Oracle® Database Gateway for WebSphere MQ

Installation and User's Guide

12*c* Release 1 (12.1)

E17930-06

April 2013

ORACLE®

Oracle Database Gateway for WebSphere MQ Installation and User's Guide, 12*c* Release 1 (12.1)

E17930-06

Primary Author: Maitreyee Chaliha

Contributing Author: Li-Te Chen

Contributor: The Oracle Database 12*c* documentation is dedicated to Mark Townsend, who was an inspiration to all who worked on this release.

Contents

3   Requirements

4   Preinstallation

5   Installation

6   Removing the Gateway

7   Configuration

8 Gateway Running Environment

A The PGM, PGM_UTL8, and PGM_SUP Packages

B   UTL_RAW Package

C   Gateway Initialization Parameters

Index

Preface

Oracle Database Gateway for WebSphere MQ provides access to WebSphere MQ services. This gateway requires a system that is capable of running 64-bit applications.

Intended Audience

This guide is intended for anyone responsible for installing, configuring, or administering the Oracle Database Gateway for WebSphere MQ. It is also for developers writing applications that access message queuing systems, particularly those developers who need to access queues owned by both WebSphere MQ and other non-Oracle message queuing systems as well as queues owned by Oracle Advanced Queuing (AQ).

Read this guide if you are responsible for tasks such as:

- Administering the gateway
- Setting up gateway security
- Using the gateway
- Diagnosing gateway errors

Before using this guide, you must understand the fundamentals of your operating system, the Oracle Database Gateways, PL/SQL, the Oracle database, and WebSphere MQ software before using this guide to install, configure, or administer the gateway.

Documentation Accessibility

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Product Name

The complete name for this product is Oracle Database Gateway for WebSphere MQ, also called DG4MQ.

Typographic Conventions

The following typographic conventions are used in this guide:

Convention	Description
monospace	Monospace type indicates commands, directory names, user names, path names, and file names.
italics	Italic type indicates variables, including variable portions of file names. It is also used for emphasis and for book titles.
UPPERCASE	Uppercase letters indicate Structured Query Language (SQL) reserved words, initialization parameters, and environment variables.
Bold	Bold type indicates screen names and fields.
SQL*Plus prompts	The SQL*Plus prompt, SQL>, appears in SQL statement and SQL*Plus command examples. Enter your response at the prompt. Do not enter the text of the prompt, "SQL>", in your response.

Command Syntax

Command syntax appears in monospace font. The dollar character ($), number sign (#), or percent character (%) are UNIX command prompts. Do not enter them as part of the command. The following command syntax conventions are used in this guide:

Convention	Description

backslash \	A backslash is the UNIX command continuation character. It is used in command examples that are too long to fit on a single line. Enter the command as displayed (with a backslash) or enter it on a single line without a backslash:
	
	`dd if=/dev/rdsk/c0t1d0s6 of=/dev/rst0 bs=10b \`
	`count=10000`

braces { }	Braces indicate required items: `.DEFINE {macro1}`
brackets [ ]	Brackets indicate optional items: `cvtcrt termname [outfile]`
ellipses ...	Ellipses indicate an arbitrary number of similar items: `CHKVAL fieldname value1 value2 ... valueN`
italics	Italic type indicates a variable. Substitute a value for the variable: `library_name`
vertical line |	A vertical line indicates a choice within braces or brackets: `FILE filesize [K|M]`

Related Publications

See the *Oracle Database Heterogeneous Connectivity User's Guide* for information common to all Oracle Database Gateways, including important information about functions, parameters, and error messages.

Related Documents

The guide includes references to the following documents:

Oracle Call Interface Programmer's Guide

Oracle Database Administrator's Guide

Oracle Database Error Messages

Oracle Database Reference

Oracle Database Utilities

Oracle Database Heterogeneous Connectivity User's Guide

Oracle Database Net Services Administrator's Guide

Oracle Database Net Services Reference

Oracle Database Security Guide

Oracle Database SQL Language Quick Reference

Oracle Database PL/SQL Packages and Types Reference

Oracle Database PL/SQL Language Reference

Oracle Database Installation Guide

1

Introduction

This chapter provides an overview of message queuing, WebSphere MQ, and the role of the gateway when accessing WebSphere MQ queues. It contains the following sections:

- Introduction to Message Queuing on page 1-1
- Introduction to WebSphere MQ on page 1-2
- Introduction to the Gateway on page 1-2

Introduction to Message Queuing

Message queuing enables distributed applications to communicate asynchronously by sending messages between the applications. The messages from the sending application are stored in a queue and are retrieved by the receiving application. The applications send or receive messages through a queue by sending a request to the message queuing system. Sending and receiving applications can use the same or different message queuing systems, allowing the message queuing system to handle the forwarding of the messages from the sender queue to the recipient queue.

Queued messages can be stored at intermediate nodes until the system is ready to forward them to the next node. At the destination node, the messages are stored in a queue until the receiving application retrieves them from the queue. Message delivery is guaranteed even if the network or application fails. This provides for a reliable communication channel between applications.

The complexity and details of the underlying model (of storing and forwarding messages between different environments) are handled by the message queuing system. By maintaining this level of abstraction, distributed applications can be developed without the need to worry about the details of how the information is transported.

Because the sending and receiving applications operate independently of one another, the sending application is less dependent on the availability of the remote application, the network between them, and the system on which the receiving application runs. This leads to a higher level of availability for the participating applications.

Messages and message queue operations can be configured by the applications to operate in specific modes. For example, a sending application can specify that queued messages should survive system crashes. As another example, the receiving application can specify a maximum waiting period for a receiving operation from a queue (in case no messages are available yet on the receiving queue).

Introduction to WebSphere MQ

WebSphere MQ is a message queuing system based on the model of message queue clients and message queue servers. The applications run either on the server node where the queue manager and queues reside, or on a remote client node. Applications can send or retrieve messages only from queues owned by the queue manager to which they are connected.

WebSphere MQ Terms

The following table describes WebSphere MQ terms used in this guide.

Term	Description
Message queues	Storage areas for messages exchanged between applications.
Message queue interface (MQI)	An application programming interface (API) for applications that want to send or receive messages through WebSphere MQ queues.
WebSphere MQ client configuration	A WebSphere MQ configuration where the queue manager and message queues are located on a different (remote) system or node than the application software. Client applications connect to the remote queue manager using IBM software that provides the necessary networking software to connect to the remote queue manager.
WebSphere MQ server configuration	A WebSphere MQ configuration where the queue manager and message queues are located on the same (local) system or node as the application software. Client applications connect to the local queue manager using MQI.
Queue manager	A WebSphere MQ feature that provides the message queuing facilities that applications use. It manages the queue definitions, configuration tables, and message queues. The queue manager also forwards messages from the sender queue to the remote recipient queues.
Triggers	A WebSphere MQ feature that enables an application to be started automatically when a message event, such as the arrival of a message, occurs. Triggers can be used to invoke programs or transactions. For example, a trigger could cause an Oracle application to call the gateway to retrieve a WebSphere MQ message and process it.

Introduction to the Gateway

The Oracle Database Gateway for WebSphere MQ enables Oracle applications to integrate with other WebSphere MQ applications. Oracle applications can send messages to other WebSphere MQ applications or receive messages from them. With the gateway, Oracle applications access WebSphere MQ message queues through remote procedure call (RPC) processing.

The gateway extends the RPC facilities that are available with the Oracle database and enables any client application to use PL/SQL to access messages in WebSphere MQ queues. The gateway provides PL/SQL procedures that are translated by the gateway into MQI calls. These procedures resemble the calls and types of MQI, but they are adapted to take full advantage of the transaction integration with the Oracle database. For more information about these procedures, refer to Appendix A.

Through WebSphere MQ, the gateway communicates with any other WebSphere MQ systems on various platforms, including mainframes, UNIX based systems, Microsoft Windows, and other desktop environments. The gateway does not require any Oracle

software on the remote system. The gateway integrates with existing WebSphere MQ applications without any changes to those applications and enables users to exploit their investment in these applications while providing them with the ability to maximize on the benefits of message-oriented systems.

The gateway also provides a way to integrate these existing WebSphere MQ applications with new technology areas, such as network computing. Any Oracle application can invoke PL/SQL procedures, including applications that use the Oracle Application Server 11*g*.

Developing Gateway Applications

If you are developing applications that access WebSphere MQ through the gateway, use the Oracle Visual Workbench for Oracle Database Gateway for WebSphere MQ. Oracle Visual Workbench enables you to define an interface for accessing WebSphere MQ and define how to convert message data that is sent or retrieved from WebSphere MQ queues.

Visual Workbench generates PL/SQL code for the interface and data conversion. This generated code is called the message interface package (MIP). The MIP provides the underlying code to interact with the gateway, performs message data conversion, and provides an easy-to-use interface for Oracle applications to exchange messages with remote WebSphere MQ applications.

> **See Also:** Refer to the *Oracle Procedural Gateway Visual Workbench for WebSphere MQ Installation and User's Guide for Microsoft Windows (32-Bit)* for more information about Oracle Visual Workbench.

When necessary, the generated MIP code can be modified to use WebSphere MQ functions that are not supported by Visual Workbench or to enhance message data conversions. Refer to Appendix A and Appendix B for more information about modifying the generated MIP code.

Gateway Terms

The following table describes gateway terms used in this guide.

Term	Description
Gateway initialization file	A file containing parameters that determine the running of the gateway.
Gateway remote procedures	Remote procedures implemented by the gateway. These procedures are used to invoke WebSphere MQ operations.
MIP (message interface package)	An Oracle PL/SQL package generated by Oracle Visual Workbench that serves as an interface between an existing WebSphere MQ application and an Oracle application. The MIP performs any necessary data conversion and invokes the gateway RPCs to perform appropriate WebSphere MQ operations. Refer to the *Oracle Procedural Gateway Visual Workbench for WebSphere MQ Installation and User's Guide for Microsoft Windows (32-Bit)* for more information about the generated packages.
Oracle database	Any Oracle database that communicates with the gateway. Oracle applications do not communicate directly with the gateway. Instead, they run PL/SQL code at an Oracle database to invoke the gateway procedures. The Oracle database can be on the same system as the gateway or on a different system.

Term	Description
Production Oracle database	As used in this guide, the production database refers to any Oracle database that you use for production, for actual business and not for testing.
PL/SQL stored procedure	A compiled PL/SQL procedure that is stored in the Oracle database or is included with the gateway.
Remote procedure call	A programming call that invokes a program on a system in response to a request from another system.
Oracle Visual Workbench	An abbreviated term for the Oracle Visual Workbench for Oracle Database Gateway for WebSphere MQ.

Advantages of Using the Gateway

Using the gateway to access WebSphere MQ provides the following advantages:

- Transactional support

  The gateway and the Oracle database enable WebSphere MQ operations and Oracle database updates to be performed in a coordinated fashion. Oracle two-phase commit protection is extended to the WebSphere MQ environment without any special programming.

- Fast remote procedures

  The remote procedures implemented by the gateway are optimized for efficient processing of WebSphere MQ requests.

  The remote procedures to the gateway and WebSphere MQ are an optimized PL/SQL package that is precompiled in the gateway. Because there are no additional software layers on the target system, overhead is minimized.

- Location transparency

  Client applications need not be on a specific operating system. For example, your Oracle application can send WebSphere MQ messages to an application on IBM MVS. If the receiving application is moved to a different platform, then you do not need to change the platform of your Oracle application.

- Flexible interface

  Using the MIPs generated by the Visual Workbench, you can use the gateway to interface with the existing procedural logic or to integrate new procedural logic into an Oracle database environment.

- Oracle database integration

  The integration of the Oracle database with the gateway enables you to benefit from existing and future Oracle features.

- Wide selection of tools

  The gateway supports any tool or application that supports PL/SQL. This includes applications built with traditional Oracle tools, such as Oracle Developer, or applications built for intranet or Internet environments supported by Oracle Application Server 11*g*. The gateway also works with packaged Oracle applications, such as Oracle Financials, and with many third-party tools, such as Visual Basic, PowerBuilder, and Lotus Notes.

- Security

The gateway is compatible with the WebSphere MQ security authorization mechanism.

Gateway Architecture

Figure 1–1 shows the components of the gateway architecture.

Figure 1–1   Components of the Gateway Architecture
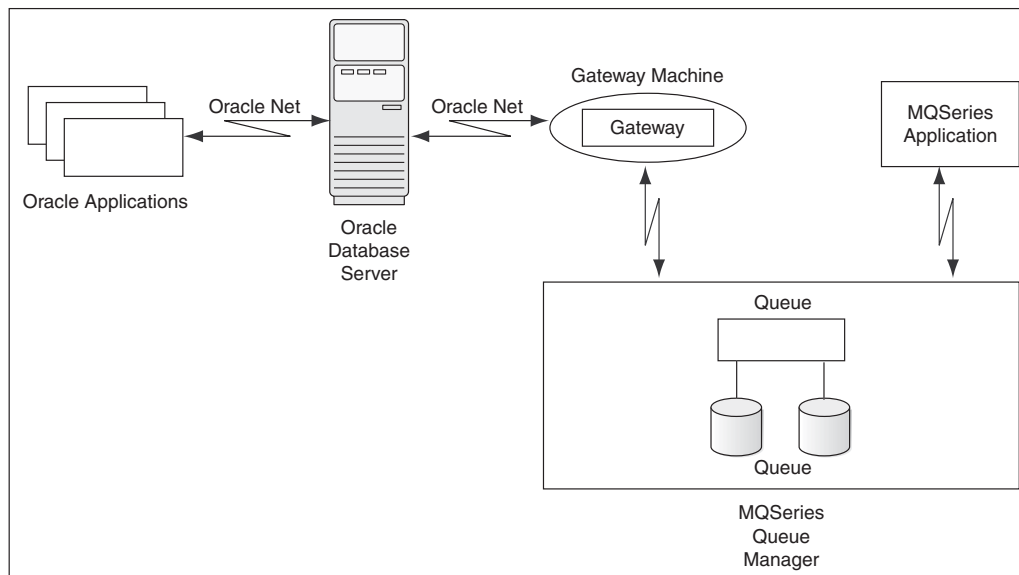
Component Descriptions

This section describes components of the gateway architecture.

Oracle Applications

Oracle applications connect to an Oracle database. They send data to and receive data from WebSphere MQ queues by invoking the gateway RPCs.

Oracle Database

Oracle applications do not connect directly to the gateway, but connect indirectly through an Oracle database. The Oracle database communicates with a gateway in the normal Oracle server-to-server manner using Oracle Net. The gateway is a single process and does not start background processes. On UNIX platforms, a gateway process is started for each user session.

Oracle Net

Oracle Net provides client to server and server-to-gateway communication. It enables an Oracle application to communicate with the Oracle database, and it enables the Oracle database to communicate with the gateway.

If the Oracle database is not on the same system as the gateway, then you must install the correct Oracle networking software on the platform where the Oracle database is installed.

Gateway

Oracle applications invoke the RPCs that are implemented by the gateway with PL/SQL. The gateway procedures map these RPCs to WebSphere MQ MQI calls to perform the corresponding WebSphere MQ operation.

The gateway is accessed through the Oracle database by using a database link name created by an Oracle `CREATE DATABASE LINK` statement. The database link is the construct used to identify Oracle databases.

WebSphere MQ Queue Manager

The WebSphere MQ server is where the WebSphere MQ queue manager and message queue are located. The WebSphere MQ server might, or might not, be on the same system as the gateway.

WebSphere MQ Application

WebSphere MQ applications connect directly to the WebSphere MQ queue manager by using WebSphere MQ MQI calls to perform the corresponding WebSphere MQ operation.

Gateway Structure

The gateway has some of the same components as an Oracle database. The following components are included:

- A directory where the gateway software is installed
- A system identifier (SID)
- An initialization file similar to the Oracle database initialization parameter file

The gateway does not have control, redo, or database files, nor does it have the full set of subdirectories and other files associated with an Oracle database.

Gateway Operation

The gateway is not started in the same way as the Oracle database. It has no background processes and does not require a management utility such as Oracle Enterprise Manager. Each Oracle database user session that accesses a gateway creates an independent process on the host system that runs the gateway.

Communication

All communication between the Oracle database, gateway, and WebSphere MQ queues is handled through RPC calls to the gateway. The PL/SQL code to do these calls is automatically generated by the Visual Workbench. For more information about communication between the gateway, the Oracle database, and WebSphere MQ, refer to Appendix A, "The PGM, PGM_UTL8, and PGM_SUP Packages".

2

Release Information

This chapter contains information that is specific to this release of the gateway. It contains the following sections:

- Changes and Enhancements on page 2-1
- Known Problems for 12c Release 1 on page 2-3
- Known Restrictions for 12c Release 1 on page 2-3

Changes and Enhancements

The following changes and enhancements apply to this release of Oracle Database Gateway for WebSphere MQ.

Changes and Enhancements

The following sections describe the changes and enhancements included in this release.

Oracle Database Dependencies

This release of Oracle Database Gateway for WebSphere MQ requires the latest released patch set for Oracle Database 12*c* Release 1 (12.1), or for the Oracle database release that you are using.

Support for Large Data Buffers

The PL/SQL RAW data type limitation is 32 KB (32767) bytes. For large loads, you must use the TABLE OF RAWS data type. For more information about support for large data buffers, refer to Appendix A.

DG4MQ Data Types

The following table provides information about Oracle Database Gateway for WebSphere MQ (DG4MQ) data types.

Data Type	V401	V804	V817 and V901	Oracle10*g* Release 2 and higher
MQOD	PGM.MQOD@dblink	PGM.MQOD	PGM.MQOD	PGM.MQOD
MQMD	PGM.MQMD@dblink	PGM.MQMD	PGM.MQMD	PGM.MQMD
MQPMO	PGM.MQPMO@dblink	PGM.MQPMO	PGM.MQPMO	PGM.MQPMO
MQGMO	PGM.MQGMO@dblink	PGM.MQGMO	PGM.MQGMO	PGM.MQGMO

Data Type	V401	V804	V817 and V901	Oracle10*g* Release 2 and higher
MQODRAW	NA	PGM.MQODRAW	PGM8.MQODRAW	NA
MQMDRAW	NA	PGM.MQMDRAW	PGM8.MQMDRAW	NA
MQPMORAW	NA	PGM.MQPMORAW	PGM8.MQPMORAW	NA
MQGMORAW	NA	PGM.MQGMORAW	PGM8.MQGMORAW	NA

PGM_UTL Procedures

The following table provides information about PGM_UTL procedures.

Procedure	V401	V804	V817 and V901	Oracle10*g* Release 2 and higher
TO_RAW	NA	PGM_UTL.TO_RAW	PGM_UTL8.TO_RAW	PGM.TO_RAW
RAW_TO_MQMD	NA	PGM_UTL.RAW_TO_MQMD	PGM_UTL8.RAW_TO_MQMD	PGM.RAW_TO_MQMD
RAW_TO_MQPMO	NA	PGM_UTL.RAW_TO_MQPMO	PGM_UTL8.RAW_TO_MQPMO	PGM.RAW_TO_MQPMO
RAW_TO_MQGMO	NA	PGM_UTL.RAW_TO_MQGMO	PGM_UTL8.RAW_TO_MQGMO	PGM.RAW_TO_MQGMO

> **Note:** For Oracle10*g* release 10.2.0, the PGM.TO_RAW, PGM.RAW_TO_MQMD, PGM.RAW_TO_MQPMO and PGM.RAW_TO_MQGMO procedures are added for backward compatibility.

DG4MQ API Prototype Changes

The following table provides information about DG4MQ application programming interface changes.

API	V401 Arguments	V804 Arguments	V817 & V901 Arguments	10g Release 2 and higher Arguments
MQOPEN	(MQOD,INT,INT)	(RAW,INT,INT)	(RAW,INT,INT)	(PGM.MQOD,INT,INT)
MQPUT	(INT,MQMD,MQPMO,RAW)	(INT,RAW,RAW,RAW)	(INT,RAW,RAW,RAW)	(INT,PGM.MQMD,PGM_MQPMO,RAW) or (INT, PGM.MQMD, PGM_MQPMO, PGM.MQPUT_BUFFER)
MQGET	(INT,MQMD,MQGMO,RAW)	(INT,RAW,RAW,RAW)	(INT,RAW,RAW,RAW)	(INT,PGM.MQMD,PGM_MQGMO,RAW) or (INT, PGM.MQMD, PGM.MQGMO, PGM_MQGET_BUFFER)
MQCLOSE	(INT,INT)	(INT,INT)	(INT,INT)	(INT,INT)

Refer to Appendix A for the details of APIs.

DG4MQ Deployment Scripts

The following scripts are new in this release:

- pgm.sql
- pgmobj.sql

- `pgmdeploy.sql`

- `pgmundeploy.sql`

The gateway procedures in the PGM package are defined in `pgm.sql` and PGM_MQ* data type definitions used by the procedures are defined in `pgmobj.sql`. For complete information about PGM package, DG4MQ gateway procedures and data type definitions, refer to Appendix A.

Large Payload Support

DG4MQ 11*g* supports large payloads or messages longer than 32767 bytes. For more information, refer to the `putlongsample.sql` and `getlongsample.sql` sample programs installed with the DG4MQ.

Database Link and Alias Library

A connection to the gateway is established through a database link. From DG4MQ 10*g* release 2 and later, this database link is no longer associated with each DG4MQ gateway procedural call (for example, `PGM.MQPUT@dblink`). From 10*g* release 2 and later, it needs to be defined only once in the `MQOD` data type used by `MQOPEN`, and this database link is registered in the object handle returned by the `MQOPEN` call. Refer to the sample programs installed with the gateway for details. By default, a public database link, `dg4mqdepdblink`, is created with your default SID when DG4MQ deployment scripts are executed.

Known Problems for 12*c* Release 1

The problems documented in this section are specific to the Oracle Database Gateway for WebSphere MQ and are known to exist in this release of the product. These problems will be fixed in a future gateway release. If you have any questions or concerns about these problems, contact Oracle Support Services.

A current list of problems is available online. Contact your local Oracle office for information about accessing this online information.

Known Restrictions for 12*c* Release 1

The following restriction is known to exist for this release.

Customizing LOG_DESTINATION

There is a known issue when customizing the gateway initialization file for gateway tracing for Microsoft Windows platform. When customizing the path name of `LOG_DESTINATION`, the delimiter must be double backslashes. For example:

```
LOG_DESTINATION=C:\\oracle\\product\\12.1\\dg4mqs\\dg4mq\\log\\dg4mqs.log
```

> **Note:** If `LOG_DESTINATION` is not defined for Microsoft Windows platform, a default name is used and the log is created in `ORACLE_HOME\dg4mq\trace` directory

Customizing deployment script pgmobj.sql

There is a known issue when customizing the gateway deployment script `pgmobj.sql` for Microsoft Windows platform. When defining the path name of `libdg4mq`, the delimiter must be backslashes. For example create or replace library `libdg4mq` as:

```
CREATE OR REPLACE LIBRARY libdg4mq as
'C:\oracle\product\12.1\dg4mqs\bin\oradg4mqs.dll' transactional
```

or

```
CREATE OR REPLACE LIBRARY libdg4mq as '$ORACLE_HOME\bin\oradg4mqs.dll'
transactional
```

CALLBACK links

Oracle Database Gateway for WebSphere MQ does not support CALLBACK links. Trying a CALLBACK link with the gateway will return the following error message:

```
ORA-02025: All tables in the SQL statement must be at the remote database
```

3

Requirements

This chapter provides information about the hardware and software required for the installation of Oracle Database Gateway for WebSphere MQ and the recommended online documentation. It contains the following sections:

- Hardware Requirements on page 3-1
- Software Requirements on page 3-1
- Oracle Database on page 3-3

Hardware Requirements

The following table contains the hardware requirements for Oracle Database Gateway for WebSphere MQ.

Table 3–1 Hardware Requirements for Oracle Database Gateway for WebSphere MQ

Hardware Items	Required for IBM AIX on POWER Systems (64-Bit)	Required for Linux x86 64 bit	Required for Oracle Solaris on SPARC (64-Bit)	Required for Oracle Solaris on x86-64 (64-Bit)	Required for HP-UX Itanium	Required for Microsoft Windows 64 bit
Temporary Disk space	400 MB	400 MB	400 MB	400 MB	400 MB	300 MB
Disk space	1.5 GB	750 MB	750 MB	750 MB	1.5 GB	300 MB
Physical Memory	512 MB	512 MB	512 MB	512 MB	512 MB	512 MB
Swap space	1 GB	1 GB	1 GB	1 GB	1 GB	NA
Processor	IBM RS/6000 AIX-Based System Processor	x86_64	Sun Solaris Operating System (SPARC) Processor	x86_64	HP Itanium processor for hp-ux 11	Intel Pentium or compatible

Software Requirements

The following table contains the software requirements for Oracle Database Gateway for WebSphere MQ.

Table 3–2   Software Requirements for Oracle Database Gateway for WebSphere MQ

Platform	Requirement	WebSphere MQ Server Software
IBM AIX on POWER Systems (64-Bit)	AIX 5L version 5.3 TL9 or higher, AIX 6.1 OS Patches: Check with your software vendor for current maintenance requirements.	When the gateway resides on the same system as the WebSphere MQ server software, then WebSphere MQ for AIX version 6.0 or later is required. When the gateway resides on a different system than the WebSphere MQ server software, then WebSphere MQ Client for AIX version 6.0 or later is required on the gateway system.
Linux x86 64 bit	One of the following operating system versions: ■ Red Hat Enterprise Linux 4.0, (Update 7 or later) ■ Red Hat Enterprise Linux 5 Update 2 ■ Red Hat Enterprise Linux 6 ■ Oracle Linux 4.0, (Update 7 or later) ■ Oracle Linux 5 Update 2 ■ Oracle Linux 5 Update 5 (with the Oracle Unbreakable Enterprise Kernel for Linux) ■ Oracle Linux 6 ■ Oracle Linux 6 (with the Oracle Unbreakable Enterprise Kernel for Linux) OS Patches: Check with your software vendor for current maintenance requirements	When the gateway resides on the same system as the WebSphere MQ server software, then WebSphere MQ for IA Linux x 86 64 bit version 6.0 or later is required. When the gateway resides on a different system than the WebSphere MQ server software, then WebSphere MQ Client for IA Linux x86 64 bit version 6.0 or later is required on the gateway system.
Oracle Solaris on SPARC (64-Bit)	Solaris 9 Update 6 or later Solaris 10 OS Patches: Check with your software vendor for current maintenance requirements.	When the gateway resides on the same system as the WebSphere MQ server software, then WebSphere MQ for Sun Solaris version 6.0 or later is required. When the gateway resides on a different system than the WebSphere MQ server software, then WebSphere MQ Client for Sun Solaris version 6.0 or later is required on the gateway system.
Oracle Solaris on x86-64 (64-Bit)	Solaris 9 Update 6 or later Solaris 10 OS Patches: Check with your software vendor for current maintenance requirements.	When the gateway resides on the same system as the WebSphere MQ server software, then WebSphere MQ for Sun Solaris version 6.0 or later is required. When the gateway resides on a different system than the WebSphere MQ server software, then WebSphere MQ Client for Sun Solaris version 6.0 or later is required on the gateway system.
HP-UX Itanium	HP-UX 11i V3 patch Bundle Sep/ 2008 (B.11.31.0809.326a) or higher	-
Microsoft Windows x64	Microsoft Windows Server 2003 - all x64 editions. The latest service pack for your operating system, MS Windows 2000 (Service Pack 1) Microsoft Windows Server 2003 R2 - all x64 editions Microsoft Windows XP Professional x64 Edition Microsoft Windows Vista x64 - Business, Enterprise, and Ultimate editions Microsoft Windows 2008 x64	When the gateway resides on the same system as the WebSphere MQ server software, or when the gateway resides on a different system than the WebSphere MQ server software, WebSphere MQ version 7.0 or later is required on the gateway system

Set the `ulimit` value for the maximum number of open files per process to 1024 or greater:

```
prompt> ulimit -n 1024
```

> **Note:** All IBM software must be installed before the gateway software is installed. For example, if WebSphere MQ software is not installed before DG4MQ, then the product link fails.

Oracle Database

The Oracle database requires the latest patch set for Oracle Database 11*g*.

4

Preinstallation

This chapter guides you through the basic concepts and preinstallation steps for Oracle Database Gateway for WebSphere MQ. It contains the following topics:

- Preinstallation Tasks on page 4-1
- About Oracle Universal Installer on page 4-5

Preinstallation Tasks

The preinstallation tasks for the Oracle Database Gateway for WebSphere MQ are divided into the following parts:

- WebSphere MQ Software
- Setting Environment Variables
- Using Windows User Account as Oracle Home User

WebSphere MQ Software

Perform the following steps to check for WebSphere MQ software:

1. Determine where the WebSphere MQ queue manager runs.

   - Local system

     If the WebSphere MQ queue manager runs on a local system, then the queue manager runs on the same system where you intend to install the gateway product set.

   - Remote system

     If the WebSphere MQ queue manager runs on a remote system, then the queue manager runs on a different system, not the system where you intend to install the gateway product set.

2. Verify that the WebSphere MQ software is already installed. If the WebSphere MQ server software is installed on a different system than the gateway, then the WebSphere MQ client software must be installed on the gateway system.

3. Identify the name of the WebSphere MQ queue manager.

4. Identify the WebSphere MQ client channel definition.

If the queue manager is installed on a different system than the gateway, then the WebSphere MQ client software is used to access the remote queue manager. A channel definition is required for this configuration.

Setting Environment Variables

Before installing Oracle Database Gateway for WebSphere MQ on UNIX platforms, set the following environment variables:

- ORACLE_HOME

- ORACLE_SID

- DISPLAY

- TMP

> **Note:** Verify that the values that you assign to the environment variables, which are listed in this section, are less than 42 characters long. Longer values might generate errors such as "Word too long" during installation.

ORACLE_HOME

ORACLE_HOME is the root directory in which Oracle software is installed.

Oracle Database Gateway for WebSphere MQ cannot share the same Oracle home directory with other Oracle products. If you have installed other Oracle products, then Oracle Database Gateway for WebSphere MQ must be installed in a different ORACLE_HOME directory. Refer to Preventing Conflicts Between ORACLE_HOME Directories on page 4-2.

> **Note:** Do not install Oracle Database Gateway for WebSphere MQ in an ORACLE_HOME directory containing other Oracle products, including the database. Such an installation could overwrite shared components, causing the products to malfunction.

Preventing Conflicts Between ORACLE_HOME Directories

To prevent a conflict between the software in an existing *ORACLE_HOME* directory and Oracle Database Gateway for WebSphere MQ, you must remove all references to the existing ORACLE_HOME directory. The following steps describe removing these references.

1. Unset your existing ORACLE_HOME variable using one of the following commands.

   - C shell:

     ```
     prompt> unsetenv ORACLE_HOME
     ```

   - Bourne/Korn shell:

     ```
     prompt> export ORACLE_HOME=
     ```

2. Edit the following environment variables so that they do not use the existing ORACLE_HOME value:

Table 4–1   Setting Environment Variables for a New ORACLE_HOME Directory

Environment Variable	Platforms
PATH	Linux, AIX-based Systems, HP-UX Itanium, and Sun Solaris
CLASSPATH	Linux, AIX-based Systems, HP-UX Itanium, and Sun Solaris
LD_LIBRARY_PATH	Linux and Sun Solaris

Table 4–1   (Cont.) Setting Environment Variables for a New ORACLE_HOME Directory

Environment Variable	Platforms
LIBPATH	AIX-based Systems
SHLIB_PATH	HP-UX Itanium

> **Note:**   Verify that the C compiler is in your PATH before you start the installation.

Setting ORACLE_HOME

Set the ORACLE_HOME environment variable by using one of the following commands:

■   C shell

```
prompt> setenv ORACLE_HOME fullpath
```

■   Bourne/Korn shell

```
prompt> ORACLE_HOME=fullpath
prompt> export ORACLE_HOME
```

ORACLE_SID

ORACLE_SID is used for the SID of the gateway.

Setting ORACLE_SID

Set the ORACLE_SID environment variable by using one of the following commands:

■   C shell

```
prompt> setenv ORACLE_SID dg4mqs
```

or

```
prompt> setenv ORACLE_SID dg4mqc
```

■   Bourne/Korn shell

```
prompt> ORACLE_SID=dg4mqs
promtp> export ORACLE_SID
```

or

```
prompt> ORACLE_SID=dg4mqc
promtp> export ORACLE_SID
```

DISPLAY

Setting the DISPLAY environment variable enables you to run Oracle Universal Installer remotely from a local work station. On the system where you run Oracle Universal Installer, set the DISPLAY environment variable to specify the system name or IP address of your local workstation.

If you get an Xlib error when starting Oracle Universal Installer such as "Failed to connect to server", "Connection refused by server", or "Can't open display", then run the commands on your local workstations as follows:

On Server where the Installer is Running

- C shell

  ```
  prompt> setenv DISPLAY hostname:0.0
  ```

- Borne or Korn shell

  ```
  prompt> export DISPLAY=hostname:0.0
  prompt> export DISPLAY
  ```

In Session on Your Workstation

- C shell

  ```
  prompt> xhost +server_name
  ```

- Borne or Korn shell

  ```
  prompt> xhost +server_name
  ```

TMP

During installation, Oracle Universal Installer uses a temporary directory for swap space. This directory must meet the "Hardware Requirements" on page 3-1. The installation might fail if you do not have sufficient space. Oracle Universal Installer checks for the TMP environment variable to locate the temporary directory. If this environment variable does not exist, then the installer uses the /tmp directory.

The following example demonstrates how to set the TMP environment variable.

- C shell

  ```
  prompt> setenv TMP full path
  ```

- Bourne/Korn shell

  ```
  prompt> TMP=full path
  prompt> export TMP
  ```

Using Windows User Account as Oracle Home User

With Windows, you log in to a user with Administrator privileges to install the Oracle Database software. You can also specify an Oracle Home User (based on a low-privileged, non-administrative user account) during installation.

The following are the Windows User Accounts:

- Windows Local User account

- Windows Domain User account

- Windows Managed Services Account (MSA)

- Windows Built-in Account

  > **See Also:** "Using Oracle Home User on Windows" in *Oracle Database Platform Guide for Microsoft Windows*

About Oracle Universal Installer

Oracle Database Gateway for WebSphere MQ uses Oracle Universal Installer to configure environment variables and install components. Oracle Universal Installer guides you through each step of the installation process, so you can choose configuration options for a customized product.

The Oracle Universal Installer includes features that perform the following tasks:

- Explore and provide installation options for products
- Detect preset environment variables and configuration settings
- Set environment variables and configuration during installation
- Uninstall products

oraInventory Directory

The Oracle Universal Installer creates the `oraInventory` directory the first time it is run on your system. The `oraInventory` directory keeps an inventory of the products that Oracle Universal Installer installs on your system as well as other installation information. If you have previously installed Oracle products, then you might already have an `oraInventory` directory.

- When a UNIX group name is specified, it grants that group the permission to write to the `oraInventory` directory. If another group attempts to run Oracle Universal Installer, then they must have permission to write to the `oraInventory` directory. If they do not have permission the installation fails.

- The user running the Oracle Universal Installer must have permission to write to the `oraInventory` directory and all its files. This is required to run the installer.

- The location of `oraInventory` is defined in `/etc/oratab/oraInst.loc` for HP-UX Inanium and AIX-Based Systems and `C:\Program Files\Oracle\Inventory\` for Microsoft Windows.

- The location of `oraInventory` is defined in `/var/opt/oraInst.loc` for Sun Solaris.

- The latest log file is `oraInventory_location/logs/installActions.log` On UNIX based systems and `C:\Program Files\Oracle\Inventory\logs\installActions.log` for Microsoft Windows. Log file names of previous installation sessions are in the following format: `installActionsdatetime.log`.

- Do not delete or manually alter the `oraInventory` directory or its contents. Doing this can prevent the Oracle Universal Installer from locating the products that you have installed on your system.

Starting Oracle Universal Installer

On UNIX based systems, perform the following steps to launch Oracle Universal Installer, which installs Oracle Database Gateway for WebSphere MQ:

1. Stop all Oracle processes and services (for example, the Oracle database).

2. Run Oracle Universal Installer.

> **Note:** Be sure you are not logged in as the `root` user when you start Oracle Universal Installer. If you are, only the `root` user will have permissions to manage Oracle Database Gateway for WebSphere MQ.

    a. Log in as the Oracle Database Gateway for WebSphere MQ user.

    b. Start Oracle Universal Installer by entering:

```
prompt> mount_point/runInstaller
```

On Microsoft Windows, perform the following steps to launch Oracle Universal Installer, which installs Oracle Database Gateway for WebSphere MQ:

1. Start your system and select MS Windows from the operating system Loader option. Log in to your MS Windows system as a member of the Administrators group.

2. If you are installing the gateway for the first time, ensure there is sufficient space on the disk where you are installing the gateway as specified in "Hardware Requirements" on page 3-1.

3. Before installing the software, stop all Oracle NT Services that are running:

    a. From the **Start** menu, go to **Setting**, then **Control Panel**, and then click **Services**. A list of all NT services is displayed.

    b. Select an Oracle NT service (these services begin with Oracle).

    c. Click **Stop**.

    d. Continue to select and stop all Oracle NT services until all active Oracle NT Services are stopped.

4. Load the installation media and start the Oracle Universal Installer.

This launches Oracle Universal Installer using which you can install Oracle Database Gateway for WebSphere MQ.

5

Installation

This chapter guides you through the installation of the Enterprise Edition of Oracle Database Gateway for WebSphere MQ. It contains the following sections:

- Installation on page 5-1
- Running root.sh on UNIX Based Systems on page 5-2

Installation

Table 5–1 guides you through the installation process. For each screen, perform the actions described in the Response column.

Table 5–1   Installing the software

Screen	Response
Welcome	Click **Next**.
File Locations	The Source section of the screen is where you specify the source location that Oracle Universal Installer uses to install Oracle Database Gateway for WebSphere MQ. You need not edit the file specification in the Path field - the default setting for this field points to the Oracle Universal Installer file on your Oracle Database Gateway for WebSphere MQ CD-ROM.
	The Path field in the Destination section of the File Locations screen is where you specify the destination for your installation. You need not edit the path specification in the Path field. The default setting for this field points to ORACLE_HOME. After you set the fields in the File Locations screen, as necessary, click **Next** to continue. After loading the necessary information from the CD-ROM, the installer displays the Available Products screen.
Available Products	Select **Oracle Database 11***g* and click **Next** to continue. Oracle Universal Installer displays the Installation Types screen.
Installation Types	Select **Custom** and click **Next** to continue. Oracle Universal Installer displays the Available Product Component screen.
Available Product Components	Use the check boxes to indicate the product components that you want to install. By default, all the available components are selected for you. You need to de-select the components that you do not want by clicking on the check boxes. Click **Next** to continue, and Oracle Universal Installer displays the Where is the WebSphere MQ Queue Manager Installed? screen.
Where is the WebSphere MQ Queue Manager Installed?	Select **Local** if the MQM runs on the same system as the gateway, or select **Remote** if the MQM runs on a different system than the gateway. Click **Next** to continue.

Table 5–1 (Cont.) Installing the software

Screen	Response
Local WebSphere MQ Queue Manager Name	If you choose Local for your MQM in the Where is the WebSphere MQ Queue Manager Installed? screen, then the Local WebSphere MQ Queue Manger Name screen is displayed. Type in the local WebSphere MQ queue manager name in the Queue Manager field. Click Next to continue, and Oracle Universal Installer displays the Summary screen.
Remote WebSphere MQ Queue Manager Name	If you choose Remote for your MQM in the Local WebSphere MQ Queue Manager Name screen, then the Remote WebSphere MQ Queue Manager Name screen is displayed. Enter the name for the remote WebSphere MQ queue manger in the Queue Manager field, and also enter the WebSphere MQ channel name in the Channel field.
	For information about server connection channels, refer to the IBM publication about WebSphere MQ Clients or ask your WebSphere MQ system administrator for the channel definition of the queue manager to which you want the gateway to connect.
	The definition syntax is:
	`CHANNEL_NAME/PROTOCOL/server_address[(port)]`
	where `CHANNEL_NAME` and `PROTOCOL` must be in uppercase, and `server_address` is the TCP/IP host name of the server. The port value is optional and is the TCP/IP port number on which the server is listening.
	If you do not provide a port number, then WebSphere MQ uses the port number that is specified in the `QM.INI` file. If no value is specified in the `QM.INI` file, then WebSphere MQ uses the port number that is identified in the TCP/IP services file for the WebSphere MQ services name. If this entry in the services file does not exist, then the default value of `1414` is used. It is important that the port number that is used by the client and the port number that is used by the server listener program be the same.
	For example: `CHANNEL1/TCP/Sales`
	Click **Next** to continue. The Oracle Universal Installer displays the Summary screen.
Oracle Universal Installer Summary	This screen enables you to review a tree list of options and components for this installation. Click **Install** to display the Installation Status screen.
Installation Status	The Installation Status screen displays the status of the installation as it proceeds, as well as the location of the Oracle Universal Installer log file for this installation session.
	Be patient as Oracle Universal Installer processes the software installation. Depending on the CPU, CD-ROM drive, and hard drive on your system, the installation process might take time to complete.
End of Installation	This is the final screen of Oracle Universal Installer in the installation process. Assuming that your installation was successful, click **Exit** to exit the installer.

Running root.sh on UNIX Based Systems

After you complete the installation, perform the following steps to run the `root.sh` script:

1. Log on as the `root` user.

2. Go to the `$ORACLE_HOME/dg4mq/admin` directory for your WebSphere MQ gateway.

   ```
   prompt> cd $ORACLE_HOME/dg4mq/admin
   ```

3. Run the `root.sh` script.

   ```
   prompt> ./root.sh
   ```

   This script enables the WebSphere MQ gateway to operate for the strict security model.

4. Exit the `root` account.

6

Removing the Gateway

This chapter describes how to remove Oracle Database Gateway from an Oracle home directory. It contains information about the following topics:

- Removing the Gateway
- Reinstalling the gateway

Removing the Gateway

To remove Oracle Database Gateway for WebSphere MQ perform the following steps:

- About the Deinstallation Tool
- Removing Oracle Software

About the Deinstallation Tool

The Deinstallation Tool (`deinstall`) is available in the installation media before installation, and is available in Oracle home directories after installation. It is located in the path `$ORACLE_HOME/deinstall`.

The `deinstall` command stops Oracle software, and removes Oracle software and configuration files on the operating system.

The command uses the following syntax, where variable content is indicated by italics:

```
deinstall -home complete path of Oracle home [-silent] [-checkonly] [-local]
[-paramfile complete path of input parameter property file] [-params name1=value
name2=value . . .] [-o complete path of directory for saving files] [-help | -h]
```

The options are:

- `-silent`

  Use this flag to run the command in silent or response file mode. If you use the `-silent` flag, then you must use the `-paramfile` flag, and provide a parameter file that contains the configuration values for the Oracle home that you want to deinstall or deconfigure.

  You can generate a parameter file to use or modify by running `deinstall` with the `-checkonly` flag. The `deinstall` command then discovers information from the Oracle home that you want to deinstall and deconfigure. It generates the properties file, which you can then use with the `-silent` option.

  You can also modify the template file `deinstall.rsp.tmpl`, located in the response folder.

- `-checkonly`

Use this flag to check the status of the Oracle software home configuration. Running the command with the -checkonly flag does not remove the Oracle configuration. The -checkonly flag generates a parameter file that you can use with the deinstall command.

- -local

    Use this flag on a multinode environment to deconfigure Oracle software in a cluster.

    When you run deconfig with this flag, it deconfigures and deinstalls the Oracle software on the local node (the node where deconfig is run). On remote nodes, it deconfigures Oracle software, but does not deinstall the Oracle software.

- -paramfile *complete path of input parameter property file*

    Use this flag to run deconfig with a parameter file in a location other than the default. When you use this flag, provide the complete path where the parameter file is located.

    The default location of the parameter file depends on the location of deconfig:

    - From the installation media or stage location: $ORACLE_HOME/inventory/response for UNIX based system and ORACLE_HOME\response for Microsoft Windows.

    - From a unzipped archive file from OTN: /ziplocation/response for UNIX based system and ziplocation\response for Microsoft Windows.

    - After installation from the installed Oracle home: $ORACLE_HOME/deinstall/response for UNIX based system and ORACLE_HOME\deinstall\response for Microsoft Windows.

- -params [name1=*value* name 2=*value* name3=value . . .]

    Use this flag with a parameter file to override one or more values that you want to change in a parameter file you have already created.

- -o *complete path of directory for saving response files*

    Use this flag to provide a path other than the default location where the properties file (deinstall.rsp.tmpl) is saved.

    The default location of the parameter file depends on the location of deconfig:

    - From the installation media or stage location before installation: $ORACLE_HOME/ for UNIX based system and ORACLE_HOME\ for Microsoft Windows

    - From a unzipped archive file from OTN: /ziplocation/response/ for UNIX based system and \ziplocation\response\ for Microsoft Windows

    - After installation from the installed Oracle home: $ORACLE_HOME/deinstall/response for UNIX based system and ORACLE_HOME\deinstall\response for Microsoft Windows

- -help | -h

    Use the help option (-help or -h) to obtain additional information about the command option flags.

Removing Oracle Software

Complete the following procedure to remove Oracle software:

1. Log in as the installation owner.

2. Run the `deinstall` command, providing information about your servers as prompted.

Reinstalling the gateway

To reinstall Oracle Database Gateway for WebSphere MQ over the same version, first remove, and then reinstall the product. Also refer to Removing the Gateway on page 6-1.

7

Configuration

After installing the gateway, follow the instructions in this chapter to configure the gateway. This chapter contains the following sections:

- Configuration Overview on page 7-1
- Configuring the Gateway on page 7-1
- Configuring Oracle Net for the Gateway on page 7-4
- Configuring Oracle Net for Oracle Database on page 7-8
- Creating a Transaction Log Queue on page 7-10
- Administering the Database Links Alias Library on page 7-10
- Installing the Oracle Visual Workbench Repository on page 7-12
- Preparing the Production Oracle Database on page 7-16

Configuration Overview

The gateway works with several components and products to communicate between the Oracle database and WebSphere MQ queues:

- Oracle Net

  The gateway and the Oracle database communicate using Oracle Net in a server-to-server manner. You must configure both, the gateway and Oracle database to have Oracle Net communication enabled, by configuring the `tnsnames.ora` and `listener.ora` files.

- Gateway initialization files and parameters

  The gateway has initialization files and parameters that you must customize for your installation. For example, you must choose your gateway system identifier (SID), and provide other information, such as the gateway log file destination.

Configuring the Gateway

The gateway is installed and preconfigured using default values for the gateway SID, directory names, file names, and gateway parameter settings. The default SID values are:

- `dg4mqs`

  This is the default SID that is used when the gateway resides on the same system as the WebSphere MQ software.

- `dg4mqc`

    This is the default SID that is used when the gateway resides on a different system than the WebSphere MQ software. In this case, the gateway functions as a remote WebSphere MQ client.

A basic gateway initialization file is also installed, and values in this file are set based on the information you entered during the installation phase.

Using the Gateway with the Default Values

If you are configuring one gateway instance, and if you have no need to change any of the default values, then most of the gateway configuration process is completed by Oracle Universal Installer. In this case, perform the following actions:

- Skip all steps under "Changing Default Values" on page 7-2.

- Skip"Step 1: Configure the Oracle Net Oracle Net Listener for the Gateway" on page 7-4

- Begin with "Step 2: Stop and Start the Oracle Net Listener for the Gateway" on page 7-6, and continue to the end of the chapter.

Using the Gateway Without the Default Values

If multiple instances of the gateway are being configured, or to modify the default values set during the installation phases, then begin with the steps under "Changing Default Values" on page 7-2 and continue to the end of the chapter.

Changing Default Values

When changing default values, choose a gateway SID and customize the gateway initialization file.

Step 1: Choose a System ID for the Gateway

The gateway SID is a string of 1 to 64 alphanumeric characters that identifies a gateway instance. The SID is used in the gateway boot file and as part of the file name for the gateway parameter file. Choose a SID different from the default SID and different from `dg4mqs` and `dg4mqc`.

You need a distinct gateway instance, and SID, for each queue manager you want to access. If you want to access two different queue managers, then you need two gateway SIDs, one for each instance of the gateway. If you have one queue manager and want to access it sometimes with one set of gateway parameter settings and at other times with different gateway parameter settings, then you can do this by having multiple gateway SIDs for one queue manager.

Step 2: Customize the Gateway Initialization File

The gateway initialization file (`init`*`sid`*`.ora`) supports all database gateway initialization parameters described in Appendix C, "Gateway Initialization Parameters". The initialization file must be available when the gateway is started.

During installation, a default initialization file is created in `ORACLE_HOME\dg4mq\admin\init`*`sid`*`.ora` on Microsoft Windows and `$ORACLE_HOME/dg4mq/admin/init`*`sid`*`.ora`, on UNIX based systems where *`sid`* is the default SID of `dg4mqs` or `dg4mqc`. If you chose an SID other than the default, then rename this file using the SID you chose in Step 1: Choose a System ID for the Gateway. Customize the default initialization file as necessary.

The following entries might appear in an initialization file:

```
LOG_DESTINATION=log_file
QUEUE_MANAGER=manager_name
AUTHORIZATION_MODEL=auth_model
TRANSACTION_MODEL=tx_model
TRANSACTION_LOG_QUEUE=tx_queue_name
TRANSACTION_RECOVERY_USER=rec_user
TRANSACTION_RECOVERY_PASSWORD=rec_password
TRACE_LEVEL=0
MQSERVER=channel
MQCCSID=character_set
```

> **See Also:** Refer to *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* for additional information

In this file:

- *log_file* specifies the full path name of the gateway log file.

- *manager_name* is the name of the WebSphere MQ queue manager to access.

- *auth_model* is the authorization model to use. The default value is RELAXED.

- *tx_model* is the transaction model to use. The default is SINGLE_SITE.

- *tx_queue_name* is the name of the queue for logging transaction IDs for distributed transactions. This is used only when *tx_model* is set to COMMIT_CONFIRM.

- *rec_user* specifies the user name that the gateway uses to start recovery of a distributed transaction. This is used only when *auth_model* is set to STRICT and tx_model is set to COMMIT_CONFIRM.

- *rec_password* specifies the password of the user name that the gateway uses to start recovery of a distributed transaction.

- *channel* specifies the location of the WebSphere MQ server and the communication method to use. The channel format is:

  channel_name/connection_type/hostname [(port_number)].

  For example:

  MQSERVER=CHAN9/TCP/dolphin(1425)

- *character_set* specifies the coded character set number used by the gateway when communicating with the WebSphere MQ queue manager. This is an optional parameter.

  This parameter is set only if the system that is running the WebSphere MQ queue manager uses a different encoding scheme than the system that runs the gateway. When set, the value of *character_set* is used by the WebSphere MQ client software on the gateway system to convert the data.

Refer to Chapter 8, "Gateway Running Environment" for more information on transaction and security models.

> **See Also:** Refer to *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* for additional information.

Configuring Oracle Net for the Gateway

The gateway requires Oracle Net to provide transparent data access to and from the Oracle database. Oracle Net uses the Oracle Net Listener to receive incoming connections from an Oracle Net client. In the case of the gateway, the Oracle Net Listener listens for incoming requests from the Oracle database. For the Oracle Net Listener to listen for the gateway, information about the gateway must be added to the Oracle Net Listener configuration file (`listener.ora`). This file is located in the `ORACLE_HOME`/network/admin directory on Microsoft Windows and `ORACLE_HOME`\network\admin directory on UNIX based systems by default, where `ORACLE_HOME` is the directory under which the gateway is installed. The default values in this file are set for you during the installation process by Oracle Universal Installer.

Using Oracle Net with Default Gateway Values

If you are configuring one gateway instance, and if you do not need to change any of the default values, then no further configuration is necessary for Oracle Net. Perform only "Step 2: Stop and Start the Oracle Net Listener for the Gateway" on page 7-6.

Using Oracle Net When Changing the Default Gateway Values

If you intend to use the Oracle Net listener for multiple gateway instances, or if you need to modify some of the default values set during the installation phase, then perform Step 1 and Step 2 in this section.

In Step 1, you add gateway information or change default information in the `listener.ora` file in the gateway directory `ORACLE_HOME`\network\admin on Microsoft Windows and `ORACLE_HOME`/network/admin on UNIX based systems.

Step 1: Configure the Oracle Net Oracle Net Listener for the Gateway

Two entries must be added to the `listener.ora` file:

- A list of Oracle Net addresses for the Oracle Net Listener to listen on

- The gateway process that the Oracle Net Listener should start in response to incoming connection requests

> **Note:** The Oracle Net Listener and the gateway must reside on the same node. If you already have a Oracle Net Listener running on the node, then you must make the changes suggested in Step 1 and 2 to your existing `listener.ora` and `tnsnames.ora` files.
>
> After making the changes, you can reload the changes by running the `reload` subcommand in the `lsnrctl` utility without shutting down the Oracle Net Listener.

Specifying Oracle Net Addresses for the Oracle Net Listener

If you are using Oracle Net and the TCP/IP protocol adapter, then the syntax of an entry in the `listener.ora` file is:

```
LISTENER=
  (ADDRESS_LIST=
      (ADDRESS=
          (PROTOCOL=TCP)
          (HOST=host_name)
          (PORT=port_number)
      )
```

```
)
```

In this entry:

- *host_name* is the name of the system where the gateway is installed.

- *port_number* specifies the IP port number used by the Oracle Net Listener. If you have other listeners running on *host_name*, such as the listener of an Oracle database on the same system, then the value of *port_number* must be different from the other listener port numbers.

If you are using Oracle Net and the interprocess socket call (IPC) protocol adapter, the syntax of the entry in listener.ora file is:

```
LISTENER=
  (ADDRESS_LIST=
    (ADDRESS=
        (PROTOCOL=IPC)
        (KEY=key_name)
    )
  )
```

In this entry:

- IPC specifies that the protocol used for connections is IPC.

- *key_name* is the unique user-defined service name.

Entry for the Gateway

To configure the Oracle Net Listener to listen for a gateway instance in incoming connection requests, add an entry to the listener.ora file using the following syntax:

```
SID_LIST_LISTENER=
  (SID_LIST=
    (SID_DESC=
        (SID_NAME=gateway_sid)
        (ORACLE_HOME=gateway_directory)
        (PROGRAM=driver)
    )
  )
```

In this entry:

- *gateway_sid* specifies the SID of the gateway and matches the gateway SID specified in the connect descriptor entry in the tnsnames.ora file. Refer to Configuring Oracle Net for Oracle Database on page 7-8.

- *gateway_directory* specifies the gateway directory in which the gateway software resides.

- *driver* is the name of the gateway executable file. If the gateway uses a local WebSphere MQ server, then the file name is dg4mqs. The file name is dg4mqc if the gateway is run as a WebSphere MQ client to access a remote WebSphere MQ server.

When you add an entry for multiple gateway instances, add the entry to the existing SID_LIST syntax:

```
SID_LIST_LISTENER=
 (SID_LIST=
     (SID_DESC=.
             .
             .
```

```
        )
        (SID_DESC=.
                  .
                  .
        )
        (SID_DESC=
            (SID_NAME=gateway_sid)
            (ORACLE_HOME=gateway_directory)
            (PROGRAM=driver)
         )
  )
```

The following are examples of entry made to the listener.ora file:

For Microsoft Windows:

```
(SID_DESC =
        (SID_NAME=dg4mqs)
        (ORACLE_HOME=gateway_directory)
        (PROGRAM=dg4mqs)
)
```

For UNIX based systems:

```
(SID_DESC =
        (SID_NAME=dg4mqs)
        (ORACLE_HOME=/oracle/app/oracle/product/dg4mq)
        (PROGRAM=dg4mqs)
)
```

> **See Also:** Refer to *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* for additional information about changing listener.ora.

Step 2: Stop and Start the Oracle Net Listener for the Gateway

The Oracle Net Listener must be started or reloaded to initiate the new settings.

> **Note:** If you already have a Oracle Net Listener running on the Oracle database where the gateway is installed, then you must change your existing listener.ora and tnsnames.ora files. After making the changes, you can reload the changes by running the reload subcommand in the lsnrctl utility without shutting down the Oracle Net Listener.
>
> Refer to the Note in Step 1: Configure the Oracle Net Oracle Net Listener for the Gateway.

- Set the gateway directory name:

  For Microsoft Windows:

  ```
  set TNS_ADMIN=c:\orant\network\admin
  ```

  If you are using the Bourne or Korn shell, then enter:

  ```
  $ ORACLE_HOME=gateway_directory;export ORACLE_HOME
  ```

  If you have the C shell, then enter:

```
$ setenv ORACLE_HOME gateway_directory
```

In this entry:

gateway_directory specifies the directory where the gateway software is installed.

- If the listener is already running, then use the `lsnrctl` command to reload the listener with the new settings:

For Microsoft Windows:

```
c:\orant\bin> lsnrctl reload your_listener_name
```

For UNIX based systems:

```
$ cd $ORACLE_HOME/bin
$ ./lsnrctl reload your_listener_name
```

In this entry:

ORACLE_HOME specifies the directory where the gateway software is installed.

- Check the status of the listener with the new settings:

For Microsoft Windows:

```
c:\orant\bin> lsnrctl status your_listener_name
```

For UNIX based systems:

```
$ ./lsnrctl status listener_name
```

The following are examples of the output from a `lsnrctl` status check:

For Microsoft Windows:

```
Connecting to (ADDRESS=(PROTOCOL=IPC)(KEY=ORAIPC))
STATUS of the LISTENER
-----------------------
Alias                   LISTENER
Version                 TNSLSNR for MS Windows: version 12.1.0.1.0 - Beta
Start Date              21-JUN-12 18:16:10
Uptime                  0 days 0 hr. 2 min. 19 sec
Trace Level             off
Security                OFF
SNMP                    OFF
Listener Parameter File \oracle\app\oracle\product\dg4mqs\network\admin\listener.ora
Listener Log File       \oracle\app\oracle\product\dg4mqs\network\log\listener.log
Services Summary...
  dg4mqs          has 1 service handler(s)
The command completed successfully
```

For UNIX based systems:

```
Connecting to (ADDRESS=(PROTOCOL=IPC)(KEY=ORAIPC))
STATUS of the LISTENER
-----------------------
Alias                   LISTENER
Version                 TNSLSNR for Solaris: version 12.1.0.1.0 - Production
Start Date              04-Jun-12 10:16:10
Uptime                  0 days 0 hr. 2 min. 19 sec
Trace Level             off
Security                OFF
SNMP                    OFF
```

```
Listener Parameter File   /oracle/app/oracle/product/dg4mqs/network/admin/listener.ora
Listener Log File         /oracle/app/oracle/product/dg4mqs/network/log/listener.log
Services Summary...
  dg4mqs            has 1 service handler(s)
The command completed successfully
```

In the example, `dg4mqs` is the default SID value that was assigned during installation. You can use any valid ID for the SID, or keep the default.

> **Note:** You must use the same SID value in the `tnsnames.ora` file, the `listener.ora` file, and the `GATEWAY_SID` environment variable in the gateway initialization file for each gateway instance being configured.

Configuring Oracle Net for Oracle Database

Any Oracle application that has access to an Oracle database can also access WebSphere MQ through the gateway. Before you use the gateway to access WebSphere MQ, you must configure the Oracle database so that it can communicate with the gateway by using Oracle Net. To configure the server, add connect descriptors to the `tnsnames.ora` file.

Any Oracle database that accesses the gateway needs a service name entry or a connect descriptor name entry in the `tnsnames.ora` file on the server, to tell the Oracle database how to make connections. This file, by default, is located in the *ORACLE_HOME*\network\admin directory on Microsoft Windows and ORACLE_HOME/network/admin directory on UNIX based systems, where ORACLE_HOME is the directory in which the Oracle database is installed. The `tnsnames.ora` file is required by the Oracle database that is accessing the gateway, and not by the gateway itself. Refer to "Configuration Overview" and to "Configuring the Gateway" on page 7-1.

> **See Also:** Refer to *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* for more information about changing the `tnsnames.ora` file.

Using Default Gateway Values

Oracle Universal Installer creates and preconfigures a `tnsnames.ora` file in the ORACLE_HOME\network/admin directory on Microsoft Windows and ORACLE_HOME/network/admin directory on UNIX based systems, where ORACLE_HOME is the directory in which the gateway software is installed. If you use the default values, and if you do not need to configure additional gateway instances, then you can append the contents of this file to the `tnsnames.ora` file of each Oracle database that accesses the gateway.

Changing Default Gateway Values

If you need to change some of the default settings, use the examples in this section to guide you.

TCP/IP Example

An Oracle database accesses the gateway using Oracle Net and the TCP/IP protocol adapter. The syntax of the connect descriptor entry in `tnsnames.ora` is:

```
tns_name_entry=
   (DESCRIPTION=
```

```
(ADDRESS=
    (PROTOCOL=TCP)
    (HOST=host_name)
    (PORT=port_number)
)
(CONNECT_DATA=
(SID=gateway_sid)
)
(HS=OK)
)
```

In this example:

- *tns_name_entry* is the *tns_name_entry* of the CREATE DATABASE LINK statement. Refer to "Creating Database Links" on page 7-10 for more information.

- TCP specifies that the protocol used for connections is TCP/IP.

- *port_number* is the port number used by the Oracle Net Oracle Net Listener that listens for the gateway. This port number can be found in the listener.ora file that is used by the Oracle Net Listener. Refer to "Specifying Oracle Net Addresses for the Oracle Net Listener" on page 7-4.

- *host_name* specifies the system on which the gateway is running. The Oracle Net Listener host name can be found in the listener.ora file used by the Oracle Net Listener that is listening for the gateway. Refer to "Specifying Oracle Net Addresses for the Oracle Net Listener" on page 7-4.

- *gateway_sid* specifies the SID of the gateway and matches the SID specified in the listener.ora file of the Oracle Net Listener that listens for the gateway.

IPC Example

An Oracle database accesses the gateway using Oracle Net and the IPC protocol adapter. The syntax of the connect descriptor entry in tnsnames.ora is:

```
tns_name_entry=
    (DESCRIPTION=
        (ADDRESS=
        (PROTOCOL=IPC)
        (KEY=key_name)
    )
    (CONNECT_DATA=
        (SID=gateway_sid)
    )
      (HS=OK)
  )
```

where:

- *tns_name_entry* is the *tns_name_entry* of the CREATE DATABASE LINK statement. Refer to "Creating Database Links" on page 7-10 for more information.

- IPC specifies that the protocol used for connections is IPC.

- *key_name* is the service name.

- *gateway_sid* specifies the SID of the gateway and matches the SID specified in the listener.ora file of the Oracle Net Listener that is listening for the gateway.

Creating a Transaction Log Queue

When the `TRANSACTION_MODEL` parameter in the gateway initialization file is set to `COMMIT_CONFIRM` to allow for distributed transactions, then an additional configuration step is required to:

- Create a WebSphere MQ queue

- Set the `TRANSACTION_LOG_QUEUE`, `TRANSACTION_RECOVERY_USER` and `TRANSACTION_ RECOVERY_PASSWORD` parameters in the gateway initialization file

Refer to "Commit-Confirm Transactions" on page 8-5 for more information about the commit-confirm transaction model and Appendix C, "Gateway Initialization Parameters" for information about `TRANSACTION_LOG_QUEUE`, `TRANSACTION_RECOVERY_ USER`, and `TRANSACTION_RECOVERY_PASSWORD`.

> **See Also:** Refer to IBM publications for information about creating and configuring a queue.

For the gateway to recover distributed transactions, a recovery account and queue must be set up in the queue manager by the WebSphere MQ system administrator. This account must be a valid WebSphere MQ user, and it must have authorization to access the recovery queue. Refer to "Authorization for WebSphere MQ Objects" on page 8-2 for more information about access privileges.

The gateway uses the recovery queue to check the status of failed transactions that were started at the queue manager by the gateway and were logged in this queue. The information in this queue is vital to the recovery process and must not be used, accessed, or updated except by the gateway.

Administering the Database Links Alias Library

A connection to the gateway is established through a database link when it is first used in an Oracle session. In this context, *connection* refers to the connection between the Oracle database and the gateway. The connection persists until the Oracle session ends. Another session or user can access the same database link and get a distinct connection to the gateway and the queue manager.

Database links are active for the duration of a gateway session. To close a database link during a session, use the `ALTER SESSION` statement.

> **See Also:** For more information about using database links, refer to the *Oracle Database Administrator's Guide*.

Using Database Links

An alias library is a schema object that represents a library in PL/SQL. Oracle Database Gateway for WebSphere MQ uses an alias library to access the shared library installed with Oracle Database Gateway for WebSphere MQ. To create the alias library, you must have the `CREATE LIBRARY PRIVILEGE`. The alias library used by Oracle Database Gateway for WebSphere MQ is `libdg4mq` and is defined in the `pgmobj.sql` script, which is created when the Oracle Database Gateway for WebSphere MQ deployment scripts are executed.

Creating Database Links

To create a database link, use the `CREATE DATABASE LINK` statement. The `USING` clause points to a connect descriptor in the `tnsnames.ora` file. The `CONNECT TO` clause

specifies the WebSphere MQ user ID and password when the security model is defined as `STRICT` with the `AUTHORIZATION_MODEL` parameter. If you do not include the `CONNECT TO` clause, then the current user ID and password are used.

When the `AUTHORIZATION_MODEL` parameter is set to `RELAXED`, you need not specify an user ID and password because the Oracle database uses the user ID and password of the user account that started the Oracle Net Listener for the gateway. If you specify an user ID and password with the `CONNECT TO` clause, then the Oracle database and gateway ignore those values. Refer to "Security Models" on page 8-1 for more information.

The syntax of `CREATE DATABASE LINK` is as follows:

```
CREATE [PUBLIC] DATABASE LINK dblink [CONNECT TO userid IDENTIFIED
             BY password] USING 'tns_name_entry';
```

where:

- *dblink* is the database link name.

- *userid* is the user ID used to establish a session at the queue manager. It is only used when `AUTHORIZATION_MODEL` is set to `STRICT` in the init*sid*.ora file. The user ID must be authorized to access all WebSphere MQ objects, and use any database object referenced in the PL/SQL commands.

  The *userid* must be in the password file on the computer on which WebSphere MQ and the gateway are installed. Otherwise, the userid must be published in the UNIX Network Information Service (NIS) when WebSphere MQ and the gateway are installed on different systems. If userid contains lowercase letters or non-alphanumeric characters, then you must surround userid with quotation marks ("). Refer to your WebSphere MQ documentation for more information about userid.

- *password* is the password used to establish a session at the queue manager. It is used only when `AUTHORIZATION_MODEL` is set to `STRICT` in the init*sid*.ora file.

  The password must be in the password file on the system on which WebSphere MQ and the gateway are installed. Otherwise, the password must be published in the Windows or UNIX Network Information Service (NIS), as the case may be when WebSphere MQ and the gateway are installed on different systems.

  If password contains lowercase letters or non alphanumeric characters, then surround password with quotation marks (").

- *tns_name_entry* is the Oracle Net TNS connect descriptor name specified in the tnsnames.ora file.

Dropping Database Links

You can drop a database link with the `DROP DATABASE LINK` statement. For example, to drop the database link named dblink, enter:

```
DROP [PUBLIC] DATABASE LINK dblink;
```

A database link should not be dropped if it is required to resolve a distributed transaction that is in doubt.

> **See Also:** Refer to the *Oracle Database Administrator's Guide* for more information about dropping database links.

Examining Available Database Links

The data dictionary of each database stores the definitions of all the database links in that database. The USER_DB_LINKS view shows the database links that are defined for a user. The ALL_DB_LINKS data dictionary views show all the defined database links.

Limiting the Number of Active Database Links

You can limit the number of connections from a user process to remote databases with the OPEN_LINKS parameter. This parameter controls the number of remote connections that any single user process can use with a single user session.

> **See Also:** Refer to the *Oracle Database Administrator's Guide* for more information about limiting the number of active database links.

Creating Alias Library

Create the Oracle Database Gateway for WebSphere MQ alias library, libdg4mq, using the Oracle Database Gateway for WebSphere MQ deployment scripts. During installation, the appropriate shared library name is defined in ORACLE_HOME\dg4mq\admin\deploy\pgmobj.sql on Microsoft Windows and *ORACLE_HOME*/dg4mq/admin/deploy/pgmobj.sql on UXIX based systems based on the DG4MQ model you choose.

For a remote model, the libdg4mqc.so shared library is used. For example:

```
CREATE OR REPLACE LIBRARY libdg4mq AS 'ORACLE_HOME/lib/libdg4mqc.so'
TRANSACTIONAL;
```

For a local model, the libdg4mqs.so shared library is used. For example:

```
CREATE OR REPLACE LIBRARY libdg4mq AS 'ORACLE_HOME/lib/libdg4mqs.so'
TRANSACTIONAL;
```

> **Note:** The file extension of shared libraries on HP-UX is .sl. For example, libdg4mqc.sl

Dropping Alias Library

Use the undeploy scripts to drop the libdg4mq Oracle Database Gateway for WebSphere MQ alias library.

Installing the Oracle Visual Workbench Repository

Install the Oracle Visual Workbench repository following the steps in this section.

You can skip the installation of the Oracle Visual Workbench repository if you do not plan to use Oracle Visual Workbench, or if you are preparing your production Oracle database, where you do not need a Visual Workbench repository, but instead need a Oracle Database Gateway for WebSphere MQ deployment. Refer to the "Preparing the Production Oracle Database" on page 7-16 for details.

Preinstallation Tasks

The following steps describe the preinstallation tasks.

Step 1: Choose a Repository Server

A repository server is an Oracle database on which the Visual Workbench repository is installed.

Step 2: Locate the Installation Scripts

The Visual Workbench repository installation scripts are installed with the Visual Workbench. If the repository is to be installed on the same system as Oracle Visual Workbench, then your repository server already has all the required installation scripts. Proceed to Step 3.

1. Create a directory on the repository server to be the script directory. For example:

For Microsoft Windows:

```
> md %ORACLE_HOME%\dg4mqadmin\repo
```

For UNIX based systems:

```
$ mkdir $ORACLE_HOME/dg4mq/admin/repo
$ chmod 777 $ORACLE_HOME/dg4mq/admin/repo
```

2. Use a file transfer program to transfer the repository zip file (repos*XXX*.zip, where *XXX* is the release number), or move all script files with the .sql suffix from the script file directory (ORACLE_HOME\dg4mqvwb\server\admin on Windows) on the Visual Workbench system to the script file directory on the repository server system.

Step 3: Upgrade the Visual Workbench Repository

Upgrade your existing Visual Workbench repository installation scripts by copying the pgm*xxx*.sql files installed with the Oracle Database Gateway for WebSphere MQ in the ORACLE_HOME\dg4mq\admin\deploy directory on Microsoft Windows and ORACLE_HOME/dg4mq/admin/deploy directory on UNIX based systems to the script file directory on the repository server system.

Step 4: Ensure that the UTL_RAW Package is Installed

All data mapping packages generated by the Visual Workbench use the UTL_RAW package, which provides routines for manipulating raw data.

From SQL*Plus, as the SYS user, issue the following statement:

```
SQL> DESCRIBE UTL_RAW
```

If the DESCRIBE statement is successful, then your repository server already has UTL_RAW installed, and you can proceed to Step 4.

If the DESCRIBE statement fails, then install UTL_RAW:

From SQL*Plus, as the SYS user, run the utlraw.sql and prvtrawb.plb scripts that are in the ORACLE_HOME\rdbms\admin directory on Microsoft Windows and ORACLE_HOME/rdbms/admin directory on UNIX based systems. You must run the utlraw.sql script first.

```
SQL> @utlraw.sql
SQL> @prvtrawb.plb
```

Step 5: Ensure that the DBMS_OUTPUT Package is Enabled

The sample programs and installation verification programs on the distribution CD-ROM use the standard DBMS_OUTPUT package.

From SQL*Plus, as `SYS` user, issue the following statement:

```
SQL> DESCRIBE DBMS_OUTPUT
```

If the `DESCRIBE` statement is successful, then your repository server has `DBMS_OUTPUT` installed, and you can proceed to Step 6.

If the `DESCRIBE` statement fails, then install `DBMS_OUTPUT`.

> **See Also:** Refer to *Oracle Database Administrator's Guide* for more information.

Step 6: Create a Database Link

Create a database link on your Oracle Production System Server to access the Oracle Database Gateway for WebSphere MQ.

If you do not already have a database link, then refer to "Administering the Database Links Alias Library" on page 7-10 for more information about creating database links.

Visual Workbench Repository Installation Tasks

Use `pgvwbrepos.sql` to install the Visual Workbench Repository on Oracle10*g*. To run `pgvwbrepos.sql`, ensure that you are currently in the `ORACLE_HOME\dg4mq\admin\repo` directory on Microsoft Windows and *ORACLE_HOME*/dg4mq/admin/repo directory on UNIX based systems, and then enter the following command:

```
sqlplus /nolog @pgvwbrepos.sql
```

> **Note:** If you are installing the Visual Workbench repository on Oracle8*i* or older, then you must use `pgvwbrepos8.sql`. All of the examples in this section are provided with the assumption that you are installing on Oracle9*i* and later.

The script takes you through the following steps:

Step 1: Enter the Database Connection Information

Use the default vale of `LOCAL` by pressing **Enter**. Next, you are prompted to enter the passwords for the `SYSTEM` and `SYS` accounts of the Oracle database. Press **Enter** after entering each password.

The script stops if any information is incorrect. Verify the information before rerunning the script.

Step 2: Check for Existing Workbench Repository

The script checks for an existing Visual Workbench repository and for the data dictionary. If neither is found, then the script proceeds to Step 3 below.

If the data dictionary exists, then the script stops. Choose another Oracle database and rerun the script, starting at "Step 1: Choose a Repository Server" on page 7-13.

If a Visual Workbench repository exists, then the script gives you the following options:

- Upgrade the existing private repository to public status and proceed to Step 3.

- Replace the existing repository with the new private repository and proceed to Step 3.

- Stop the script.

Step 3: Check for The Required PL/SQL Packages

The script checks for the existence of `UTL_RAW`, `DBMS_OUTPUT`, and `DBMS_PIPE` in the Oracle database. If this software exists, then the script proceeds to Step 4.

The script stops if this software does not exist. Refer to *Oracle Database Administrator's Guide* about the missing software. After the software is installed, rerun the script.

Step 4: Install the UTL_PG Package

The script checks for the existence of the `UTL_PG` package. If it does not exist, then the `UTL_PG` package is installed. The script then proceeds to Step 5.

If `UTL_PG` exists, then you are prompted to reinstall it. Press Return to reinstall `UTL_PG`.

Step 5: Create the Administrative User and All Repository Tables

This step creates the administrative user for the Visual Workbench repository as `PGMADMIN` with the initial password of `PGMADMIN`. This user owns all objects in the repository.

After this step, a private Visual Workbench repository that includes the `PGM_SUP`, `PGM_BQM`, and `PGM_UTL8` packages, is created in the Oracle database, which only the `PGMADMIN` user can access.

Step 6: Create Public Synonyms and Development Roles

This is an optional step to change the private access privileges of the Visual Workbench repository. The private status enables only the `PGMADMIN` user to have access to the repository. If you enter `N` and press **Enter**, then the repository retains its private status.

A public status enables the granting of access privileges to other users besides `PGMADMIN`. If you want to give the repository public status, then enter **Y** and press **Enter**.

After the Repository is Created

After creating the Visual Workbench repository, there is one optional step, granting development privileges for the Visual Workbench repository to users.

To allow users, other than the `PGMADMIN` user, to perform development operations on the Visual Workbench repository, `PGMADMIN` must grant them the necessary privileges. To do this, perform the following:

- Ensure that the repository has a public status. It has this status if you created it by using Steps 1 to 6 of the `pgvwbrepos.sql` script. If you did not use Step 6, then rerun the script. When you get to Step 2 of the script, enter **A** at the prompt to upgrade the private repository to public status.

- Use SQL*Plus to connect to the repository as the `PGMADMIN` user and grant the `PGMDEV` role to each user. For example:

```
SQL> GRANT PGMDEV TO SCOTT;
```

Deinstall the Visual Workbench Repository

To deinstall a Visual Workbench repository on Oracle10*g*, use the repository script `pgvwbremove.sql`. To run this script, ensure that you are currently under the Oracle database `ORACLE_HOME\dg4mq\admin\repo` directory on Microsoft Windows and *ORACLE_HOME*/dg4mq/admin/repo directory on UNIX based systems (where you copied the scripts), and then enter the following command:

```
sqlplus /nolog @pgvwbremove.sql
```

> **Note:** If you are deinstalling the Visual Workbench Repository on Oracle8*i* or earlier, then you must use `pgvwbremove8.sql`. All the examples in this section are provided with the assumption that you are installing on Oracle9*i* and later.

The script takes you through the following steps:

Step 1: Enter the Database Connection Information

Use the default value of `LOCAL` by pressing **Enter**.

Next, you are prompted to enter the passwords for the `SYSTEM`, `SYS`, and `PGMADMIN` accounts of the Oracle database. Press **Enter** after entering each password.

The script stops if any of the information is incorrect. Verify the information before rerunning the script.

Step 2: Check for the Existing Workbench Repository

Enter **Y** and press **Enter** for the prompt to remove public synonyms and development roles. This returns the repository to private status. You can exit the script now by entering **N** and pressing **Enter**, or you can proceed to the next prompt.

If you are certain you want to remove the private repository, then enter **Y** and press **Enter**. The script removes all repository tables and related packages.

Preparing the Production Oracle Database

These preparations include preparing, installing, and removing PL/SQL packages on the production database.

Introduction

Before you can compile MIPs on a production Oracle database, the following PL/SQL packages must be present on the production Oracle database:

- `DBMS_PIPE`, `DBMS_OUTPUT`, and `UTL_RAW`

  These packages are shipped with each Oracle database and are typically preinstalled.

- `PGM`, `PGM_BQM`, `PGM_SUP`, and `UTL_PG`

  These packages are shipped with your Oracle Database Gateway for WebSphere MQ. They are installed during the creation process of the Visual Workbench repository. Do not execute deployment script on the Oracle database with an installed Visual Workbench repository. If the Oracle database used for the

repository is different from the Oracle database used in the production environment, you must install these packages on the production Oracle database.

This section describes how to run the following scripts:

- `pgmdeploy.sql`,

   A deployment script that is used to verify the existence of the required PL/SQL packages and install them if they do not exist on the production Oracle database.

- `pgmundeploy.sql`

   A script to remove the PL/SQL packages from a production Oracle database.

Verifying and Installing PL/SQL Packages

1. Locate the following scripts:

   - `pgm.sql`

   - `pgmbqm.sql`

   - `pgmdeploy.sql`

   - `pgmsup.sql`

   - `pgmundeploy.sql`

   - `prvtpg.sql`

   - `utlpg.sql`

   These scripts are installed with the gateway, in the `ORACLE_HOME\dg4mq\admin\deploy` directory on Microsoft Windows and `ORACLE_HOME/dg4mq/admin/deploy` directory on UNIX based systems, where `ORACLE_HOME` is the gateway home directory.

2. If your production Oracle database is on a system that is different from the gateway, then use a file transfer method, such as FTP, to transfer files in the `ORACLE_HOME\dg4mq\admin\deploy` directory on Microsoft Windows and `ORACLE_HOME/dg4mq/admin/deploy` directory on UNIX based systems, where `ORACLE_HOME` is the gateway home directory on your gateway system. On your production Oracle database system, change directory to the directory containing the deployment scripts that you just transferred and skip to Step 4.

3. If your production Oracle database is on the same system as the gateway, then change the directory to `ORACLE_HOME\dg4mq\admin\deploy` directory on Microsoft Windows and `ORACLE_HOME/dg4mq/admin/deploy` on UNIX based systems, where `ORACLE_HOME` is the gateway home directory.

4. Run the `pgmdeploy.sql` script by as follows:

   ```
   $ sqlplus /nolog @pgmdeploy.sql
   ```

5. At the script prompt: Enter the connect string for the Oracle database... [LOCAL], press **Enter** to use the default value of `LOCAL`.

6. At the script prompt Enter the following required Oracle database password, enter the password of the `SYS` account.

After the script verifies the `SYS` account password, it connects to the production Oracle database. The script verifies and reports the PL/SQL packages that are installed there:

- If any of the Oracle database packages, DBMS_OUTPUT, DBMS_PIPE or UTL_RAW are missing, then the script stops. Have your DBA install the missing packages and re-run the deployment script.

- If any of the Oracle packages, PGM, PGM_BQM, PGM_SUP, and UTL_PG are missing, then the script installs them on the production Oracle database.

Removing the PL/SQL Packages

You can remove the PL/SQL packages that were installed by the pgmdeploy.sql script if, for example, none of your applications in the production environment uses a MIP. To remove these packages, perform the following steps:

1. On your production Oracle database, change to the directory containing the deployment scripts by entering the following command:

   For Microsoft Windows:

   ```
   > cd ORACLE_HOME\dg4mq\admin\deploy
   ```

   For UNIX based systems:

   ```
   $ cd $ORACLE_HOME/dg4mq/admin/deploy
   ```

2. Run the pgmundeploy.sql as follows:

   ```
   $ sqlplus /nolog @pgmundeploy.sql
   ```

3. At the script prompt: Enter the connect string for the Oracle database... [LOCAL], press [Return] to use the default of LOCAL.

4. At the script prompt, enter the required Oracle database passwords, enter the password of the SYS account.

After the script verifies the SYS account password, it connects to the production Oracle database and removes the packages installed by the pgmdeploy.sql script.

After the pgmundeploy.sql script completes successfully, applications on the production Oracle database fail if they attempt to reference any of the MIPs that are compiled there.

8

Gateway Running Environment

This chapter describes the gateway running environment. It contains the following sections:

- Security Models on page 8-1
- Transaction Support on page 8-3
- Troubleshooting on page 8-5

Security Models

WebSphere MQ has its own authorization mechanism. Applications are allowed to perform certain operations on queues or queue managers only when their effective user ID has authorization for each operation. The effective user ID, typically the operating system user, depends on the WebSphere MQ environment and the platform it runs on.

The effective user ID in an Oracle environment is not dependent on an operating system account or the platform. Because of this difference, the gateway provides two authorization models for Oracle applications to work with WebSphere MQ:

- Relaxed
- Strict

Although Oracle and operating system user IDs can be longer than 12 characters, the length of user IDs used for either model cannot exceed 12 characters. Oracle user accounts do not have a minimum number of characters required for their passwords, but some platforms and operating systems do. Take their requirements into consideration when deciding on a password or user ID.

The authorization model is configured with the `AUTHORIZATION_MODEL` parameter in the gateway initialization file. Refer to Appendix C, "Gateway Initialization Parameters" for more information about the `AUTHORIZATION_MODEL` parameter.

Relaxed Model

This model discards the Oracle user name and password. The authorizations granted to the effective user ID of the gateway by the queue manager are the only associations an Oracle application has. For example, if the gateway user ID is granted permission to open or read messages, or place messages on a queue, then all Oracle applications that access the gateway can request those operations.

The effective user ID is determined by how the gateway runs:

- If the gateway runs as an MQI client application, then the user ID is determined by the MQI channel definition.

  > **See Also:**   Refer to IBM publications for more information about channel definitions

- If the gateway runs as an MQI server application, then the effective user ID of the gateway is the user account that started the Oracle Net listener and has authorization to all the WebSphere MQ objects that the Oracle application wants to access. Refer to "Authorization for WebSphere MQ Objects" on page 8-2 for more information.

Oracle recommends using the relaxed model only if your application has minimal security requirements.

Strict Model

This model uses the Oracle user ID and password provided in the CREATE DATABASE LINK statement when a database link is created, or the current Oracle user ID and password if none was provided with CREATE DATABASE LINK.

The Oracle user ID:

- Must match a user account for the system that runs the gateway and for the system that runs the WebSphere MQ queue manager
- Must have authorization for all the accessed WebSphere MQ objects. Refer to "Authorization for WebSphere MQ Objects" on page 8-2 for more information.

The authorization process to verify the Oracle user ID and password varies, depending on how the gateway runs.

Authorization Process for a WebSphere MQ Server Application

If the gateway runs as a WebSphere MQ server application, then the authorization process checks the user ID and password against the local or network password file. If they match, then the gateway performs a SET-UID for the user ID and continues to run under this user ID. Further WebSphere MQ authorization checks happen for this user ID.

Authorization Process for a WebSphere MQ Client Application

If the gateway runs as a WebSphere MQ client application, then the authorization process checks the user ID and password against the local or network password file. If they match, then the MQ_USER_ID and MQ_PASSWORD WebSphere MQ environment variables are set to the values of the user ID and password. If the channel definition specifies the MCAUSER WebSphere MQ environment variable as blank characters, then WebSphere MQ authorization checks are performed for the user ID.

If MCAUSER is set, not set, or security exits are defined for the MQI channel, then these override the gateway efforts.

> **See Also:**   Refer to IBM publications for more information about WebSphere MQ environment variables.

Authorization for WebSphere MQ Objects

The effective user ID for the relaxed model and the Oracle user ID for the strict model require the WebSphere MQ authorizations described in Table 8–1.

Table 8–1   WebSphere MQ Access Authorization

Type of Access	WebSphere MQ Authorization Keywords	Alternate WebSphere MQ Authorization Keywords
Permission to access the WebSphere MQ queue manager	`all` or `allmqi`	`connect` `setid`
Permission to send messages to a WebSphere MQ queue	`all` or `allmqi`	`passall` `passid` `put` `setid`
Permission to receive messages from a WebSphere MQ queue	`all` or `allmqi`	`browse` `get` `passall` `passid` `setid`

> **See Also:**   Refer to IBM publications for more information about WebSphere MQ authorizations.

Transaction Support

Transactions from an Oracle application that use the gateway and invoke WebSphere MQ message queue operations are managed by the transaction coordinator at the Oracle database where the transaction originates.

Non-Oracle Data Sources and Distributed Transactions

When an Oracle distributed database contains a gateway, the gateway must be properly configured to take part in a distributed transaction. The outcome of a distributed transaction involving a gateway should be that all participating sites roll back or commit their parts of the distributed transaction. All participating sites, including gateway sites, that are updated during a distributed transaction must be protected against failure and must be able to take part in the two-phase commit mechanism.

A gateway that updates a target system as part of a distributed transaction must be able to take part in the automatic recovery mechanism, which might require that recovery information be recorded in transaction memory at the target system.

If a SQL-based gateway is involved in a distributed transaction, the distributed database must be in a consistent state after the distributed transaction is committed.

A database gateway or a SQL-based gateway with the procedural option translates remote procedure calls into target system calls. From the viewpoint of the Oracle transaction model, the gateway is like an Oracle database executing a PL/SQL block containing SQL statements that are used to access an Oracle database.

For a database gateway, it is unknown if a target system call alters data. To ensure the consistency of a distributed database, it must be assumed that a database gateway updates the target system. Accordingly, all remote procedure calls sent to a database gateway take part in a distributed transaction and must be protected by the two-phase commit protocol. For example, you could issue the following SQL*Plus statements:

```
EXECUTE REMOTE_PROC@FACTORY;
```

```
INSERT INTO DEBIT@FINANCE
ROLLBACK;
```

In this example, REMOTE_PROC is a remote procedure call to access a database gateway, DEBIT is an Oracle table residing in an Oracle database, and FACTORY and FINANCE are database links used to access the remote sites.

Transaction Capability Types

When gateways are involved in a distributed transaction, the transaction capabilities of the non-Oracle data source determine whether the data source can participate in two-phase commit operations or distributed transactions.

Depending on the capabilities of the non-Oracle data source, transactions can be classified as one of the following types:

Type	Description
Read-only	During a distributed transaction, the gateway provides read-only access to the data source, so the gateway can only be queried. A Read-only is used for target systems that use the presumed-commit model or do not support rollback mechanisms.
Single-site	During a distributed transaction, the target system is either read-only (other sites can be updated) or the only site updated (can participate in remote transactions). Single-site is used for target systems that support rollback, commit, and presumed-abort, but cannot prepare or commit-confirm as they have no distributed transaction memory, the ability to remember what happened during and after a distributed transaction.
Commit-confirm	The gateway is a partial partner in the Oracle transaction mode. During a distributed transaction in which it is updated, the gateway must be the commit point site. Commit-confirm is used for target systems that support rollback, commit, presumed-abort, and commit-confirm, but do not support prepare. The commit-confirm capability requires distributed transaction memory.
Two-phase commit	The gateway is a partial partner in the Oracle transaction model. During a distributed transaction, the gateway cannot be the commit point site.
	Two-phase commit is used for target systems that support rollback, commit, presumed-abort, and prepare, but do not support commit-confirm, because they have no distributed transaction memory.
Two-phase commit-commit confirm	The gateway is a full partner in the Oracle transaction model. During a distributed transaction, the gateway can be the commit point site, depending on the commit point strength defined in the gateway initialization file.
	This transaction type is used for target systems that support a full two-phased commit transaction model. That is, the target system supports rollback, commit, presumed-abort, prepare, and commit-confirm.

Transaction Capability Types of Oracle Database Gateway for WebSphere MQ

Transactions from an Oracle application (that invoke WebSphere MQ message queue operations and that are using the gateway) are managed by the Oracle transaction coordinator at the Oracle database where the transaction originates. The Oracle Database Gateway for WebSphere MQ provides the following transaction types:

- Single-site

■ Commit-confirm

Single-Site Transactions

Single-site transactions are supported for all WebSphere MQ environments and platforms. Single-Site means that the gateway can participate in a transaction only when queues belonging to the same WebSphere MQ queue manager are updated. An Oracle application can select, but not update, data on any Oracle database within the same transaction that sends to, or receives a message from, a WebSphere MQ queue. To update objects in the Oracle database, the transaction involving the WebSphere MQ queue should first be committed or rolled back.

This default mode of the gateway is implemented using WebSphere MQ single-phase, where the queue manager acts as the synchronizing point coordinator.

Commit-Confirm Transactions

Commit-Confirm transactions are enhanced forms of single-site transactions and are supported for all WebSphere MQ environments and platforms. Commit-confirm means that the gateway can participate in transactions when queues belonging to the same WebSphere MQ queue manager are updated and, at the same time, any number of Oracle databases are updated. Only one gateway with the commit-confirm model can join the distributed transaction because the gateway operates as the focal point of the transaction. To apply changes to queues of more than one queue manager, updates applied to one queue manager need to be committed before a new transaction is started for the next queue manager.

As with single-site transactions, commit-confirm transactions are implemented using WebSphere MQ single-phase, but it requires a dedicated recovery queue at the queue manager to log the transaction ID. At commit time, the gateway places a message in this queue with the message ID set to the Oracle transaction ID. After the gateway calls the queue manager to commit the transaction, the extra message on the transaction log queue becomes part of the overall transaction. This makes it possible to determine the outcome of the transaction in case of system failure, allowing the gateway to recover a failed transaction. When a transaction completes successfully, the gateway removes the associated message from the queue.

The WebSphere MQ administrator must create a reserved queue at the queue manager. The name of this queue is specified in the gateway initialization file with the `TRANSACTION_LOG_QUEUE` parameter. All Oracle users that access WebSphere MQ through the gateway should have full authorization for this queue. The transaction log queue is reserved for transaction logging only and must not be used, accessed, or updated other than by the gateway. When a system failure occurs, the Oracle recovery process checks the transaction log queue to determine the recovery strategy.

Two gateway initialization parameters, `TRANSACTION_RECOVERY_USER` and `TRANSACTION_RECOVERY_PASSWORD`, are set in the gateway initialization file to specify the user ID and password for recovery purposes. When set, the gateway uses this user ID and password combination for recovery work. The recovery user ID should have full authorization for the transaction log queue.

Refer to Appendix C, "Gateway Initialization Parameters" for more information about configuring the gateway for commit-confirm transactions.

Troubleshooting

This section includes information about messages, error codes, gateway tracing, and gateway operations.

Message and Error Code Processing

The gateway architecture includes a number of components. Any of these components can detect and report an error condition while processing PL/SQL code. An error condition can be complex, involving error codes and supporting data from multiple components. In all cases, the Oracle application receives a single Oracle error code on which to act.

Error conditions are represented in the following ways:

- Errors from the Oracle database

  Messages from the Oracle database are in the format ORA-*xxxxx* or PLS-*xxxxx*, where *xxxxx* is a code number. ORA-*xxxxx* is followed by text explaining the error code. Refer to the *Oracle Database Error Messages* for explanations of these errors.

  For example:

  ```
  PLS-00306: wrong number or types of arguments in call to 'MQOPEN'
  ORA-06550: line7, column 3:
  PL/SQL: Statement ignored
  ```

- Gateway and WebSphere MQ errors

  When possible, a WebSphere MQ error code is converted to an Oracle error code. If that is not possible, then the Oracle error ORA-29400 with the corresponding WebSphere MQ error code is returned. Refer to "Common Error Codes" on page 8-6 for more information.

  For Example:

  ```
  ORA-29400: data cartridge error
  MQI MQCONNX failed. completion code=2, reason code=2058
  ```

  > **Note:** Because the Oracle database distinguishes only between a successful or failed outcome of all user operations, MQI calls that return a warning are reported as a successful operation.

Interpreting Gateway Messages

Error codes are generally accompanied by additional message text, beyond the text associated with the Oracle message number. The additional text includes details about the error.

Gateway messages have the following format:

```
ORA-nnnnn:error_message_text
gateway_message_line
```

where:

- nnnn is an Oracle error number.

- *error_message_text* is the text of the message associated with the error.

- *gateway_message_line* is additional message text generated by the gateway.

Common Error Codes

The error conditions that are described in this section are common error conditions that an application might receive while using the gateway. However, do not cover all error situations.

ORA-01017: invalid username/password; logon denied

    Cause: Invalid username or password

    Action: Logon denied

ORA-29400: The MQSeries MQI call "call_name" fails with reason code mqi_code

    Cause: An MQI call to a WebSphere MQ queue manager failed. The gateway could not complete the current operation.

    Action: If *call_name* is MQOPEN and *mqi_code* is 2035, then do the following:

- If the gateway is configured for the relaxed security model, then use the WebSphere MQ administrative command interface to grant sufficient message privileges to the user account that started the Oracle Net listener. These privileges allow the user to send and receive messages for the specified WebSphere MQ queue. Refer to IBM publications for more information.

- If the gateway is configured for the strict security model, use the WebSphere MQ administrative command interface to grant message privileges to the user name specified in the CREATE DATABASE LINK statement. If no user name was specified in the CREATE DATABASE LINK statement, the privileges are granted to the current Oracle user ID. These privileges enable the user to send and receive messages for the specified WebSphere MQ queue. Refer to IBM publications for more information.

    If *call_name* is MQOPEN, and if *mqi_code* is 2085, then verify that the queue that is specified in the WebSphere MQ profile exists at the WebSphere MQ queue manager that you are trying to access and that the queue name is correctly spelled and in the correct letter case.

> **See Also:** Refer to IBM publications for more information on mqi_codes other than 2035 and 2085..

Gateway Tracing

The gateway has a trace feature for testing and debugging purposes. The trace feature collects information about the gateway running environment, MQI calls, and parameter values of the MQI calls. The amount of trace data to collect is based on the tracing level selected with the TRACE_LEVEL parameter. Refer to Appendix C, "Gateway Initialization Parameters" for more information about enabling tracing.

> **Note:** Do not enable tracing when your application is running in a production environment because it reduces gateway performance.

The trace data is written to the directory and file specified by the LOG_DESTINATION parameter.

LOG_DESTINATION Parameter

This is a gateway initialization parameter.

Gateway: SQL-based and procedural

Default Value: The default value is SID_agt_PID.trc.

Range of Values: None

Syntax: `LOG_DESTINATION = log_file`

Parameter Description

`LOG_DESTINATION = log_file`

`LOG_DESTINATION` specifies the file name or directory where the gateway writes logging information. When `log_file` already exists, logging information is written to the end of file.

If you do not specify `LOG_DESTINATION`, then the default log file is created each time that the gateway starts up.

Verifying Gateway Operation

If your application cannot connect to the gateway, then rerun the application with the gateway trace feature enabled. If no trace information is written to the log file specified by `LOG_DESTINATION`, or if the log file is not created at all, then verify that:

- The Oracle Net configuration for the gateway and the Oracle database is set up properly (refer to "Configuring Oracle Net for the Gateway" on page 7-4)

- A database link exists between the Oracle database and the gateway was created ("Administering the Database Links Alias Library" on page 7-10)

If the Oracle Net configuration and database link are set up correctly, then check the operation of the gateway with the `test.sql` script:

1. Change directory to the gateway sample directory by entering:

   For Microsoft Windows:

   ```
   > cd %ORACLE_HOME%\dg4mq\sample
   ```

   For UNIX based systems:

   ```
   $ cd $ORACLE_HOME/dg4mq/sample
   ```

2. Using an editor, modify the `test.sql` script as follows:

   a. Specify the database link name that you created for the gateway. To do this, replace the characters `@dg4mq` with `@dblink`, where `dblink` is the name you chose when the database link was created.

   b. Replace the characters `YOUR_QUEUE_NAME` with a valid WebSphere MQ queue name.

3. Using SQL*Plus, connect to your Oracle database as a valid user.

4. Run `test.sql`, a script that sends and retrieves a message from a WebSphere MQ queue. A successful completion displays the following output:

   ```
   SQL> @test.sql
   message put on queue = 10203040506070809000
   MQPUT: CorrelId length = 24
   MQPUT: MsgId length = 24
   MQPUT returned with reason code 0
   MQGET returned with reason code 0
   message read back = 10203040506070809000
   ```

   An unsuccessful test displays the following output:

   ```
   SQL> @test.sql
   message put on queue = 10203040506070809000
   ```

```
Error: Oracle Database Gateway for WebSphere MQ verification script failed.
ORA-29400: data cartridge error
MQI MQOPEN failed. completion code=2, reason code=2085
```

A

The PGM, PGM_UTL8, and PGM_SUP Packages

Use the Visual Workbench when developing applications that access WebSphere MQ through the gateway. The Visual Workbench defines an interface for accessing WebSphere MQ and automatically generates the PL/SQL code (the MIP) for Oracle applications to interface with the gateway. Refer to the *Oracle Procedural Gateway Visual Workbench for WebSphere MQ Installation and User's Guide for Microsoft Windows (32-Bit)* for more information about Visual Workbench.

The MIP uses definitions from the `PGM`, `PGM_UTL8`, and `PGM_SUP` packages. When necessary, you can alter the MIP to include WebSphere MQ functions that are not supported by Visual Workbench. This is done with the definitions and procedures from the `PGM`, and `PGM_UTL8`, and `PGM_SUP` packages.

The `PGM`, `PGM_UTL8`, and `PGM_SUP` packages are installed when the Visual Workbench repository or the DG4MQ deployment environment is created. For more information, refer to "Installing the Oracle Visual Workbench Repository" on page 7-12 and "Preparing the Production Oracle Database" on page 7-16.

This appendix discusses the `PGM`, `PGM_UTL8`, and `PGM_SUP` packages in the following sections:

- PGM Package, DG4MQ Gateway Procedures, and Data Type Definitions on page A-1
- MQCLOSE Procedure on page A-6
- MQGET Procedure on page A-7
- MQOPEN Procedure on page A-13
- MQPUT Procedure on page A-15
- PGM_SUP Package on page A-18

PGM Package, DG4MQ Gateway Procedures, and Data Type Definitions

The gateway procedures and type definitions of the `PGM` package are modeled after the WebSphere MQ MQI calls. For all the relevant calls and structures found in MQI, a corresponding counterpart exists in `PGM` and the associated data type definitions exist in `pgmobj.sql`. The gateway procedures and `PGM` type definitions are named the same as their MQI counterparts. However, the data types of arguments or structure fields are changed into corresponding PL/SQL data types.

Using these procedures and type definitions in an Oracle application is very similar to writing a WebSphere MQ application. The fields of all PGM type definitions are initialized. These initialization values are based on default values defined by MQI.

The use of gateway procedures and PGM type definitions requires extensive knowledge of MQI and WebSphere MQ programming in general. These procedures and records follow the MQI flowchart, semantics, and syntax rules.

The PGM package is installed when the Visual Workbench repository or the DG4MQ deployment environment is created and is granted public access. It has no schema because the gateway omits all schema names when describing or running a procedure. No schema qualifiers need to be prefixed to the names of the procedures and type definitions.

> **See Also:** Refer to IBM *MQSeries Application Programming Reference* for complete information about writing WebSphere MQ applications and using MQI calls.

Summary of Procedures and Type Definitions

The gateway procedures and PGM provide the following procedures and type definitions:

Table A–1   Procedures and Type Definitions

Procedure	Procedure Purpose	Type Definitions Used by the Procedure
MQOPEN	Opens a queue.	PGM.MQOD and PGM.MQOH
MQPUT	Sends a message to the queue that was opened by MQOPEN	PGM.MQMD PGM.MQOH PGM.MQPMO
	Sends a message longer than 32767 bytes to the queue	PGM.MQMD PGM.MQOH PGM.MQPMO PGM.MQPUT_BFFER
MQGET	Retrieves or scans a message from the queue that was opened by MQOPEN	PGM.MQMD PGM.MQOH PGM.MQGMO
	Sends a message longer than 32767 bytes to the queue	PGM.MQMD PGM.MQOH PGM.MQGMO PGM.MQGET_BFFER
MQCLOSE	Closes the queue that was opened by MQOPEN	Does not use a type definition.

Procedure Conventions

The gateway procedures are described in alphabetic order in this appendix. The type definitions are described with the procedures that use them. Only type definition fields that can be changed are described. Other fields equivalent to MQI fields are left out because they are reserved for WebSphere MQ, are not supported by the gateway, or contain values that should not be changed.

A procedure's definition is shown using the IBM argument names associated with the equivalent MQI call. For example:

```
MQGET(hobj, mqmd, mqgmo, msg)
```

The syntax of the `MQGET` call is as follows:

```
MQGET(handle, descript, get_options, message);
where:
```

- *handle* is your name for the first argument specified in the definition as `hobj`.

- *descript* is your name for the second argument specified in the definition as `mqmd`.

- *get_options* is your name for the third argument specified in the definition as `mqgmo`.

- *message* is your name for the fourth argument specified in the definition as `msg`.

  You can use your own names for these arguments if you code the arguments in the order shown in the definition.

For more information about PL/SQL, refer to the *Oracle Database PL/SQL Language Reference*.

MQI Calls Performed by the Gateway

The following MQI calls have no equivalent procedures in the gateway because the Oracle database and the gateway automatically perform the functions of these MQI calls:

- `MQBACK`

  Transaction control is handled by the Oracle transaction coordinator. The Oracle application does not need to invoke a separate `MQBACK` call to undo the changes sent to WebSphere MQ.

- `MQCONN`

  A connection to a queue manager is established by the Oracle database and the gateway whenever an Oracle application refers to a gateway procedure. The database link name that is used when calling the gateway procedure determines which queue manager the gateway connects to.

- `MQCMIT`

  Transaction control is handled by the Oracle transaction coordinator. An Oracle application does not need to invoke a separate `MQCMIT` call to commit the changes sent to WebSphere MQ.

- `MQDISC`

  Connections to a queue manager are closed by the Oracle database and gateway. An Oracle application does not need to close the connection with the queue manager. Ending the current Oracle session or dropping the database link causes the queue manager connection to end.

Unsupported MQI Calls

The following MQI calls are not supported by the gateway:

- `MQINQ`

- `MQPUT1`

■   MQSET

Migration Tips

This section provides information about how to upgrade Oracle9*i* DG4MQ and existing customized PL/SQL application programs to use Oracle Database Gateway for WebSphere MQ features. DG4MQ data types and RPC API prototypes are changed to meet the requirements of the gateway infrastructure.

When upgrading DG4MQ to Oracle 10*g* release 2 or higher, Oracle recommends that you install the newer version of DG4MQ on a separate development Oracle system. After you have finished with system configuration and testing, transfer all of the COBOL copy books and regenerate and recompile MIPs using the Oracle Visual Workbench. For customized codes, make necessary changes and recompile.

Migrating DG4MQ Releases 8 and 9 PL/SQL Applications

To migrate DG4MQ releases 8 and 9 PL/SQL applications:

1. In the PL/SQL declarative section, remove `dblink` references from the following DG4MQ data types:

    ■   `PGM8.MQOD`

    ■   `PGM8.MQMD`

    ■   `PGM8.MQPMO`

    ■   `PGM8.MQGMO`

    Then remove the following `PGM8.MQ*RAW` data types:

    ■   `PGM8.MQODRAW`

    ■   `PGM8.MQMDRAW`

    ■   `PGM8.MQPMORAW`

    ■   `PGM8.MQGMORAW`

2. In the PL/SQL declarative section, change the data type of the handle of the queue, the third argument of `PGM.MQOPEN`, from `BINARY_INTEGER` to `PGM.MQOH` and replace the package name `PGM8` with `PGM`.

    Change the data type of the handles of the queue, the third argument of `PGM.MQOPEN`, from `BINARY_INTEGER` to `PGM.MQOH`.

    For example, for version 8 and 9 change the following data types to those listed for Oracle 10*g*:

    ```
    objdesc     PGM8.MQOD;
    msgdesc     PGM8.MQMD;
    putmsgopts PGM8.MQPMO;
    getmsgopts PGM8.MQGMO;
    hobj        BINARY_INTEGER;
    mqodRaw     PGM8.MQODRAW;
    mqmdRaw     PGM8.MQMDRAW;
    mqpmoRaw    PGM8.MQPMORAW;
    mqgmoRaw    PGM8.MQGMORAW;
    ```

    The data types for Oracle 10*g* release 2 and higher:

    ```
    objdesc     PGM.MQOD;
    msgdesc     PGM.MQMD;
    putmsgopts  PGM.MQPMO;
    ```

```
getmsgopts  PGM.MQGMO;
hobj        PGM.MQOH;
```

3. In the PL/SQL executable section, remove `dblink` references from the following DG4MQ procedures:

```
PGM8.MQOPEN@dblink()
PGM8.MQPUT@dblink()
PGM8.MQGET@dblink()
PGM8.MQCLOSE@dblink()
```

Then define the `dblink` in the new `PGM.MQOD` type where the object queue name is defined.

For example, for version 8 and 9:

```
objdesc.objectname := 'QUEUE1';
```

For Oracle 10*g* release 2 and higher:

```
objdesc.objectname := 'QUEUE1';
objdesc.dblinkname := 'dblink';
```

4. If necessary, change the package name `PGM8` of all DG4MQ procedures to `PGM`.

For example, for version 8 and 9:

```
PGM8.MQOPEN@dblink();
PGM8.MQPUT@dblink();
PGM8.MQGET@dblink();
PGM8.MQCLOSE@dblink();
```

For Oracle 10*g* release 2 and higher:

```
PGM.MQOPEN;
PGM.MQPUT;
PGM.MQGET;
PGM.MQCLOSE;
```

5. In the PL/SQL executable section, remove all statements starting with `PGM_UTL8.RAW_TO_*`, remove all `PGM_UTL8.TO_RAW` statements, and replace all references to the `MQ*RAW` data types with their matching `MQ*` data types in the following DG4MQ procedures:

   - `PGM.MQOPEN;`
   - `PGM.MQPU;`
   - `PGM.MQGET;`
   - `PGM.MQCLOSE;`

For example, for versions 8 and 9:

```
mqodRaw := PGM_UTL8.TO_RAW(objdesc);
PGM8.MQOPEN@dblink(mqodRaw, options, hobj);
objdesc := PGM_UTL8.RAW_TO_MQMD(mqodRaw);
mqmdRaw := PGM_UTL8.TO_RAW(msgdesc);
mqpmoRaw := PGM_UTL8.TO_RAW(putmsgopts);
PGM8.MQPUT@dblink(hobj, mqmdRaw, mqpmoRaw, putbuffer);
putmsgopts := PGM_UTL8.RAW_TO_MQPMO(mqpmoRaw);
msgdesc := PGM_UTL8.RAW_TO_MQMD(mqmdRaw);

mqmdRaw := PGM_UTL8.TO_RAW(msgdesc);
mqgmoRaw := PGM_UTL8.TO_RAW(getmsgopts);
```

```
PGM8.MQGET@dblink(hobj, mqmdRaw, mqgmoRaw, putbuffer);
getmsgopts := PGM_UTL8.RAW_TO_MQGMO(mqgmoRaw);
msgdesc := PGM_UTL8.RAW_TO_MQMD(mqmdRaw);
```

For Oracle 10*g* release 2 and higher:

```
PGM.MQOPEN(objdesc, options, hobj);
PGM.MQPUT(hobj, msgdesc, putmsgopts, putbuffer);
PGM.MQGET(hobj, msgdesc, getmsgopts, getbuffer);
```

6. In PL/SQL executable section, remove all statements that reference the old `MQ*RAW` data types.

Migrating DG4MQ Release 4.0.1.*.* PL/SQL Applications

To migrate applications:

1. In the PL/SQL declarative section, remove `dblink` references from the following DG4MQ data types:

- `PGM.MQOD`

- `PGM.MQMD`

- `PGM.MQPMO`

- `PGM.MQGMO`

2. In the PL/SQL executable section, remove `dblink` references from the following DG4MQ procedures and define the `dblink` in the new `PGM.MQOD` object where the object queue name is defined:

- `PGM.MQOPEN@dblink`()

- `PGM.MQPUT@dblink`()

- `PGM.MQGET@dblink`()

- `PGM.MQCLOSE@dblink`()

For example, for version 4:

```
PGM.MQOPEN@dblink(objdesc, options, hobj);
objdesc.objectname :='QUEUE1';
PGM.MQPUT@dblink(hobj, msgdesc, putmsgopts, putbuffer);
PGM.MQGET@dblink(hobj, msgdesc, getmsgopts, putbuffer);
PGM.MQCLOSE@dblink(hobj, options);
```

MQCLOSE Procedure

`MQCLOSE` closes a queue. On return, the queue handle is invalid and your application must reopen the queue with another call to `MQOPEN` before issuing another `MQPUT`, `MQGET`, or `MQCLOSE` call to the queue.

`MQCLOSE` differs from MQI calls in the following ways:

- The connection handle argument is omitted from `MQCLOSE` because the gateway automatically takes care of managing queue manager connections.

- The MQI completion code is not included in the procedure argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

- The MQI reason code is not included in the procedure argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message returned to the caller.

Definition

MQCLOSE(*hobj*, *options*)

where:

- *hobj* contains the handle for the queue to close. The handle is returned by a previous call to MQOPEN. This input argument is a new PGM.MQOH object in Oracle 10*g* release 2.

- *options* specifies the close action. Use PGM_SUP.MQCO_NONE or the other PGM_SUP constants for a close option. Refer to MQCLOSE Values on page A-23. This input argument is of the BINARY_INTEGER PL/SQL data type.

You can use your own variable names when arguments are in the required order as follows:

MQCLOSE(handle, close_options);

MQGET Procedure

MQGET retrieves a message from a queue. The queue must already be open from a previous call to MQOPEN with the PGM_SUP.MQOO_INPUT_AS_Q_DEF (or an equivalent option) option set. Retrieved messages for this form of MQGET must be shorter than 32767 bytes.

MQGET differs from MQI calls in the following ways:

- The connection handle argument is omitted from MQGET because the gateway automatically takes care of managing queue manager connections.

- The MQI completion code is not included in the procedure's argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

- The MQI reason code is not included in the procedure's argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message that was returned to the caller.

- The msg length argument is not included in the procedure's argument list because the Oracle database and the gateway automatically keep track of the message data length.

Definition

MQGET(*hobj*, *mqmd*, *mqgmo*, *msg*)

where:

- *hobj* contains the handle for the queue to open. The handle is returned by a previous call to MQOPEN. This input argument is a new PGM.MQOH object in Oracle 10*g* release 2.

- *mqmd* is used on input to describe the attributes of the message being retrieved. Use the fields of the PGM.MQMD object type definition to describe these attributes.

  On output, mqmd contains information about how the request was processed. The queue manager sets some of the PGM.MQMD object fields on return.

This input and output argument is PL/SQL PGM.MQMD data type. For the details of PGM.MQMD, refer to PGM.MQMD Type Definition on page A-10.

- *mqgmo*is used on input to describe the option values that control the retrieve request. Use the fields of the PGM.MQGMO object type definition to describe these options.

  On output, the queue manager sets some of the PGM.MQGMO object fields on return.

  This input and output argument is PL/SQL PGM.MQGMO data type. For the details of the PGM.MQGMO object, refer to PGM.MQGMO Type Definition on page A-13.

- *msg* contains the retrieved message. This output argument is PL/SQL data type RAW or PGM.MQGET_BUFFER.

Examples

1. Using your own variable names when arguments are in the required order:

   ```
   MQGET(handle, descript, opts, message);
   ```

2. The following example, which is provided as a sample with the gateway (ORACLE_ HOME\dg4mq\getsample.sql on Microsoft Windows and *ORACLE_ HOME*/dg4mq/sample/getsample.sql on UNIX based systems), reads all messages from a WebSphere MQ queue. For more information, refer to the IBM publication on WebSphere MQ Application Programming.

Example A–1   getsample.sql

```
--
-- Copyright Oracle, 2007 All Rights Reserved.
--
-- NAME
--   getsample.sql
--
-- DESCRIPTION
--
--   Specify the database link name you created for the gateway. To do this,
--   replace the database link name 'YOUR_DBLINK_NAME' with the dblink name
--   you chose when the database link was created.
--
--   This script performs a test run for the MQSeries gateway. In this
--   script the queuename is 'YOUR_QUEUE_NAME', replace it with a valid
--   queue name at the queue manager the gateway is configured for.
--
-- NOTES
--   Run the script from the SQL*Plus command line.
--
--   Make the sure the user is granted 'EXECUTE' on package dbms_output
--

SET SERVEROUTPUT ON
DECLARE

    objdesc      PGM.MQOD;
    msgDesc      PGM.MQMD;
    getOptions   PGM.MQGMO;
    objectHandle PGM.MQOH;
    message      raw(32767);

BEGIN
```

```
    objdesc.OBJECTNAME := 'QUEUE1';
    objdesc.DBLINKNAME := 'dg4mqdepdblink';

    -- Open the queue 'YOUR_QUEUE_NAME' for reading.

    PGM.MQOPEN(objdesc, PGM_SUP.MQOO_INPUT_AS_Q_DEF, objectHandle);

    -- Get all messages from the queue.

    WHILE TRUE LOOP

        -- Reset msgid and correlid to get the next message.

        msgDesc.MSGID := PGM_SUP.MQMI_NONE;
        msgDesc.CORRELID := PGM_SUP.MQCI_NONE;

        PGM.MQGET(objectHandle, msgDesc, getOptions, message);

        -- Process the message....
        DBMS_OUTPUT.PUT_LINE('message read back = ' || rawtohex(message));

 END LOOP;

EXCEPTION

    WHEN PGM_SUP.NO_MORE_MESSAGES THEN

        DBMS_OUTPUT.PUT_LINE('Warning: No more message found on the queue');

        -- Close the queue again.

        PGM.MQCLOSE(objectHandle, PGM_SUP.MQCO_NONE);

    WHEN OTHERS THEN

        -- Re-raise the error;

        DBMS_OUTPUT.PUT_LINE('Error: Oracle Database Gateway for WebSphere MQ
 verification script failed.');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
 raise;

END;
/
```

Notes:

> **Note:** The PL/SQL block fails if the exception clause is left out. In that case, the PGM_SUP.NO_MORE_MESSAGES error code is raised. The MSGID and CORRELID fields that are used for MQGET are set after each call to MQGET. If they are not reset at each cycle, then MQGET checks for the next message that has the same identifiers as the last read operation, which usually do not exist. The PL/SQL block would only read one message.

PGM.MQMD Type Definition

PGM.MQMD specifies the control information that accompanies a message when it travels between the sending and receiving applications. It also contains information about how the message is handled by the queue manager or by the receiving application. PGM.MQMD describes the attributes of the message being retrieved.

You can use the default values for PGM.MQMD fields or change the fields for your application requirements. For example, to change a field value, do the following:

mqmd.field_name := field_value;

where:

- *mqmd* is the PGM.MQMD object data type and it describes the attributes of the message being retrieved

- *field_name* is a field name of the PGM.MQMD object type definition. You can set as many fields as necessary. Refer to Table A–2 for field names and descriptions.

- *field_value* is the value to assign to *field_name*. You can specify a value or use a PGM_SUP constant to assign a value.

Table A–2   PGM.MQMD Object Fields

Field Name	Description	PL/SQL Data Type	Initial Value
REPORT	Allows the application that sends a message to specify which report message (or messages) should be created by the queue manager when an expected or unexpected event occurs. Use a PGM_SUP constant to assign a value. Refer to REPORT Field on page A-21.	RAW(4)	PGM_SUP.MQRO_NONE
MSGTYPE	Specifies the message type: reply message, report message, or normal message (datagram). Use a PGM_SUP constant to assign a value. Refer to MSGTYPE Field on page A-20.	BINARY_INTEGER	PGM_SUP.MQMT_DATAGRAM
EXPIRY	Specifies the amount of time that a message stays in a queue. The expiration period is in tenths of a second, and is set by the sending application. Use a PGM_SUP constant to assign a value. Refer to EXPIRY Field on page A-20.	BINARY_INTEGER	PGM_SUP.QMEI_UNLIMITED
FEEDBACK	Used with the REPORT field to indicate the kind of report. Use a PGM_SUP constant to assign a value. Refer to FEEDBACK Field on page A-20.	BINARY_INTEGER	PGM_SUP.MQFB_NONE
ENCODING	Used for numeric values in the message data. Use a PGM_SUP constant to assign a value. Refer to ENCODING Field on page A-19.	RAW(4)	PGM_SUP.MQENC_NATIVE
CODEDCHARSETID	Specifies the coded character set identifier of the characters in the message. Use a PGM_SUP constant to assign a value. Refer to CODEDCHARSETID Field on page A-19.	BINARY_INTEGER	PGM_SUP.MQCCSI_DEFAULT

Table A–2   (Cont.) PGM.MQMD Object Fields

Field Name	Description	PL/SQL Data Type	Initial Value
FORMAT	A free format name used to inform the receiver about the contents of the message. Specify a format or use a `PGM_SUP` constant. Refer to FORMAT Field on page A-20.	CHAR(8)	PGM_SUP.MQFMT_ NONE
PRIORITY	Specifies message priority. Specify a value greater than or equal to 0 (zero is the lowest priority), or use a `PGM_ SUP` constant. Refer to PRIORITY Field on page A-21.	BINARY_INTEGER	PGM_SUP.MQPRI_ PRIORITY_AS_Q_ DEF
PERSISTENCE	An input field for the sending application. Persistent messages survive when a queue manager is restarted. Non persistent messages and messages in temporary queues are lost when a queue manager is restarted. Specify the desired persistence with a `PGM_SUP` constant. Refer to PERSISTENCE Field on page A-21.	BINARY_INTEGER	PGM_SUP.MQPER_ PERSISTENCE_AS_ Q_DEF
MSGID	Specifies the message identifier of the message to be retrieved (when receiving a message). If no value is specified when a sending a message (`PGM_SUP.MQMI_NONE`), then the queue manager assigns a unique value.	RAW(24)	PGM_SUP.MQMI_NONE
CORRELID	Specifies the correlation identifier for the message to retrieve when receiving a message (refer to the `MSGID` field). When sending a message, specify any value, or use `PGM_SUP.MQCI_NONE` if the message does not require a correlation ID.	RAW(24)	PGM_SUP.MQCI_NONE
BACKOUTCOUNT	An output field for the `MQGET` procedure. It indicates the number of times a message was placed back on a queue because of a rollback operation.	BINARY_INTEGER	Zero
REPLYTOQ	Specifies the name of the reply-to queue. This is an input field for `MQPUT` and allows the sending application to indicate where reply messages should be sent. It is also an output field for `MQGET` and tells the receiving application where to send a reply.	CHAR(48)	NULL
REPLYTOQMGR	Specifies the queue manager to which the reply message or report should be sent. This is an input field for `MQPUT` and an output field for `MQGET`.	CHAR(48)	NULL

Table A–2   (Cont.) PGM.MQMD Object Fields

Field Name	Description	PL/SQL Data Type	Initial Value
USERIDENTIFIER	An output field for receiving applications. It identifies the user that sent the message. Sending applications can specify a user on input if the CONTEXT field for the mqpmo argument of MQPUT was set to PGM_SUP.MQPMO_SET_IDENTITY_ CONTEXT or to PGM_SUP.MQPMO_SET_ ALL_CONTEXT.	CHAR(12)	NULL
ACCOUNTINGTOKEN	Used to transfer accounting information between applications. Sending applications provide accounting information or use PGM_ SUP.MQACT_NONE to specify that no accounting information is included.	CHAR(32)	PGM_SUP.MQACT_ NONE
APPLIDENTITYDATA	Specifies more information to send along with the message to help the receiving application provide more information about the message or its sender.	CHAR(32)	NULL
PUTAPPLTYPE	Describes the kind of application that placed the message on the queue. Use a PGM_SUP constant to assign a value. Refer to PUTAPPLTYPE Field on page A-21.	BINARY_INTEGER	PGM.MQAT_NO_ CONTEXT
PUTAPPLNAME	Specifies the name of the application that placed the message on the queue. Sending applications specify a name or let the queue manager fill in this field. This is an output field for receiving applications.	CHAR(28)	NULL
PUTDATE	Specifies the date on which a message was placed on the queue. Sending applications can set a date or let the queue manager take care of it. The date format used by the queue manager is YYYYMMDD. This is an output field for receiving applications.	CHAR(8)	NULL
PUTTIME	Specifies the time that a message was placed on the queue. Sending applications can set a time or let the queue manager take care of it. The time format that is used by the queue manager is HHMMSSTH. This is an output field for receiving applications.	CHAR(8)	NULL
APPLORIGINDATA	Used by the sending application to add information to the message about the message origin. This is an output field for receiving applications.	CHAR(4)	NULL

PGM.MQGMO Type Definition

Use PGM.MQGMO to specify option and control information about how the message is retrieved from a queue. You can use the default values for PGM.MQGMO fields or change the fields for your application requirements. For example, to change a field value:

```
mqgmo.field_name := field_value
```

where:

- mqgmo is the PGM.MQGMO object data type, and it specifies option and control information about how the message is retrieved from a queue.

- field_name is a field name of the PGM.MQGMO type definition. You can set as many fields as necessary. Refer to Table A–3 for names and field descriptions.

- field_value is the value to assign to *field_name*. You can specify a value or use a PGM_SUP constant to assign a value.

Table A–3   PGM.MQGMO Fields

Field Name	Description	PL/SQL Data Type	Initial Value
OPTIONS	Specifies options to control the MQGET procedure. Add one or more PGM_SUP constants to set it. Refer to OPTIONS Field on page A-18.	BINARY_INTEGER	PGM.MQGMO_ SYNCPOINT (Messages that are retrieved from the queue are coordinated by the Oracle transaction coordinator.)
WAITINTERVAL	Specifies the maximum time in milliseconds that MQGET waits for a message to arrive in the queue. WAITINTERVAL should be equal to or greater than 0, or set to the value of PGM_SUP.MQWI_UNLIMITED (unlimited wait interval).	BINARY_INTEGER	Zero
RESOLVEDQNAME	Contains the resolved name of the destination queue from which the message was retrieved. This is an output field set by the queue manager upon return from the call.	CHAR(48)	NULL

MQOPEN Procedure

MQOPEN establishes access to a queue. Depending on the mode selected to open the queue, an application can issue subsequent MQPUT, MQGET, or MQCLOSE calls.

MQOPEN differs from MQI calls in the following ways:

- The connection handle argument is omitted from MQOPEN because the gateway automatically takes care of managing queue manager connections.

- The MQI completion code is not included in the procedure argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

- The MQI reason code is not included in the procedure argument list. If the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message that is returned to the caller.

Definition

```
MQOPEN(mqod, options, hobj)
```

where:

- `mqod` specifies the queue to open. Use the fields of the `PGM.MQOD` type definition to describe these attributes. On output, the queue manager sets some of the `PGM.MQOD` object fields on return.

  This input and output argument is PL/SQL `PGM.MQOD` data type. For details of `PGM.MQOD`, refer to PGM.MQOD Type Definition on page A-14.

- `options` specifies the kind of open. Refer to MQOPEN Values on page A-23. This input argument is of the PL/SQL `BINARY_INTEGER` data type.

- `hobj` contains the handle of the queue after the queue is opened and becomes an input argument for subsequent PGM calls. The queue handle remains valid until one of the following conditions occur:

  - The queue is closed by a call to `MQCLOSE`

  - The current transaction is made permanent by a `COMMIT` or `ROLLBACK` command

  - The Oracle user session is ended by a `DISCONNECT` command. This output argument is of the `PGM.MQOH` data type.

You can use your own variable names when arguments are in the required order as follows:

```
MQOPEN and(descript, open_options, handle);
```

PGM.MQOD Type Definition

`PGM.MQOD` is used to define the object to open. You can use the default values for `PGM.MQOD` fields or change the fields for your application requirements. For example, you can change a field value as follows:

```
mqod.field_name := field_value
```

where:

- `mqod` is the `PGM.MQOD` data type and specifies the object to open.

- `field_name` is a field name of the `PGM.MQOD` type definition. You can set as many fields as necessary. Refer to Table A–4 for field names and descriptions.

- `field_value` is the value to assign to `field_name`. You can specify a value or use a `PGM_SUP` constant to assign a value.

Table A–4   PGM.MQOD Object Fields

Field Name	Description	PL/SQL Data Type	Initial Value
OBJECTTYPE	Specifies the object to open. Use a `PGM_SUP` constant to assign a value. Refer to OBJECTTYPE Field on page A-22.	BINARY_INTEGER	PGM_SUP.MQOT_Q(queue)
DBLINKNAME	Specifies the database link name.	CHAR(64)	NULL
OBJECTNAME	Specifies the local name of the object as defined by the queue manager.	CHAR(48)	NULL

Table A–4 (Cont.) PGM.MQOD Object Fields

Field Name	Description	PL/SQL Data Type	Initial Value
OBJECTQMGRNAME	Specifies the name of the queue manager for the object defined by OBJECTNAME. Leave OBJECTQMGRNAME set to null values because the gateway supports only the opening of objects at the connected queue.	CHAR(48)	NULL
DYNAMICQNAME	Is ignored unless the OBJECTNAME field specifies the name of a model queue. When a model queue is involved, then this field specifies the name of the dynamic queue to be created at the queue manager to which the gateway is connected.	CHAR(48)	AMQ.*
ALTERNATEUSERID	If the options argument of MQOPEN is set to the value of PGM_SUP.MQOO_ALTERNATE_USER_AUTHORITY, then this field specifies the alternate user ID which the queue manager uses to check the authorization for the queue being opened.	CHAR(12)	NULL

MQPUT Procedure

MQPUT sends a message to a queue. The queue must already be open by a previous call to MQOPEN with its options argument set to the value of PGM_SUP.MQOO_OUTPUT.

MQPUT differs from MQI calls as follows:

- The connection handle argument is omitted from MQPUT because the gateway automatically takes care of managing queue manager connections.

- The MQI completion code is not included in the procedure argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

- The MQI reason code is not included in the procedure argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message returned to the caller.

- The msg length argument is not included in the procedure argument list because the Oracle database and the gateway automatically keep track of the message data length.

Definition

MQPUT(*hobj*, mqmd, mqpmo, *msg*)

where:

- hobj contains the handle for the queue to send the message to. The handle is returned by a previous call to MQOPEN. This input argument is a new PGM.MQOH in Oracle10*g* release 2.

- mqmd is used on input to describe the attributes of the message being retrieved. Use the fields of the PGM.MQMD type definition to describe these attributes. On output, mqmd contains information about how the request was processed. The queue manager sets some of the PGM.MQMD fields on return.

This input and output argument is a PGM.MQMD. For details of PGM.MQMD, refer to PGM.MQMD Type Definition on page A-10.

- mqpmo is used on input to describe the option values that control the put request. Use the fields of the PGM.MQPMO type definition to describe these options. On output, the queue manager sets some of the PGM.MQPMO fields on return.

  This input and output argument is PGM.MQPMO. For details of PGM.MQPMO, refer to PGM.MQPMO Type Definition on page A-17.

- msg contains the message to send. This input argument is PL/SQL data type RAW or PGM.MQPUT_BUFFER.

Example

1. You can use your own variable names when arguments are in the required order:

   ```
   MQPUT(handle, descript, options, message);
   ```

2. The following sample, which is provided as a sample with the gateway (ORACLE_ HOME\dg4mq\sample\putsample.sql on Microsoft Windows and *ORACLE_ HOME*/dg4mq/sample/putsample.sql on UNIX based systems), sends a message shorter than 32767 bytes:

Example A–2   putsample.sql

```
--
-- Copyright Oracle, 2005 All Rights Reserved.
--
-- NAME
--   putsample.sql
--
-- DESCRIPTION
--
--   Specify the database link name you created for the gateway. To do this,
--   replace the database link name 'YOUR_DBLINK_NAME' with the dblink name
--   you chose when the database link was created.
--
--   This script performs a test run for the MQSeries gateway. In this
--   script the queuename is 'YOUR_QUEUE_NAME', replace it with a valid
--   queue name at the queue manager the gateway is configured for.
--
-- NOTES
--   Run the script from the SQL*Plus command line.
--
--   Make the sure the user is granted 'EXECUTE' on package dbms_output
--

SET SERVEROUTPUT ON

DECLARE
    objdesc     PGM.MQOD;
    msgDesc     PGM.MQMD;
    putOptions  PGM.MQPMO;
    objectHandle PGM.MQOH;
    message     raw(255);

BEGIN

    objdesc.OBJECTNAME := 'QUEUE1';
    objdesc.DBLINKNAME := 'dg4mqdepdblink';
```

```
-- Open the queue 'YOUR_QUEUE_NAME' for sending.

PGM.MQOPEN(objdesc, PGM_SUP.MQOO_OUTPUT, objectHandle);

-- Put the message buffer on the queue.

message := '010203040050607080900';

PGM.MQPUT(objectHandle, msgDesc, putOptions, message);

-- Print the message we are putting on the queue

dbms_output.put_line('message put on queue = ' || rawtohex(message));

-- Close the queue again.

PGM.MQCLOSE(objectHandle, PGM_SUP.MQCO_NONE);

EXCEPTION

-- something else went wrong.. tell the user.

WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: Procedural Gateway for IBM MQSeries
verification script failed.');
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    PGM.MQCLOSE(objectHandle, PGM_SUP.MQCO_NONE);

END;
/
```

PGM.MQPMO Type Definition

PGM.MQPMO is used to define the mqpmo argument of MQPUT. It specifies option and control information for processing a message.

You can use the default values for PGM.MQPMO fields or change the fields for the application requirements. For example, to change a field value:

mqpmo.*field_name* := *field_value*

where:

- mqpmo is the PGM.MQPMO data type and specifies option and control information about how the message is processed and put into a queue.

- field_name is a field name of the PGM.MQPMO type definition. You can set as many fields as necessary. Refer to Table A–5 for field names and descriptions.

- field_value is the value to assign to *field_name*. You can specify a value or use a PGM_SUP constant to assign a value.

Table A–5   PGM.MQPMO Fields

Field Name	Description	PL/SQL Data Type	Initial Value
OPTIONS	Specifies options to control the MQPUT procedure. The field is set by adding one or more of the PGM_SUP definitions. Refer to "OPTIONS Field" on page A-22.	BINARY_INTEGER	PGM.MQPMO_ SYNCPOINT (Messages placed on the queue are coordinated by the Oracle transaction coordinator.)
CONTEXT	Specifies the object handle of the input queue. It is only used when the OPTIONS field has the bit PGM_SUP.MQPMO_PASS_ IDENTITY_ CONTEXT or the bit PGM_SUP.MQPMO_PASS_ ALL_CONTEXT set.	BINARY_INTEGER	Zero
RESOLVEDQNAME	Contains the resolved name of the destination queue. This is an output field set by the queue manager on return.	CHAR(48)	NULL
RESOLVEDQMGRNAME	Contains the resolved name of the queue manager for the queue name returned in the RESOLVEDQNAME field. This is an output field set by the queue manager on return.	CHAR(48)	NULL

PGM_SUP Package

PGM_SUP contains constant and exception definitions to use with the gateway procedures and PGM type definitions. Using these values requires extensive knowledge of MQI and WebSphere MQ programming in general. These definitions follow the MQI definition rules. For complete information about writing WebSphere MQ applications, refer to the IBM *MQSeries Application Programming Reference*.

PGM.MQGMO Values

The following sections provide information about PGM.MQGMO values.

OPTIONS Field

```
MQGMO_NO_WAIT                    constant binary_integer := 0;
MQGMO_NONE                      constant binary_integer := 0;
MQGMO_WAIT                      constant binary_integer := 1;
MQGMO_SYNCPOINT                 constant binary_integer := 2;
MQGMO_NO_SYNCPOINT              constant binary_integer := 4;
MQGMO_SET_SIGNAL                constant binary_integer := 8;
MQGMO_BROWSE_FIRST              constant binary_integer := 16;
MQGMO_BROWSE_NEXT               constant binary_integer := 32;
MQGMO_ACCEPT_TRUNCATED_MSG      constant binary_integer := 64;
MQGMO_MARK_SKIP_BACKOUT         constant binary_integer := 128;
MQGMO_MSG_UNDER_CURSOR          constant binary_integer := 256;
MQGMO_LOCK                      constant binary_integer := 512;
MQGMO_UNLOCK                    constant binary_integer := 1024;
MQGMO_BROWSE_MSG_UNDER_CURSOR   constant binary_integer := 2048;
MQGMO_SYNCPOINT_IF_PERSISTENT   constant binary_integer := 4096;
MQGMO_FAIL_IF_QUIESCING         constant binary_integer := 8192;
MQGMO_CONVERT                   constant binary_integer := 16384;
MQGMO_LOGICAL_ORDER             constant binary_integer := 32768;
```

```
MQGMO_COMPLETE_MSG            constant binary_integer := 65536;
MQGMO_ALL_MSGS_AVAILABLE      constant binary_integer := 131072;
MQGMO_ALL_SEGMENTS_AVAILABLE  constant binary_integer := 262144;
```

VERSION Field

```
MQGMO_VERSION_1               constant binary_integer := 1;
MQGMO_CURRENT_VERSION         constant binary_integer := 1;
MQGMO_VERSION_2               constant binary_integer := 2;
MQGMO_VERSION_3               constant binary_integer := 3;
```

MATCHOPTIONS Field

```
MQMO_DEFAULT                  constant binary_integer := 3;
MQMO_NONE                     constant binary_integer := 0;
MQMO_MATCH_MSG_ID             constant binary_integer := 1;
MQMO_MATCH_CORREL_ID          constant binary_integer := 2;
MQMO_MATCH_GROUP_ID           constant binary_integer := 4;
MQMO_MATCH_MSG_SEQ_NUMBER     constant binary_integer := 8;
MQMO_MATCH_OFFSET             constant binary_integer := 16;
MQMO_MATCH_MSG_TOKEN          constant binary_integer := 32;
```

WAITINTERVAL

```
PGM_SUP.MQWI_UNLIMITED CONSTANT BINARY_INTEGER := -1;
PGM_SUP.MQWI_UNITS     CONSTANT BINARY_INTEGER := 1000;
```

PGM.MQMD Values

The following sections provide information about PGM.MQOD values.

CODEDCHARSETID Field

```
PGM_SUP.MQCCSI_DEFAULT  CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQCCSI_Q_MGR    CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQCCSI_EMBEDDED CONSTANT BINARY_INTEGER := -1;
```

ENCODING Field

```
PGM_SUP.
MQENC_NATIVE
  CONSTANT RAW(4) := '00000111';
```

ENCODING Field, Values for Binary Integers

```
PGM_SUP.MQENC_INTEGER_UNDEFINED CONSTANT RAW(4) := '00000000';
PGM_SUP.MQENC_INTEGER_NORMAL    CONSTANT RAW(4) := '00000001';
PGM_SUP.MQENC_INTEGER_REVERSED  CONSTANT RAW(4) := '00000002';
```

ENCODING Field, Values for Floating Point Numbers

```
PGM_SUP.MQENC_FLOAT_UNDEFINED     CONSTANT RAW(4) := '00000000';
PGM_SUP.MQENC_FLOAT_IEEE_NORMAL   CONSTANT RAW(4) := '00000100';
PGM_SUP.MQENC_FLOAT_IEEE_REVERSED CONSTANT RAW(4) := '00000200';
PGM_SUP.MQENC_FLOAT_S390          CONSTANT RAW(4) := '00000300';
```

ENCODING Field, Mask Values

```
PGM_SUP.MQENC_INTEGER_MASK  CONSTANT RAW(4) := '0000000f';
PGM_SUP.MQENC_DECIMAL_MASK  CONSTANT RAW(4) := '000000f0';
PGM_SUP.MQENC_FLOAT_MASK    CONSTANT RAW(4) := '00000f00';
PGM_SUP.MQENC_RESERVED_MASK CONSTANT RAW(4) := 'fffff000';
```

ENCODING Field, Values for Packed Decimal Integers

```
PGM_SUP.MQENC_DECIMAL_UNDEFINED CONSTANT RAW(4) := '00000000';
PGM_SUP.MQENC_DECIMAL_NORMAL    CONSTANT RAW(4) := '00000010';
PGM_SUP.MQENC_DECIMAL_REVERSED  CONSTANT RAW(4) := '00000020';
```

EXPIRY Field

```
PGM_SUP.MQEI_UNLIMITED   CONSTANT BINARY_INTEGER := -1;
PGM_SUP.MQEI_MIN_EXPIRY  CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQEI_UNITS       CONSTANT BINARY_INTEGER := 10;
```

FEEDBACK Field

```
PGM_SUP.MQFB_NONE                  CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQFB_SYSTEM_FIRST          CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQFB_EXPIRATION            CONSTANT BINARY_INTEGER := 258;
PGM_SUP.MQFB_COA                   CONSTANT BINARY_INTEGER := 259;
PGM_SUP.MQFB_COD                   CONSTANT BINARY_INTEGER := 260;
PGM_SUP.MQFB_QUIT                  CONSTANT BINARY_INTEGER := 256;
PGM_SUP.MQFB_CHANNEL_COMPLETED     CONSTANT BINARY_INTEGER := 262;
PGM_SUP.MQFB_CHANNEL_FAIL_RETRY    CONSTANT BINARY_INTEGER := 263;
PGM_SUP.MQFB_CHANNEL_FAIL          CONSTANT BINARY_INTEGER := 264;
PGM_SUP.MQFB_APPL_CANNOT_BE_STARTED CONSTANT BINARY_INTEGER := 265;
PGM_SUP.MQFB_TM_ERROR              CONSTANT BINARY_INTEGER := 266;
PGM_SUP.MQFB_APPL_TYPE_ERROR       CONSTANT BINARY_INTEGER := 267;
PGM_SUP.MQFB_STOPPED_BY_MSG_EXIT   CONSTANT BINARY_INTEGER := 268;
PGM_SUP.MQFB_XMIT_Q_MSG_ERROR      CONSTANT BINARY_INTEGER := 271;
PGM_SUP.MQFB_SYSTEM_LAST           CONSTANT BINARY_INTEGER := 65535;
PGM_SUP.MQFB_APPL_FIRST            CONSTANT BINARY_INTEGER := 65536;
PGM_SUP.MQFB_APPL_LAST             CONSTANT BINARY_INTEGER := 999999999;
```

FORMAT Field

```
MQFMT_NONE              constant char(8) := '        ';
MQFMT_ADMIN            constant char(8) := 'MQADMIN ';
MQFMT_CHANNEL_COMPLETED constant char(8) := 'MQCHCOM ';
MQFMT_CICS             constant char(8) := 'MQCICS  ';
MQFMT_COMMAND_1        constant char(8) := 'MQCMD1  ';
MQFMT_COMMAND_2        constant char(8) := 'MQCMD2  ';
MQFMT_DEAD_LETTER_HEADER constant char(8) := 'MQDEAD  ';
MQFMT_DIST_HEADER      constant char(8) := 'MQHDIST ';
MQFMT_EVENT            constant char(8) := 'MQEVENT ';
MQFMT_IMS              constant char(8) := 'MQIMS   ';
MQFMT_IMS_VAR_STRING   constant char(8) := 'MQIMSVS ';
MQFMT_MD_EXTENTION     constant char(8) := 'MQHMDE  ';
MQFMT_PCF              constant char(8) := 'MQPCF   ';
MQFMT_REF_MSG_HEADER   constant char(8) := 'MQHREF  ';
MQFMT_STRING           constant char(8) := 'MQSTR   ';
MQFMT_TRIGGER          constant char(8) := 'MQTRIG  ';
MQFMT_WORK_INFO_HEADER constant char(8) := 'MQHWIH  ';
MQFMT_XMIT_Q_HEADER    constant char(8) := 'MQXMIT  ';
```

MSGTYPE Field

```
PGM_SUP.MQMT_SYSTEM_FIRST CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQMT_REQUEST      CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQMT_REPLY        CONSTANT BINARY_INTEGER := 2;
PGM_SUP.MQMT_DATAGRAM     CONSTANT BINARY_INTEGER := 8;
PGM_SUP.MQMT_REPORT       CONSTANT BINARY_INTEGER := 4;
PGM_SUP.MQMT_SYSTEM_LAST  CONSTANT BINARY_INTEGER := 65535;
PGM_SUP.MQMT_APPL_FIRST   CONSTANT BINARY_INTEGER := 65536;
```

```
PGM_SUP.MQMT_APPL_LAST    CONSTANT BINARY_INTEGER := 999999999;
```

PERSISTENCE Field

```
PGM_SUP.MQPER_PERSISTENT          CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQPER_NOT_PERSISTENT      CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQPER_PERSISTENCE_AS_Q_DEF CONSTANT BINARY_INTEGER := 2;
```

PRIORITY Field

```
PGM_SUP.MQPRI_PRIORITY_AS_Q_DEF CONSTANT BINARY_INTEGER := -1;
PGM_SUP.MQPRI_MIN_PRIORITY      CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQPRI_MAX_PRIORITY      CONSTANT BINARY_INTEGER := 9;
```

PUTAPPLTYPE Field

```
MQAT_UNKNOWN     constant binary_integer := -1;
MQAT_NO_CONTEXT  constant binary_integer := 0;
MQAT_CICS        constant binary_integer := 1;
MQAT_MVS         constant binary_integer := 2;
MQAT_OS390       constant binary_integer := 2;
MQAT_IMS         constant binary_integer := 3;
MQAT_OS2         constant binary_integer := 4;
MQAT_DOS         constant binary_integer := 5;
MQAT_AIX         constant binary_integer := 6;
MQAT_UNIX        constant binary_integer := 6;
MQAT_QMGR        constant binary_integer := 7;
MQAT_OS400       constant binary_integer := 8;
MQAT_WINDOWS     constant binary_integer := 9;
MQAT_CICS_VSE    constant binary_integer := 10;
MQAT_WINDOWS_NT  constant binary_integer := 11;
MQAT_VMS         constant binary_integer := 12;
MQAT_GUARDIAN    constant binary_integer := 13;
MQAT_NSK         constant binary_integer := 13;
MQAT_VOS         constant binary_integer := 14;
MQAT_IMS_BRIDGE  constant binary_integer := 19;
MQAT_XCF         constant binary_integer := 20;
MQAT_CICS_BRIDGE constant binary_integer := 21;
MQAT_NOTES_AGENT constant binary_integer := 22;
MQAT_USER_FIRST  constant binary_integer := 65536;
MQAT_USER_LAST   constant binary_integer := 999999999;
MQAT_DEFAULT     constant binary_integer := 6;
```

REPORT Field

```
MQRO_NEW_MSG_ID                constant raw(4) := '00000000';
MQRO_COPY_MSG_ID_TO_CORREL_ID  constant raw(4) := '00000000';
MQRO_DEAD_LETTER_Q             constant raw(4) := '00000000';
MQRO_NONE                      constant raw(4) := '00000000';
MQRO_PAN                       constant raw(4) := '00000001';
MQRO_NAN                       constant raw(4) := '00000002';
MQRO_PASS_CORREL_ID            constant raw(4) := '00000040';
MQRO_PASS_MSG_ID               constant raw(4) := '00000080';
MQRO_COA                       constant raw(4) := '00000100';
MQRO_COA_WITH_DATA             constant raw(4) := '00000300';
MQRO_COA_WITH_FULL_DATA        constant raw(4) := '00000700';
MQRO_COD                       constant raw(4) := '00000800';
MQRO_COD_WITH_DATA             constant raw(4) := '00001800';
MQRO_COD_WITH_FULL_DATA        constant raw(4) := '00003800';
MQRO_EXPIRATION                constant raw(4) := '00200000';
```

```
MQRO_EXPIRATION_WITH_DATA       constant raw(4) := '00600000';
MQRO_EXPIRATION_WITH_FULL_DATA  constant raw(4) := '00E00000';
MQRO_EXCEPTION                  constant raw(4) := '01000000';
MQRO_EXCEPTION_WITH_DATA        constant raw(4) := '03000000';
MQRO_EXCEPTION_WITH_FULL_DATA   constant raw(4) := '07000000';
MQRO_DISCARD_MSG                constant raw(4) := '08000000';
```

VERSION Field

```
MQMD_VERSION_1              constant binary_integer := 1;
MQMD_VERSION_2              constant binary_integer := 2;
MQMD_CURRENT_VERSION        constant binary_integer := 2;
```

Report Field, Mask Values

```
PGM_SUP.MQRO_REJECT_UNSUP_MASK         CONSTANT RAW(4) := '101c0000';
PGM_SUP.MQRO_ACCEPT_UNSUP_MASK         CONSTANT RAW(4) := 'efe000ff';
PGM_SUP.MQRO_ACCEPT_UNSUP_IF_XMIT_MASK CONSTANT RAW(4) := '0003ff00';
```

PGM.MQOD Values

The following sections provide information about PGM.MQOD values.

OBJECTTYPE Field

```
PGM_SUP.MQOT_Q       CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQOT_PROCESS CONSTANT BINARY_INTEGER := 3;
PGM_SUP.MQOT_Q_MGR   CONSTANT BINARY_INTEGER := 5;
PGM_SUP.MQOT_CHANNEL CONSTANT BINARY_INTEGER := 6;
```

OBJECTTYPE Field, Extended Values

```
MQOT_ALL               constant binary_integer := 1001;
MQOT_ALIAS_Q           constant binary_integer := 1002;
MQOT_MODEL_Q           constant binary_integer := 1003;
MQOT_LOCAL_Q           constant binary_integer := 1004;
MQOT_REMOTE_Q          constant binary_integer := 1005;
MQOT_SENDER_CHANNEL    constant binary_integer := 1007;
MQOT_SERVER_CHANNEL    constant binary_integer := 1008;
MQOT_REQUESTER_CHANNEL constant binary_integer := 1009;
MQOT_RECEIVER_CHANNEL  constant binary_integer := 1010;
MQOT_CURRENT_CHANNEL   constant binary_integer := 1011;
MQOT_SAVED_CHANNEL     constant binary_integer := 1012;
MQOT_SVRCONN_CHANNEL   constant binary_integer := 1013;
MQOT_CLNTCONN_CHANNEL  constant binary_integer := 1014;
```

VERSION Field

```
MQOD_VERSION_1              constant binary_integer := 1;
MQOD_VERSION_2              constant binary_integer := 2;
MQOD_CURRENT_VERSION        constant binary_integer := 2;
```

PGM.MQPMO Values

The following sections provide information about PGM.MQPMO values.

OPTIONS Field

```
MQPMO_NONE                 constant binary_integer := 0;
MQPMO_SYNCPOINT            constant binary_integer := 2;
MQPMO_NO_SYNCPOINT         constant binary_integer := 4;
```

```
MQPMO_DEFAULT_CONTEXT           constant binary_integer := 32;
MQPMO_NEW_MSG_ID                constant binary_integer := 64;
MQPMO_NEW_CORREL_ID             constant binary_integer := 128;
MQPMO_PASS_IDENTITY_CONTEXT     constant binary_integer := 256;
MQPMO_PASS_ALL_CONTEXT          constant binary_integer := 512;
MQPMO_SET_IDENTITY_CONTEXT      constant binary_integer := 1024;
MQPMO_SET_ALL_CONTEXT           constant binary_integer := 2048;
MQPMO_ALTERNATE_USER_AUTHORITY  constant binary_integer := 4096;
MQPMO_FAIL_IF_QUIESCING         constant binary_integer := 8192;
MQPMO_NO_CONTEXT                constant binary_integer := 16384;
MQPMO_LOGICAL_ORDER             constant binary_integer := 32768;
```

VERSION Field

```
MQPMO_VERSION_1                 constant binary_integer := 1;
MQPMO_VERSION_2                 constant binary_integer := 2;
MQPMO_CURRENT_VERSION           constant binary_integer := 2;
```

MQCLOSE Values

The following sections provide information about MQCLOSE values.

hobj Argument

```
PGM_SUP.MQHO_UNUSABLE_HOBJ CONSTANT BINARY_INTEGER := -1;
```

options Argument

```
PGM_SUP.MQCO_NONE         CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQCO_DELETE       CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQCO_DELETE_PURGE CONSTANT BINARY_INTEGER := 2;
```

MQOPEN Values

The following sections provide information about MQOPEN values.

options Argument

```
MQOO_BIND_AS_Q_DEF              constant binary_integer := 0;
MQOO_INPUT_AS_Q_DEF             constant binary_integer := 1;
MQOO_INPUT_SHARED               constant binary_integer := 2;
MQOO_INPUT_EXCLUSIVE            constant binary_integer := 4;
MQOO_BROWSE                     constant binary_integer := 8;
MQOO_OUTPUT                     constant binary_integer := 16;
MQOO_INQUIRE                    constant binary_integer := 32;
MQOO_SET                        constant binary_integer := 64;
MQOO_SAVE_ALL_CONTEXT           constant binary_integer := 128;
MQOO_PASS_IDENTITY_CONTEXT      constant binary_integer := 256;
MQOO_PASS_ALL_CONTEXT           constant binary_integer := 512;
MQOO_SET_IDENTITY_CONTEXT       constant binary_integer := 1024;
MQOO_SET_ALL_CONTEXT            constant binary_integer := 2048;
MQOO_ALTERNATE_USER_AUTHORITY   constant binary_integer := 4096;
MQOO_FAIL_IF_QUIESCING          constant binary_integer := 8192;
MQOO_BIND_ON_OPEN               constant binary_integer := 16384;
MQOO_BIND_NOT_FIXED             constant binary_integer := 32768;
MQOO_RESOLVE_NAMES              constant binary_integer := 65536;
```

Maximum Lengths for Fields of PGM Type Definitions

These constants contain the maximum lengths allowed for fields used by the PGM Type Definitions. For example, the constant PGM_SUP.MQ_ACCOUNTING_TOKEN_LENGTH specifies that the maximum length for PGM.MQMD.ACCOUNTINGTOKEN is 32 characters.

```
MQ_ABEND_CODE_LENGTH           constant binary_integer := 4;
MQ_ACCOUNTING_TOKEN_LENGTH     constant binary_integer := 32;
MQ_APPL_IDENTITY_DATA_LENGTH   constant binary_integer := 32;
MQ_APPL_ORIGIN_DATA_LENGTH     constant binary_integer := 4;
MQ_ATTENTION_ID_LENGTH         constant binary_integer := 4;
MQ_AUTHENTICATOR_LENGTH        constant binary_integer := 8;
MQ_CANCEL_CODE_LENGTH          constant binary_integer := 4;
MQ_CLUSTER_NAME_LENGTH         constant binary_integer := 48;
MQ_CORREL_ID_LENGTH            constant binary_integer := 24;
MQ_CREATION_DATE_LENGTH        constant binary_integer := 12;
MQ_CREATION_TIME_LENGTH        constant binary_integer := 8;
MQ_DATE_LENGTH                 constant binary_integer := 12;
MQ_EXIT_NAME_LENGTH            constant binary_integer := 128;
MQ_FACILITY_LENGTH             constant binary_integer := 8;
MQ_FACILITY_LIKE_LENGTH        constant binary_integer := 4;
MQ_FORMAT_LENGTH               constant binary_integer := 8;
MQ_FUNCTION_LENGTH             constant binary_integer := 4;
MQ_GROUP_ID_LENGTH             constant binary_integer := 24;
MQ_LTERM_OVERRIDE_LENGTH       constant binary_integer := 8;
MQ_MFS_MAP_NAME_LENGTH         constant binary_integer := 8;
MQ_MSG_HEADER_LENGTH           constant binary_integer := 4000;
MQ_MSG_ID_LENGTH               constant binary_integer := 24;
MQ_MSG_TOKEN_LENGTH            constant binary_integer := 16;
MQ_NAMELIST_DESC_LENGTH        constant binary_integer := 64;
MQ_NAMELIST_NAME_LENGTH        constant binary_integer := 48;
MQ_OBJECT_INSTANCE_ID_LENGTH   constant binary_integer := 24;
MQ_NAME_LENGTH                 constant binary_integer := 48;
MQ_PROCESS_APPL_ID_LENGTH      constant binary_integer := 256;
MQ_PROCESS_DESC_LENGTH         constant binary_integer := 64;
MQ_PROCESS_ENV_DATA_LENGTH     constant binary_integer := 128;
MQ_PROCESS_NAME_LENGTH         constant binary_integer := 48;
MQ_PROCESS_USER_DATA_LENGTH    constant binary_integer := 128;
MQ_PUT_APPL_NAME_LENGTH        constant binary_integer := 28;
MQ_PUT_DATE_LENGTH             constant binary_integer := 8;
MQ_PUT_TIME_LENGTH             constant binary_integer := 8;
MQ_Q_DESC_LENGTH               constant binary_integer := 64;
MQ_Q_MGR_DESC_LENGTH           constant binary_integer := 64;
MQ_Q_MGR_IDENTIFIER_LENGTH     constant binary_integer := 48;
MQ_Q_MGR_NAME_LENGTH           constant binary_integer := 48;
MQ_Q_NAME_LENGTH               constant binary_integer := 48;
MQ_REMOTE_SYS_ID_LENGTH        constant binary_integer := 4;
MQ_SERVICE_NAME_LENGTH         constant binary_integer := 32;
MQ_SERVICE_STEP_LENGTH         constant binary_integer := 8;
MQ_START_CODE_LENGTH           constant binary_integer := 4;
MQ_STORAGE_CLASS_LENGTH        constant binary_integer := 8;
MQ_TIME_LENGTH                 constant binary_integer := 8;
MQ_TRAN_INSTANCE_ID_LENGTH     constant binary_integer := 16;
MQ_TRANSACTION_ID_LENGTH       constant binary_integer := 4;
MQ_TP_NAME_LENGTH              constant binary_integer := 64;
MQ_TRIGGER_DATA_LENGTH         constant binary_integer := 64;
MQ_USER_ID_LENGTH              constant binary_integer := 12;
```

Error Code Definitions

Error Code -29400: Data Cartridge Error

This error code indicates that the MQI opcode implemented in DG4MQ fails. Refer to IBM WebSphere reference manual for information about the cause by looking up the opcode and its completion code and reason code.

```
MQI opcode failed. completion code=xxxx. reason code=xxxx.
```

Example A–3   test.sql

```
--
-- Copyright Oracle, 2005 All Rights Reserved.
--
-- NAME
--   test.sql
--
-- DESCRIPTION
--
--   Specify the database link name you created for the gateway. To do this,
--   replace the database link name 'YOUR_DBLINK_NAME' with the dblink name
--   you chose when the database link was created.
--
--   This script performs a test run for the MQSeries gateway. In this
--   script the queuename is 'YOUR_QUEUE_NAME', replace queuename with
--   a valid queue name at the queue manager the gateway is configured
--   for.
--
--   First the script puts a raw message of 10 bytes on the specified
--   queue.
--
--   When successfully completed the put operation, the script does a
--   get on the same queue to read the message back.
--
--   The contents of both messages put and retrieved from the queue are
--   printed to standard out for verification by the user.
--
-- NOTES
--   Run the script from the SQL*Plus command line.
--
--   Make the sure the user is granted 'EXECUTE' on package dbms_output
--

set serveroutput on

declare

  objdesc     PGM.MQOD;
  hobj        PGM.MQOH;
  msgdesc     PGM.MQMD;
  putmsgopts PGM.MQPMO;
  getmsgopts PGM.MQGMO;
  options     binary_integer;
  putbuffer  raw(10) := '10203040506070809000';
  getbuffer  raw(10);

begin

  --
  -- Print the message we are putting on the queue
```

```
               --

               dbms_output.put_line('message put on queue = ' || rawtohex(putbuffer));

               --
               -- Specify queue name and dblink name (replace with proper names).
               --
               objdesc.objectname := 'YOUR_QUEUE_NAME';
               objdesc.dblinkname := 'YOUR_DBLINK_NAME';

               --
               -- Specify a put operation.
               --

               options := pgm_sup.MQOO_OUTPUT;

               --
               -- Open the queue.
               --

               PGM.MQOPEN(objdesc, options, hobj);

               --
               -- Put the message buffer on the queue.
               --

               PGM.MQPUT(hobj, msgdesc, putmsgopts, putbuffer);

               --
               -- Define close options.
               --

               options := pgm_sup.MQCO_NONE;

               --
               -- Close queue.
               --

               PGM.MQCLOSE(hobj, options);

               --
               -- Specify a get operation.
               --

               options := pgm_sup.MQOO_INPUT_AS_Q_DEF;

               --
               -- Open queue.
               --

               PGM.MQOPEN(objdesc, options, hobj);

               --
               -- Get message from the queue.
               --

               getmsgopts.msglength := 10;
               PGM.MQGET(hobj, msgdesc, getmsgopts, getbuffer);

               --
```

```
   -- Define close options.
   --

   options := pgm_sup.MQCO_NONE;

   --
   -- Close the queue again.
   --

   PGM.MQCLOSE(hobj, options);

   --
   -- Print the result
   --

   dbms_output.put_line('message read back = ' || rawtohex(getbuffer));

exception

   --
   -- When no more messages... tell the user and close the queue.
   --

   when pgm_sup.NO_MORE_MESSAGES then
      dbms_output.put_line('Warning: No message found on the queue');
      options := pgm_sup.MQCO_NONE;
      PGM.MQCLOSE(hobj, options);

   --
   -- something else went wrong.. tell the user.
   --
   when others then
      dbms_output.put_line('Error: Procedural Gateway for IBM MQSeries verification
 script failed.');
      dbms_output.put_line(SQLERRM);

end;
/
```

B

UTL_RAW Package

Use the Oracle Visual Workbench for developing applications that access WebSphere MQ through the gateway. The Oracle Visual Workbench defines an interface for accessing WebSphere MQ and automatically generates the PL/SQL code (the MIP) for Oracle applications to interface with the gateway. Refer to the *Oracle Procedural Gateway Visual Workbench for WebSphere MQ Installation and User's Guide for Microsoft Windows (32-Bit)* for more information about Oracle Visual Workbench. This appendix includes the following sections:

- Message Data Types on page B-1

- UTL_RAW Functions on page B-1

Message Data Types

Messages sent to a WebSphere MQ queue or retrieved from a WebSphere MQ queue are transferred as untyped data by the MIP procedures. When data profiles are defined in the MIP, the MIP converts message data from Oracle data types to target data types that the receiving application understands. The message data is packed into a buffer of the RAW data type before being sent to the WebSphere MQ queue. The same conversion process applies when receiving a message. The MIP unpacks the message from the buffer and converts it to specified Oracle data types.

The MIP uses the functions of the UTL_RAW package to perform the message data conversions. The UTL_RAW package is a PL/SQL package that contains procedures for converting and packing message data which is sent back and forth through the WebSphere MQ queues using the RAW data type and PL/SQL data types.

When necessary, you can enhance the message data conversions in the generated MIP with the UTL_RAW functions. When no data profiles are defined in the MIP, you can create your own data conversion procedures with UTL_RAW functions, calling these functions before sending a message and immediately after receiving a message.

The UTL_RAW package is not included with the gateway. It is shipped with each Oracle database. Refer to your Oracle DBA for information about installing the UTL_RAW package.

UTL_RAW Functions

The UTL_RAW functions are called with the following syntax:

```
UTL_RAW.function(arg1, arg2, ...)
```

The function name, arguments, their Oracle data types, and the return value data type are provided with each function description in this appendix. For ease of description,

the functions are described with PL/SQL syntax that shows the resulting function value placed in a variable as follows:

```
result := UTL_RAW.function(arg1, arg2, ...);
```

However, the function can also be used as a component in a PL/SQL expression. For example, the function takes two characters strings, *Hello* and *world!*, converts them to raw message data with UTL_RAW.CAST_TO_RAW, concatenates them with UTL_RAW.CONCAT, and uses the gateway to send them to a WebSphere MQ queue. The same message is retrieved from the queue, converted to a character data type with UTL_RAW.CAST_TO_VARCHAR2, and then printed.

UTL_RAW.TO_RAW

PGM_UTL.TO_RAW converts values of the PGM.MQOD, PGM.MQMD, PGM.MQPMO and PGM.MQGMO object to into raw values.

Syntax

```
result := PGM_UTL.TO_RAW(input);
```

where:

- *result* is a variable that holds the output value of the function. It is of the RAW data type.

- *input* is the input value of the PGM.MQOD, PGM.MQMD, PGM.MQPMO or PGM.MQGMO data type objects that is converted to raw data.

UTL_RAW.BIT_AND

UTL_RAW.BIT_AND performs a bitwise logical AND operation on two raw values. If the values have different lengths, then the AND operation is terminated after the last byte of the shorter of the two values. The unprocessed portion of the longer value is appended to the partial result to produce the final result. The length of the resulting value equals the longer of the two input values.

Syntax

```
result := UTL_RAW.BIT_AND(input1, input2);
```

where:

- result is the variable that holds the output value of the function. It is data type RAW. The value is null if *input1* or *input2* is null.

- input1 is an input value of data type RAW to BIT_AND with *input2*.

- input2 is an input value of data type RAW to BIT_AND with *input1*.

UTL_RAW.BIT_COMPLEMENT

UTL_RAW.BIT_COMPLEMENT performs a bitwise logical COMPLEMENT operation of a raw value. The length of the resulting value equals the length of the input value.

Syntax

```
result := UTL_RAW.BIT_COMPLEMENT(input);
```

where:

- `result` is the variable that holds the output value of the function. It is of `RAW` data type. The value is null if *input* is null.

- `input` is an input value of the `RAW` data type on which to perform the `COMPLEMENT` operation.

UTL_RAW.BIT_OR

`UTL_RAW.BIT_OR` performs a bitwise logical `OR` operation of two raw values. If the values have different lengths, then the `OR` operation is terminated after the last byte of the shorter of the two values. The unprocessed portion of the longer value is appended to the partial result to produce the final result. The length of the resulting value equals the length of the longer of the two input values.

Syntax

result := UTL_RAW.BIT_OR(*input1*, *input2*);

where:

- `result` is the variable that holds the output value of the function. It is of the data type `RAW`. The value is null if *input1* or *input2* is null.

- `input1` is an input value of the `RAW` data type to `BIT_OR` with *input2*.

- `input2` is an input value of the `RAW` data type to `BIT_OR` with *input1*.

UTL_RAW.BIT_XOR

`UTL_RAW.BIT_XOR` performs a bitwise logical `EXCLUSIVE OR` operation of two raw values. If the values have different lengths, then the `EXCLUSIVE OR` operation is terminated after the last byte of the shorter of the two values. The unprocessed portion of the longer value is appended to the partial result to produce the final result. The length of the resulting value equals the longer of the two input values.

Syntax

result := UTL_RAW.BIT_XOR(*input1*, *input2*);

where:

- `result` is the variable that holds the output value of the function. It is data type `RAW`. The value is null if *input1* or *input2* is null.

- `input1` is an input value of the `RAW` data type to `EXCLUSIVE OR` with *input2*.

- `input2` is an input value of the `RAW` data type to `EXCLUSIVE OR` with *input1*.

UTL_RAW.CAST_TO_RAW

`UTL_RAW.CAST_TO_RAW` converts a value of data type `VARCHAR2` into a raw value with the same number of bytes. The input value is treated as if it were composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is only changed to data type `RAW`.

Syntax

result := UTL_RAW.CAST_TO_RAW(*input*);

where:

- result is the variable that holds the output value of the function. It is data type RAW. The value is null if *input* is null.

- input is the input value of the VARCHAR2 data type to convert to raw data.

UTL_RAW.CAST_TO_VARCHAR2

UTL_RAW.CAST_TO_VARCHAR2 converts a raw value into a value of data type VARCHAR2 with the same number of data bytes. The result is treated as if it were composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is only changed to data type VARCHAR2.

Syntax

```
result := UTL_RAW.CAST_TO_VARCHAR2(input);
```

where:

- result is the variable that holds the output value of the function. It is data type VARCHAR2. The value is null if *input* is null.

- input is the input value of the RAW data type to convert to data type VARCHAR2.

UTL_RAW.COMPARE

UTL_RAW.COMPARE compares one raw value to another raw value. If they are identical, then UTL_RAW.COMPARE returns 0. If they are not identical, then COMPARE returns the position of the first byte that does not match. If the input values have different lengths, then the shorter input value is padded on the right by a value that you specify.

Syntax

```
result := UTL_RAW.COMPARE(input1, input2[, pad]);
```

where:

- result is the variable that holds the output value of the function. It is of data type NUMBER. A value of 0 is returned if the values of *input1* and *input2* are null or identical or the position, numbered from 1, of the first mismatched byte.

- input1 is the first input value of the RAW data type to compare.

- input2 is the second input value of the RAW data type to compare.

- padis a single byte value used to pad the shorter input value. The default is X'00'.

UTL_RAW.CONCAT

UTL_RAW.CONCAT concatenates a set of up to 12 raw values into a single raw value. The values are appended together, left to right, in the order that they appear in the parameter list. Null input values are skipped, and the concatenation continues with the next non-null value.

If the sum of the lengths of the input values exceeds 32 767 bytes, then a VALUE_ERROR exception is raised.

Syntax

```
result := UTL_RAW.CONCAT(input1, ...  input12);
```

where:

- `result` is the variable that holds the output value of the function. It is data type `RAW`.

- *input1 ... input12* are the input values of `RAW` data type to concatenate.

UTL_RAW.CONVERT

`UTL_RAW.CONVERT` converts a raw value to a different character set A `VALUE_ERROR` exception is raised for any of the following conditions:

- The input value is null or 0 in length

- One or both of the specified character sets is missing, null, or 0 in length

- The character set names are invalid or unsupported by the Oracle database

Syntax

result := UTL_RAW.CONVERT(*input*, *new_charset*, *old_charset*);

where:

- `result` is the variable that holds the output value of the function. It is of the `RAW` data type.

- `input` is the input value of the `RAW` data type to convert.

- `new_charset` is the Globalization Support character set to convert *input* to.

- `old_charset` is the Globalization Support character set that input is currently using.

UTL_RAW.COPIES

`UTL_RAW.COPIES` returns one or more copies of a value. The values are concatenated together. A `VALUE_ERROR` exception is raised for any of the following conditions:

- The input value is null or has a length of 0

- A negative number of copies is specified

- The length of the result exceeds 32 767 bytes

Syntax

result := UTL_RAW.COPIES(*input*, *number*);

where:

- `result` is the variable that holds the output value of the function. It is of the `RAW` data type.

- `input` is a value of the `RAW` data type to copy.

- `number` is the number of times to copy input. It must be a positive value.

UTL_RAW.LENGTH

`UTL_RAW.LENGTH` returns the length, in bytes, of a raw value.

Syntax

result := UTL_RAW.LENGTH(*input*);

where:

■   `result` is the output value of the function. It is of the `NUMBER` data type.

■   `input` is the input value of the RAW data type to evaluate.

UTL_RAW.OVERLAY

`UTL_RAW.OVERLAY` replaces a portion of a raw value with a new string of raw data. If the new data is shorter than the length of the overlay area, then the new data is padded to make it long enough. If the new data is longer than the overlay area, then the extra bytes are ignored. If you specify an overlay area that exceeds the length of the input value, then the input value is extended according to the length specified. If you specify a starting position for the overlay area that exceeds the length of the input value, then the input value is padded to the position specified, and then the input value is further extended with the new data.

A `VALUE_ERROR` exception is raised for any of the following conditions:

■   The new data used to overlay the input value is null or has a length of 0

■   The portion of the input value to overlay is not defined

■   The length of the portion to overlay exceeds 32 767 bytes

■   The number of bytes to overlay is defined as less than 0

■   The position within the input value to begin the overlay operation is defined as less than 1

Syntax

`result := UTL_RAW.OVERLAY(new_bytes, input, position, length, pad);`

where:

■   `result` is the variable that holds the output value of the function. It is of the `RAW` data type.

■   `new_bytes` is the new value, a byte string of the `RAW` data type, to overlay input with. Bytes are selected from `new_bytes` beginning with the leftmost byte.

■   `input` is the input value of data type `RAW` to overlay.

■   `position` is the position within input, numbered from 1, at which to begin overlaying. This value must be greater than 0. The default is 1.

■   `length` is the number of bytes to overlay. This must be greater than, or equal to, 0. The default is the length of `new_bytes`.

■   `pad` is a single byte value used to pad when length exceeds the overlay length or when position exceeds the length of `input`. The default is X'00'.

UTL_RAW.REVERSE

`UTL_RAW.REVERSE` reverses the byte sequence of a raw value from end-to-end. For example, this function reverses X'0102F3' to X'F30201' or xyz to zyx. The length of the resulting value is the same length as the input value. A `VALUE_ERROR` exception is raised if the input value is null or has a length of 0.

Syntax

`result := UTL_RAW.REVERSE(input);`

where:

■ `result` is the output value of the function. It is of the `RAW` data type.

■ `input` is the input value of the `RAW` data type to be reversed.

UTL_RAW.SUBSTR

`UTL_RAW.SUBSTR` removes bytes from a raw value. If you specify a positive number as the starting point for the bytes to remove, then `SUBSTR` counts from the beginning of the input value to find the first byte. If you specify a negative number, then `UTL_RAW.SUBSTR` counts backwards from the end of the input value to find the first byte.

A `VALUE_ERROR` exception is raised for any of the following conditions:

■ The position to begin the removal is specified as 0

■ The number of bytes to remove is specified as less than 0

Syntax

`result := UTL_RAW.SUBSTR(input, position[,length]);`

where:

■ `result` is the variable that holds the output value of the function. It is of the `RAW` data type. The value is the specified byte or bytes from `input`, or the value is a null value if `input` is null.

■ `input` is the input value of the `RAW` data type from which to extract a portion of its bytes.

■ `position` is the byte position from which to start extraction. This value cannot be 0. If the value of `position` is negative, then `SUBSTR` counts backwards from the end of input.

■ `length` is the number of bytes to extract from `input` after `position`. This value must be greater than 0. When not specified, all bytes to the end of input are returned.

UTL_RAW.TRANSLATE

`UTL_RAW.TRANSLATE` changes the value of some of the bytes in a raw value according to a scheme that you specify. Bytes in the input value are compared to a matching string, and when found to match, the byte at the same position in the replacement string is copied to the result. It is omitted from the result if the offset exceeds the length of the replacement string. Bytes in the input value that do not appear in the matching string are copied to the resulting value. Only the leftmost occurrence of a byte in the matching string is used, and subsequent duplicate occurrences are ignored.

If the matching string contains more bytes than the replacement string, then the extra bytes at the end of the matching string have no corresponding bytes in the replacement string. Any bytes in the input value that match such bytes are omitted from the resulting value.

A `VALUE_ERROR` exception is raised for any of the following conditions:

■ The input value is null or has a length of 0

■ The matching string is null or has a length of 0

■ The replacement string is null or has a length of 0

Syntax

result := UTL_RAW.TRANSLATE(*input*, *match*, *replace_bytes*);

where:

- `result` is the variable that holds the output value of the function. It is of data type `RAW`.

- `input` is the input value of data type `RAW` to change.

- `match` specifies the byte 0codes to search for in *input* and to change to *replace_bytes*. It is of data type `RAW`.

- `replace_bytes` specifies the byte codes that replace the codes specified by match. It is of data type `RAW`.

UTL_RAW.TRANSLITERATE

`UTL_RAW.TRANSLITERATE` replaces all occurrences of any bytes in a matching string with the corresponding bytes of a replacement string. Bytes in the input value are compared to the matching string, and if they are not found, then they are copied unaltered to the resulting value. If they are found, then they are replaced in the resulting value by the byte at the same offset in the replacement string, or with the pad byte that you specify when the offset exceeds the length of the replacement string. Only the leftmost occurrence of a byte in the matching string is used. Subsequent duplicate occurrences are ignored. The result value of `UTL_RAW.TRANSLITERATE` is always the same length as the input value.

If the replacement string is shorter than the matching string, then the pad byte is placed in the resulting value when a selected matching string byte has no corresponding byte in the replacement string. A VALUE_ERROR exception is raised when the input value is null or has a length of 0.

`UTL_RAW.TRANSLITERATE` differs from `UTL_RAW.TRANSLATE` in the following ways:

- Bytes in the input value that are undefined in the replacement string are padded with a value that you specify

- The resulting value is always the same length as the input value

Syntax

result := UTL_RAW.TRANSLITERATE (*input*, *replace_bytes*, *match*, *pad*);

where:

- `Result` is the output value of the function. It is data type `RAW`.

- `Input` is the input value of data type `RAW` to change.

- `Replace_bytes` specifies the byte codes to which corresponding bytes of match are changed. This value can be any length that is valid for the RAW data type. The default is a null value and effectively extends with pad to the length of match as necessary.

- `Match` specifies the `byte` codes to match in *input*. The value can be any length that is valid for the RAW data type. The default is X'00' through X'FF'.

- `Pad` is a singe byte value that is used to extend the length of *replace_bytes* when *replace_bytes* is shorter than match. The default is X'00'.

  `UTL_RAW.TRANSLATE` differs from the `UTL_RAW.TRANSLITERATE` function in the following ways:

- The raw values used for the matching and replacement strings have no default values

- Bytes in the input value that are undefined in the replacement string are omitted in the resulting value

- The resulting value can be shorter than the input value

UTL_RAW.XRANGE

UTL_RAW.XRANGE returns a raw value containing all valid one-byte codes within a range that you specify. If the starting byte value is greater than the ending byte value, then the succession of resulting bytes begin with the starting byte, wrapping from X'FF' to X'00', and end at the ending byte.

When specified, the values for the starting and ending bytes must be single-byte raw values.

Syntax

result := UTL_RAW.XRANGE(*start*, *end*);

where:

- result is the output value of the function. It is of data type RAW.

- start is a single byte code to start with. The default is X'00'.

- end is a single byte code to end with. The default is X'FF'.

C
Gateway Initialization Parameters

The gateway has its own initialization parameters, which are described in this appendix, and supports the initialization parameters for Oracle Database Gateways.

This appendix contains the following sections:

- Gateway Initialization File on page C-1
- Gateway Parameters on page C-1

Gateway Initialization File

The gateway initialization file is called init*sid*.ora. A default initialization file is created in the directory *ORACLE_HOME*\dg4mq\admin on Microsoft Windows and *ORACLE_HOME*/dg4mq/admin on UNIX based systems during the installation of the Oracle Database Gateway for WebSphere MQ.

Gateway Parameters

This section describes gateway parameters, listing the default value, range of values, and the syntax for usage. This section describes the following parameters:

- LOG_DESTINATION
- AUTHORIZATION_MODEL
- QUEUE_MANAGER
- TRACE_LEVEL
- TRANSACTION_LOG_QUEUE
- TRANSACTION_MODEL
- TRANSACTION_RECOVERY_PASSWORD
- TRANSACTION_RECOVERY_USER

LOG_DESTINATION

The following table describes the LOG_DESTINATION parameter:

LOG_DESTINATION	Use
Syntax	LOG_DESTINATION = *log_file*
Default value	SID_agt_PID.trc (PID is the process ID of the gateway)
Range of values	None

LOG_DESTINATION specifies the full path name of the gateway log file.

AUTHORIZATION_MODEL

The following table describes how to use the AUTHORIZATION_MODEL parameter:

AUTHORIZATION_MODEL	Use
Syntax	AUTHORIZATION_MODEL = {RELAXED|STRICT}
Default value	RELAXED
Range of values	RELAXED or STRICT

AUTHORIZATION_MODEL defines the authorization model for the gateway user. The relaxed model specifies that authorizations that are granted to the effective user ID of the gateway by the queue manager are the only associations that an Oracle application has.

The strict model specifies that the Oracle user ID and password (that are provided when a database link is created), or the current user ID and password (when the Oracle user ID and password are not provided), should be checked against the local or network password file.

Refer to "Security Models" on page 8-1 for more information about effective user IDs.

QUEUE_MANAGER

The following table describes the QUEUE_MANAGER parameter:

QUEUE_MANAGER	Use
Syntax	QUEUE_MANAGER = manager_name
Default value	None
Range of values	None

QUEUE_MANAGER, a required parameter, specifies the name of the queue manager that the gateway connects to at logon time. The effective user ID of the gateway should have the correct user privileges or should be authorized to connect to this queue manager. Specify manager_name using the following rules:

- 1 to 48 alphanumeric characters in length
- No leading or embedded blank characters
- Trailing blank characters are permitted

Refer to "Security Models" on page 8-1 for more information about effective user IDs.

TRACE_LEVEL

The following table describes the TRACE_LEVEL parameter:

TRACE_LEVEL	Use
Syntax	TRACE_LEVEL = level
Default value	0
Range of values	0 to 7

TRACE_LEVEL controls whether tracing information is collected as the gateway runs. When set to collect information, the trace data is written to the log file that is specified by the LOG_DESTINATION parameter. Specify *level* as an integer from 0 to 3, which is the sum of the desired trace values. The following table describes the significance of these values:

Trace Level	Description
0	Specifies that no tracing is to be done.
1	Specifies that general tracing is to be done. This includes the user ID that is used to log on to the WebSphere MQ queue manager, the name of the queue manager, the gateway transaction mode, security mode, and so on.
2	Specifies that tracing is to be done for all MQI calls that are issued by the gateway.
3	Specifies that tracing is to be done for all parameter values that are passed to, or received from, the MQI calls that were issued by the gateway.

For more information about MQI calls.

> **See Also:** Refer to IBM publications, for more information about MQI calls.

TRANSACTION_LOG_QUEUE

The following table describes how to use TRANSACTION_LOG_QUEUE.

TRANSACTION_LOG_QUEUE	Description
Syntax	TRANSACTION_LOG_QUEUE = *tx_queue_name*
Default value	None
Range of values	None

TRANSACTION_LOG_QUEUE specifies the name of the queue for logging transaction IDs. Specify *tx_queue_name* using the following rules:

- 1 to 48 alphanumeric characters in length

- No leading or embedded blank characters

- Trailing blank characters are permitted

Refer to "Creating a Transaction Log Queue" on page 7-10 for more information.

TRANSACTION_MODEL

The following table describes how to use TRANSACTION_MODEL.

TRANSACTION_MODEL	Description
Syntax	TRANSACTION_MODEL = {COMMIT_CONFIRM	SINGLE_SITE}
Default value	SINGLE_SITE
Range of values	COMMIT_CONFIRM or SINGLE_SITE

TRANSACTION_MODEL defines the transaction mode of the gateway. Specify a value for TRANSACTION_MODEL as described in the following table:

Item	Description
COMMIT_CONFIRM	Specifies that the gateway can participate in transactions when queues belonging to the same WebSphere queue manager are updated. At the same time, any number of Oracle databases are updated. Only one gateway with the commit-confirm model can join the distributed transaction, because the gateway operates as the focal point of the transaction.
	When this value is specified, you must also set the RECOVERY_USER, RECOVERY_PASSWORD, and TRANSACTION_LOG_QUEUE parameters.
SINGLE_SITE	Specifies that the gateway can participate in a transaction only when queues belonging to the same WebSphere queue manager are updated. An Oracle application can select, but not update, data at any Oracle database within the same transaction that accesses WebSphere MQ.

TRANSACTION_RECOVERY_PASSWORD

The following table describes TRANSACTION_RECOVERY_PASSWORD.

TRANSACTION_RECOVERY_PASSWORD	Description
Default value	*
Range of values	An asterisk (*), which indicates that the parameter must be encrypted, or any valid password
Syntax	TRANSACTION_RECOVERY_PASSWORD = rec_password
	or
	TRANSACTION_RECOVERY_PASSWORD = *

TRANSACTION_RECOVERY_PASSWORD specifies the password of the user that the gateway uses to start recovery of a transaction. The default value is set to an asterisk (*), and this asterisk indicates that the value of this parameter is stored in an encrypted form in a separate password file. To specify or change a valid password for encrypted gateway parameters, you need to use the dg4pwd gateway utility to do the work. For more information, refer to "Using the dg4pwd Utility".

The TRANSACTION_RECOVERY_PASSWORD parameter is required only when TRANSACTION_MODEL is set to COMMIT_CONFIRM. Refer to "Creating a Transaction Log Queue" on page 7-10 for more information.

Passwords in the Gateway Initialization File

The gateway uses user IDs and passwords to access information in the remote database on the WebSphere MQ server. Some user IDs and passwords must be defined in the gateway initialization file to handle functions such as resource recovery. In a security-conscious environment, plain-text passwords are regarded as insecure when they are accessible in the initialization file. A new encryption feature has been added to the gateway to make such passwords more secure. The dg4pwd utility can be used to encrypt passwords that would normally be stored in the gateway initialization file. Using this feature is optional, but highly recommended by Oracle. With this feature, passwords are no longer stored in the initialization file but are stored in a

password file in an encrypted form. This makes the password information more secure. The following section describes how to use this feature.

Using the dg4pwd Utility

The `dg4pwd` utility is used to encrypt passwords that would normally be stored in the gateway initialization file. The utility works by reading the initialization file and looks for parameters with a special value. The value is the asterisk (*). The asterisk indicates that the value of this parameter is stored in an encrypted form in another file. The following sample is a section of the initialization file with this value.

TRANSACTION_RECOVERY_PASSWORD=*

The initialization file is first edited to set the value of the parameter to the asterisk (*). Then the `dg4pwd` utility is run, specifying the gateway SID on the command line. The utility reads the initialization file and prompts the user to enter the values to be encrypted.

The syntax of this command is:

```
dg4pwd gateway_sid
```

In this command, `gateway_sid` is the SID of the gateway.

The following is an example, assuming that the gateway SID is `dg4mqs`:

```
% dg4pwd dg4mqs
ORACLE Gateway Password Utility (dg4mqseries) Constructing password file for
Gateway SID dg4mqs
Enter the value for TRANSACTION_RECOVERY_PASSWORD
welcome
%
```

In this example, the `TRANSACTION_RECOVERY_PASSWORD` parameter is identified as requiring encryption. The user enters the value (for example, `welcome`) and presses the Enter key. If more parameters require encryption, then you are prompted for their values. The encrypted data is stored in the `dg4mq\admin` directory on Microsoft Windows and `dg4mq/admin` directory on UNIX based systems.

Note: It is important that the `ORACLE_HOME` environment variable specifies the correct gateway home to ensure that the correct gateway initialization file is read.

TRANSACTION_RECOVERY_USER

The following table describes how to use the `TRANSACTION_RECOVERY_USER` parameter:

Item	Description
Syntax	`TRANSACTION_RECOVERY_USER = rec_user`
Default value	None.
Range of values	Any valid operating system user ID that is authorized by WebSphere MQ Manager (MQM)

`TRANSACTION_RECOVERY_USER` specifies the user name that the gateway uses to start the recovery of a transaction. This parameter is required only when `AUTHORIZATION_MODEL` is set to `STRICT`, and `TRANSACTION_MODEL` is set to `COMMIT_CONFIRM`. Refer to "Creating a Transaction Log Queue" on page 7-10 for more information.

Index

utlraw.sql script, 7-13

V

variable
  environment
    MCAUSER, 8-2
    MQ_PASSWORD, 8-2
    MQ_USER_ID, 8-2
Visual Workbench
  deinstall repository, 7-16
  description, 1-3
  development privileges for repository, 7-15
  installing the repository, 7-12
  MIP, A-1
Visual Workbench Repository
  upgrade, 7-13

W

WebSphere MQ
  access authorization, 8-2
  client configuration definition, 1-2
  common error messages, 8-6
  description, 1-2
  error codes, 8-6
  queue manager definition, 1-2
  triggers, 1-2
  WebSphere MQ server, 1-6