 194-27
- WPG\_DOCLOAD package, 239-1
- WRAP Functions, 53-15
- WRITE Procedures, 82-81
- WRITE\_DATA procedure, 235-34
- WRITE\_LINE function, 237-31
- WRITE\_LINE procedure, 224-95
- WRITE\_RAW function, 237-32
- WRITE\_RAW procedure, 224-96
- WRITE\_RAW\_DATA procedure, 235-35
- WRITE\_TEXT function, 237-33
- WRITE\_TEXT procedure, 224-97
- WRITEAPPEND Procedures, 82-83
- WRITETOBUFFER procedure, 170-158
- WRITETOCLOB procedure, 170-159
- WRITETOFILE procedure, 170-160

## X

- XA\_COMMIT Function, 162-13
- XA\_END Function, 162-14
- XA\_FORGET Function, 162-15
- XA\_GETLASTOER Function, 162-16
- XA\_PREPARE Function, 162-17
- XA\_RECOVER Function, 162-18

---

XA\_ROLLBACK Function, 162-19  
XA\_SETTIMEOUT Function, 162-20  
XA\_START Function, 162-21  
XDBEvent Type Subprograms, 169-6  
XDBHandler Type Subprograms, 169-9  
XDBHandlerList Type Subprograms, 169-8  
XDBLink Type Subprograms, 169-11  
XDBPath Type Subprograms, 169-10  
XDBRepositoryEvent Type Subprograms, 169-7  
XDBUriType, 245-27  
XDBURITYPE function, 245-35  
XDBUriType subtype, 245-27  
    CREATEURI function, 245-28  
    GETBLOB function, 245-29  
    GETCLOB function, 245-30  
    GETCONTENTTYPE function, 245-31  
    GETEXTERNALURL function, 245-32  
    GETURL function, 245-33  
    GETXML function, 245-34  
    methods, 245-27  
    XDBURITYPE function, 245-35  
XDBZ Constants, 168-4  
XFORM\_BIN\_CAT procedure, 45-95  
XFORM\_BIN\_NUM procedure, 45-97  
XFORM\_CLIP procedure, 45-100  
XFORM\_COL\_REM procedure, 45-102  
XFORM\_EXPR\_NUM procedure, 45-104  
XFORM\_EXPR\_STR procedure, 45-106  
XFORM\_MISS\_CAT procedure, 45-109  
XFORM\_MISS\_NUM procedure, 45-112  
XFORM\_NORM\_LIN procedure, 45-114  
XFORM\_STACK procedure, 45-117  
XML tags  
    specifying list of  
        configuring index parameters, 247-13  
XML\_TO\_LCR Function, 91-37  
XMLINDEXADDPATH Procedure, 164-10  
XMLINDEXREMOVEPATH Procedure, 164-11  
XMLType  
    CREATENONSCHEMABASEDXML function, 259-4  
    CREATESCHEMABASEDXML function, 259-5  
    CREATEXML function, 259-6  
    description, 259-1  
    EXISTSNODE function, 259-8  
    EXTRACT function, 259-9  
    GETBLOBVAL function, 259-10  
    GETCLOBVAL function, 259-11  
    GETNUMBERVAL function, 259-13  
    GETROOTELEMENT function, 259-14  
    GETSCHEMAURL function, 259-15  
    GETSTRINGVAL function, 259-16  
    ISFRAGMENT function, 259-17  
    ISSCHEMABASED function, 259-18  
    ISSCHEMAVALID function, 259-19  
    ISSCHEMAVALIDATED function, 259-20  
    NAMESPACE function, 259-12  
    SCHEMAVALIDATE procedure, 259-21  
    SETSCHEMAVALIDATED procedure, 259-22  
    TOOBJECT procedure, 259-23  
    TRANSFORM function, 259-24

---

XMLTYPE function, 259-25  
XMLTYPE function, 259-25  
XPath parameters  
    adding to attribute list, 65-19  
    dropping from attribute list, 65-19  
XPath predicates  
    indexing set of  
        configuring XML element or XML attribute, 247-11  
XPINDEX\_PARAMETERS procedure, 65-32  
XRANGE function, 232-39