

Oracle® Application Server InterConnect

User's Guide

10g (9.0.4)

Part No. B10404-01

August 2003

Oracle Application Server InterConnect User's Guide, 10g (9.0.4)

Part No. B10404-01

Copyright © 2002, 2003 Oracle Corporation. All rights reserved.

Contributing Authors: Rahul Pathak, Herb Stiel, Jeff Hutchins, Maneesh Joshi, Bo Stern

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle*MetaLink*, Oracle Store, Oracle*9i*, Oracle*9iAS Discoverer*, SQL*Plus, and PL/SQL are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Send Us Your Comments	xi
Preface.....	xiii
1 Getting Started with OracleAS InterConnect	
What is OracleAS InterConnect?	1-2
OracleAS InterConnect Components	1-3
Standard Messaging.....	1-5
OracleAS InterConnect Integration Process.....	1-7
Design Time.....	1-7
Runtime.....	1-7
Separation of Integration Logic and Platform Functionality	1-8
Unique Integration Methodology	1-9
Integration Lifecycle Management	1-11
Using Adapters for Integration	1-12
2 Using iStudio	
Overview of iStudio.....	2-2
iStudio Concepts.....	2-2
Starting iStudio	2-7
Parts of the iStudio Window	2-8
Menu Bar.....	2-9
Toolbar	2-11
Design Navigation Tree.....	2-12

Deploy Navigation Tree	2-12
Context Menus	2-13
Detail View	2-13
Using Workspaces in iStudio	2-13
Creating a New Workspace.....	2-13
Opening an Existing Workspace	2-14
Using Projects in iStudio	2-14
Creating a New Project	2-15
Opening an Existing Project.....	2-16

3 Creating Applications, Common Views, and Business Objects

Overview of Applications	3-2
Application View	3-2
Application Data Types	3-2
Creating an Application.....	3-2
Overview of Common Views and Business Objects	3-3
Defining Common Views	3-3

4 Using Events in iStudio

Overview of Events	4-2
Event Maps	4-2
Creating Events	4-3
Publishing and Subscribing to an Event	4-4
Publishing an Event.....	4-5
Subscribing to an Event	4-12

5 Using Procedures in iStudio

Using Procedures	5-2
Creating a Procedure.....	5-3
Invoking and Implementing a Procedure	5-4
Invoking a Procedure	5-5
Implementing a Procedure	5-12
Exporting Stored Procedures	5-18

6 Enabling Infrastructure

Enabling Infrastructure	6-2
Content-Based Routing	6-2
Cross Reference Tables	6-2
Domain Value Maps.....	6-2
Working with Content-Based Routing	6-2
Modifying Content-Based Routing	6-2
Working with Cross Reference Tables	6-7
Creating Cross-Reference Tables.....	6-7
Adding Applications to Cross Reference Tables	6-7
Removing Applications From Cross Reference Tables.....	6-7
Populating Cross Reference Tables.....	6-8
Working with Domain Value Mappings	6-9
Creating a Domain Value Mapping Table	6-9
Adding Applications to Domain Value Mappings.....	6-9
Removing Applications From Domain Value Mappings	6-9
Modifying Domain Value Mappings.....	6-10
Deleting Domain Value Mappings	6-10
Deleting Domain Value Mapping Tables.....	6-10
Modifying Attribute Mappings	6-11
Removing Attribute Mappings.....	6-11
Adding Custom Transformations	6-11
Deleting Custom Transformations.....	6-12
Adding Mapping Variables	6-12
Deleting a Mapping Variable.....	6-13
Cross Reference Table Walk-Through	6-13
Domain Value Mappings Walk-Through	6-14

7 Using Oracle Workflow

Oracle Workflow Overview	7-2
Oracle Workflow Solves Common Business Problems.....	7-2
OracleAS InterConnect Integration with Oracle Workflow	7-4
Design Time Tools	7-4
Runtime.....	7-7
Using Oracle Workflow to Apply Business Logic	7-8

Install Oracle Workflow Components	7-8
Design Business Process	7-8
Deploy Business Processes for Runtime.....	7-8
Design Business Process	7-8
Process Bundle	7-9
Business Process.....	7-9
Activity	7-9
Creating a Process Bundle	7-11
Creating a Business Process	7-11
Populating a Business Process with Activities	7-11
Deploying to Oracle Workflow.....	7-13
Launching Oracle Workflow Tools.....	7-16
Modifying Existing Oracle Workflow Processes.....	7-18

8 Runtime System Concepts and Components

Integration Architecture	8-2
Features	8-2
Messaging Paradigms	8-2
Message Delivery.....	8-3
Message Retention.....	8-3
Routing Support.....	8-3
Error Management.....	8-4
Scalability and Load Balancing.....	8-4
File Persistence	8-6
Components	8-7
Adapters.....	8-7
Repository	8-9
Advanced Queues.....	8-10
Oracle Workflow	8-10
Real Application Clusters Configuration	8-10
Introduction.....	8-10
OracleAS InterConnect Adapters Supporting RAC	8-10
Configuration	8-11
Sample hub.ini File	8-12
Sample Database Adapter adapter.ini File that Shows the Spoke Database Entry	8-12

9 Runtime Management

Introduction to Runtime Management	9-2
Starting Oracle Enterprise Manager Console	9-2
Features	9-4
Common Features for Adapters and the Repository	9-4
Repository Specific Features	9-5
Adapter Specific Features.....	9-5

A Integration Scenario

Integration Scenario Overview	A-2
The New Centralized System	A-2
The Legacy System	A-2
The Integration Scenario.....	A-3
Modeling the Integration	A-4
Integration Modeling using iStudio.....	A-5
Implementing the Scenario	A-6
Review Legacy System Database Trigger	A-6
Create a Project	A-7
Create the Common View Business Object	A-8
Create Business Object Events	A-8
Create Applications.....	A-11
Create a Cross Reference Table	A-12
Create Publish Events	A-13
Subscribing to Events.....	A-16
Create Content Based Routing.....	A-25
Create an Oracle Workflow Process Bundle.....	A-26
Deploy the Process Bundle to Oracle Workflow.....	A-29
Creating Objects in Oracle Workflow for Modeling	A-31
Modeling Business Logic in Oracle Workflow	A-35
Deployment	A-39
Setting Queues	A-39
Pushing Metadata.....	A-40
Exporting and Installing Code.....	A-40
Conclusion	A-42

B Using the Data Definition Description Language

About D3L	B-2
What Is D3L?	B-2
When Is D3L Used?	B-3
Native Format Message and D3L File Example	B-4
Native Format Message Contents Description in a D3L File	B-4
Native Format Message Configuration with a D3L File	B-5
D3L File Structure	B-9
Supported D3L Data Types	B-12
Comma-Separated Values File Parsing with D3L.....	B-27
D3L Integration with OracleAS InterConnect Adapters	B-31
Runtime Initialization	B-31
Native Format Message to Common View Incoming Message Translations	B-32
Common View to Native Format Message Outgoing Messages Translations	B-34
Installing D3L	B-36
Configuring D3L	B-37
Task 1: Configure D3L with iStudio.....	B-37
Task 2: Create a Native Format Message	B-37
Task 3: Create a D3L File Describing the Native Format Message.....	B-38
Task 4: Configure a Native Format Message with a D3L File.....	B-38
Task 5: Configure D3L with OracleAS InterConnect Adapters	B-39
Task 6: Import a D3L File in iStudio	B-40
Task 7: Define Metadata Properties with Each Event (Optional)	B-41
D3L Use Case	B-42
D3L Use Case Overview	B-43
Creating Data Type Definitions for Application Views.....	B-43
Configuring the aqapp_pub and fileapp_sub Applications in iStudio	B-46
Installing the Advanced Queuing and FTP Adapters.....	B-58
Running the D3L Use Case.....	B-62
Using Other Adapters and XML Mode	B-66
Additional D3L Sample Files and DTD	B-68
Additional D3L Sample Files	B-68
D3L DTD	B-75

C Transformations

Copy Fields	C-2
Copy Object	C-2
Concat Fields	C-2
Expand Fields.....	C-2
Set Constant.....	C-3
True Conditional Lookup XRef	C-3
True Conditional Lookup DVM	C-3
Conditional Copy	C-4
True Conditional Copy	C-4
True Conditional Concat	C-4
True Conditional To Number	C-5
False Conditional Copy	C-5
False Conditional Concat.....	C-5
False Conditional To Number	C-6
To Number	C-6
Substring	C-7
Char Replace.....	C-7
String Replace.....	C-7
LTrim	C-7
RTrim.....	C-8
LPad.....	C-8
RPad.....	C-8
Lookup XRef.....	C-9
Delete XRef	C-9
Lookup DVM	C-9
Truncate	C-10
Increment	C-10
DatabaseOperation.....	C-10

Index

Send Us Your Comments

Oracle Application Server InterConnect User's Guide, 10g (9.0.4)

Part No. B10404-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- Fax: (650) 506-7356 Attn: Oracle Application Server Documentation Manager
- Postal service:
Oracle Corporation
Oracle Application Server Documentation Manager
500 Oracle Parkway, M/S 10p6
Redwood Shores, CA 94065 USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface contains these topics:

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

Audience

This guide is targeted at the following types of users:

- Business analysts and integration engineers, for iStudio.
- System Administrators, for the runtime component.

The audience should have the following pre-requisites, which are discussed but not explained:

- Domain knowledge of the applications being integrated.
- Database concepts and working knowledge of SQL, PL/SQL, or SQL* Plus.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Chapter 1, "Getting Started with OracleAS InterConnect"

Introduces OracleAS InterConnect and presents an overview of the product and the tools.

Chapter 2, "Using iStudio"

Describes iStudio and how to create workspaces and projects.

Chapter 3, "Creating Applications, Common Views, and Business Objects"

Describes how to create and manage applications, common views, and business objects using iStudio.

Chapter 4, "Using Events in iStudio"

Describes using iStudio to create, publish, and subscribe to events.

Chapter 5, "Using Procedures in iStudio"

Describes using iStudio to create, invoke, and implement procedures.

Chapter 6, "Enabling Infrastructure"

Describes the enabling infrastructure tasks in iStudio including creating cross reference tables and domain value mappings.

Chapter 7, "Using Oracle Workflow"

Describes how OracleAS InterConnect works with Oracle Workflow.

Chapter 8, "Runtime System Concepts and Components"

Describes the runtime components and concepts of OracleAS InterConnect.

Chapter 9, "Runtime Management"

Introduces the Runtime Management Console and describes how you use it to manage your integration environment.

Appendix A, "Integration Scenario"

Provides an integration scenario and model based on a fictitious company, Acme, Inc. using OracleAS InterConnect.

Appendix B, "Using the Data Definition Description Language"

Describes how to use the data definition description language (D3L) in native format message-to-application view and application view-to-native format message translations.

Appendix C, "Transformations"

Provides a list of OracleAS InterConnect transformations.

Related Documentation

For more information, see these Oracle resources:

- Oracle Application Server Documentation Library CD-ROM
- Oracle Application Server Platform Specific Documentation on Oracle Application Server Disk 1
- *Oracle Application Server InterConnect Installation Guide*
- *Oracle Application Server InterConnect Release Notes*

In North America, printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com/>

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

<http://www.oraclebookshop.com/>

Other customers can contact their Oracle representative to purchase printed documentation.

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

<http://technet.oracle.com/membership/index.htm>

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

<http://technet.oracle.com/docs/index.htm>

Conventions

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

- [Conventions in Text](#)
- [Conventions in Code Examples](#)
- [Conventions for Microsoft Windows Operating Systems](#)

Conventions in Text

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

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an index-organized table .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle9i Database Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> reside on the same disk.
UPPERCASE monospace (fixed-width) font	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.

Convention	Meaning	Example
lowercase monospace (fixed-width) font	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter <code>sqlplus</code> to open SQL*Plus. The password is specified in the <code>orapwd</code> file. Back up the datafiles and control files in the <code>/disk1/oracle/dbs</code> directory. The <code>department_id</code> , <code>department_name</code> , and <code>location_id</code> columns are in the <code>hr.departments</code> table. Set the <code>QUERY_REWRITE_ENABLED</code> initialization parameter to <code>true</code> . Connect as <code>oe</code> user. The <code>JRepUtil</code> class implements these methods.
<i>lowercase italic monospace (fixed-width) font</i>	Lowercase italic monospace font represents placeholders or variables.	You can specify the <i>parallel_clause</i> . Run <code>Uold_release.SQL</code> where <i>old_release</i> refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[]	Brackets enclose one or more optional items. Do not enter the brackets.	<code>DECIMAL (digits [, precision])</code>
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	<code>{ENABLE DISABLE}</code>
	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	<code>{ENABLE DISABLE}</code> <code>[COMPRESS NOCOMPRESS]</code>

Convention	Meaning	Example
...	Horizontal ellipsis points indicate either: <ul style="list-style-type: none"> That we have omitted parts of the code that are not directly related to the example That you can repeat a portion of the code 	<pre>CREATE TABLE ... AS subquery; SELECT col1, col2, ... , coln FROM employees;</pre>
.	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	<pre>acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;</pre>
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	<pre>CONNECT SYSTEM/system_password DB_NAME = database_name</pre>
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	<pre>SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;</pre>
lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	<pre>SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;</pre>

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - <i>HOME_NAME</i> > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	c:\winnt "\ "system32 is the same as C:\WINNT\SYSTEM32
C:\>	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the <i>command prompt</i> in this manual. The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	C:\oracle\oradata> C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\" C:\>imp SYSTEM/ <i>password</i> FROMUSER=scott TABLES=(emp, dept)
<i>HOME_NAME</i>	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	C:\> net start Oracle <i>HOME_NAME</i> TNSListener

Convention	Meaning	Example
<i>ORACLE_HOME</i> and <i>ORACLE_BASE</i>	<p>In releases prior to Oracle8i release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level <i>ORACLE_HOME</i> directory that by default used one of the following names:</p> <ul style="list-style-type: none"> ■ C:\orant for Windows NT ■ C:\orawin95 for Windows 95 ■ C:\orawin98 for Windows 98 <p>This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level <i>ORACLE_HOME</i> directory. There is a top level directory called <i>ORACLE_BASE</i> that by default is C:\oracle. If you install Oracle9i release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is C:\oracle\ora90. The Oracle home directory is located directly under <i>ORACLE_BASE</i>.</p> <p>All directory path examples in this guide follow OFA conventions.</p> <p>Refer to <i>Oracle8i Quick Reference for Windows</i> for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.</p>	Go to the <i>ORACLE_BASE\ORACLE_HOME\rdms\admin</i> directory.

Getting Started with OracleAS InterConnect

This chapter provides an overview of Oracle Application Server InterConnect (OracleAS InterConnect), its features, and components.

Topics include:

- [What is OracleAS InterConnect?](#)
- [Standard Messaging](#)
- [OracleAS InterConnect Integration Process](#)

What is OracleAS InterConnect?

OracleAS InterConnect is an integral component of Oracle Application Server and provides a comprehensive application integration framework to enable seamless integration of enterprise software. It is built on top of the Oracle Application Server integration platform and leverages its underlying functionalities. It is designed to integrate heterogeneous systems, such as Oracle Applications, non-Oracle Applications, or third party messaging oriented middleware. This integration can be deployed within an enterprise or across enterprise boundaries through the Internet.

The technical design goals for OracleAS InterConnect are to:

- Elevate the integration problem from a technical coding exercise to a functional modeling exercise, thereby reducing or eliminating the programming effort normally associated with integration.
- Develop and expose an integration methodology that promotes reuse, and reduces the complexity and management issues that arise over the software lifecycle.

OracleAS InterConnect provides a complete framework for e-Business application integration across the Application-to-Application, Application Service Provider, and Business-to-Business domains. OracleAS InterConnect components can be deployed for these domains in the following methods:

- Application-to-Application—Applications are distributed within a Local Area Network or across a Wide Area Network. OracleAS InterConnect is deployed within the organization to integrate these applications.
- Application Service Provider—Applications are distributed across firewall boundaries with some applications residing inside the Application Service Provider firewall and others inside the customer firewall. OracleAS InterConnect can be deployed inside either one of the two firewalls, or both, to integrate these applications across the firewalls.
- Business-to-Business—Similar to the Application Service Provider model only the Application Service Provider is replaced with another business.

OracleAS InterConnect Components

OracleAS InterConnect has the following core components:

- [OracleAS InterConnect Hub](#)
- [OracleAS InterConnect Adapters](#)
- [OracleAS InterConnect Development Kit](#)

OracleAS InterConnect Hub

The Hub consists of a middle tier repository server program communicating with a back-end database. The repository has the following functionality:

- At design time, all integration logic defined in iStudio is stored in tables in the repository as metadata.
- At runtime, the repository provides access to this metadata for adapters to integrate applications.

The repository server is deployed as a stand-alone Java application running outside the database. The repository schema is a set of tables in the Oracle Application Server Infrastructure.

OracleAS InterConnect Adapters

Adapters have two major tasks:

- Provide connectivity between an application and the hub.
- Transform and route messages between the application and the hub.

Adapters are deployed as stand-alone Java applications running outside the database. Adapters are physically located with the applications they connect to, either on the same machine as the application itself or on a separate machine. Adapters are usually not deployed on the hub machine.

See Also: ["Using Adapters for Integration"](#) on page 1-12

OracleAS InterConnect Development Kit

iStudio is a design time integration specification tool targeted at business analysts. This tool helps business analysts specify the integration logic at a functional level, instead of a technical coding level. iStudio exposes the integration methodology using simple wizards and reduces, or eliminates, the need for writing code to specify the integration logic. This reduces the total time required to complete an integration.

iStudio is a multi-user tool with fine-grained locking for all OracleAS InterConnect first class objects. Therefore, multiple users can work simultaneously on the same integration scenario without compromising the consistency of the metadata.

iStudio allows business analysts to complete the following tasks:

- Define data to be exchanged across applications.
- Semantically map data across applications.
- Define the business process collaboration across applications using Oracle Workflow and associate the semantic maps with business processes, if required.
- Configure and deploy the integration.

iStudio is deployed as a stand-alone Java application running outside the database and can be deployed on any machine with access to the hub machine running Windows.

See Also: [Chapter 2, "Using iStudio"](#)

OracleAS InterConnect SDKs OracleAS InterConnect SDK allows you to customize OracleAS InterConnect to meet your integration needs.

iStudio SDK The iStudio SDK is a collection of Java jar and Javadoc files usually deployed on the same machine as iStudio. The iStudio SDK is only available on Windows. Using the iStudio SDK and Java, users can build the following:

- New transformation functions.
- New browsers to import application-native data structures and APIs into iStudio.

Documentation and samples are provided with the iStudio SDK.

Adapter SDK The Adapter SDK is a collection of Java jar and Javadoc files that can be deployed on any machine. The Adapter SDK is available on all tier one platforms. Using the Adapter SDK, you can write new adapters in Java for applications or protocols not currently supported by OracleAS InterConnect. Specifically, only the bridge sub-component must be written. The agent is a generic engine already written and is part of each adapter.

Documentation and samples are provided with the Adapter SDK.

Oracle Workflow Oracle Workflow provides a comprehensive business process management system that enables traditional workflow applications, as well as

process collaboration in a single solution. Using Oracle Workflow Business Event System, OracleAS InterConnect can model an integration solution on business processes. With OracleAS InterConnect and Oracle Workflow, business collaborations across two or more applications can be defined to implement the organization's business processes.

Standard Messaging

OracleAS InterConnect provides all the basic services expected of a messaging middleware platform including:

- **Guaranteed delivery of messages**—All messages have guaranteed delivery end-to-end. Messages are delivered exactly once and in the order sent.
- **Scalability**—Multiple adapters are instantiated to serve one application. The hub runs in an Oracle Real Application Clusters (RAC) environment.
- **Load Balancing**—Messages can be partitioned based on load between multiple adapters servicing one application. One or more adapters can serve all messages for one application. In addition, one or more adapters can be dedicated per integration point in which the application participates.
- **Runtime Management**—The Oracle Enterprise Manager Console helps manage the integration scenario and components at runtime. The Console allows users to start and stop components, monitor message flow, detect problems, and manage errors.
- **Deployment Support**—The messaging hub consists of Advanced Queues that are configured for runtime. Number of queues to create, naming these queues, and matching adapters with messages in a specific named queue can be configured using iStudio.

The following supplementary features do not require any additional coding:

- **Content Based Routing**—Route messages by building business rules based on message content. For example, a procurement system routes fulfillment requests to different fulfillment centers based on an originating location.
- **Cross Referencing**—Correlate keys that uniquely identify the entities in one application with corresponding entities created in other applications. For example, a purchase order created in a procurement system has a native ID *X*. It is then routed to a fulfillment system and the purchase order is created in the fulfillment system with native ID *Y*. Therefore, *X* and *Y* must be cross referenced for OracleAS InterConnect to correlate communication about this same logical

entity in two different systems without each system understanding the native ID of the other system.

- **Domain Value Mapping**—Map code tables across systems. For example, a purchase order in a procurement system has a PO Status field with possible domain values `Booked` and `Shipped`. The corresponding field in a fulfillment system has the possible domain values 1 and 2. OracleAS InterConnect allows the user to create the mappings `booked=1`, `shipped=2` so that it can correlate these values at runtime without each system understanding the domain value set of the other system.

Supported Messaging Paradigms

OracleAS InterConnect supports the following messaging paradigms. These paradigms are defined in iStudio at design time. The definitions are used at runtime to route the messages appropriately.

- **Publish/Subscribe Messaging**—An application publishes a message if it sends data out to the OracleAS InterConnect hub without knowing the destination applications. Furthermore, data is not expected in return. An application subscribes to a message if it receives the data from the OracleAS InterConnect hub regardless of who sent the data. Furthermore, it does not send any data out in return. Events in iStudio are used to model this paradigm.
- **Request/Reply Messaging**—An application publishes a message and expects a message in return as a reply. The application subscribing to the request sends a reply back to the sender after processing the request. Procedures in iStudio are used to model this paradigm. Request/Reply has the following two types of characteristics:
 - **Synchronous**—The application making the request is blocked until it receives a reply.
 - **Asynchronous**—The application makes the request and proceeds with normal processing. It does not wait for a response. A reply is delivered next and is consumed by the application.
- **Point-to-Point Messaging**—Both Publish/Subscribe and Request/Reply can acquire a point-to-point characteristic if the sending application explicitly calls out which application should receive the message. This can be modeled using content-based routing in iStudio.

OracleAS InterConnect Integration Process

Application integration using OracleAS InterConnect involves the following two phases:

- [Design Time](#)
- [Runtime](#)

[Figure 1-1](#) provides an overview of design time and runtime phases in integration.

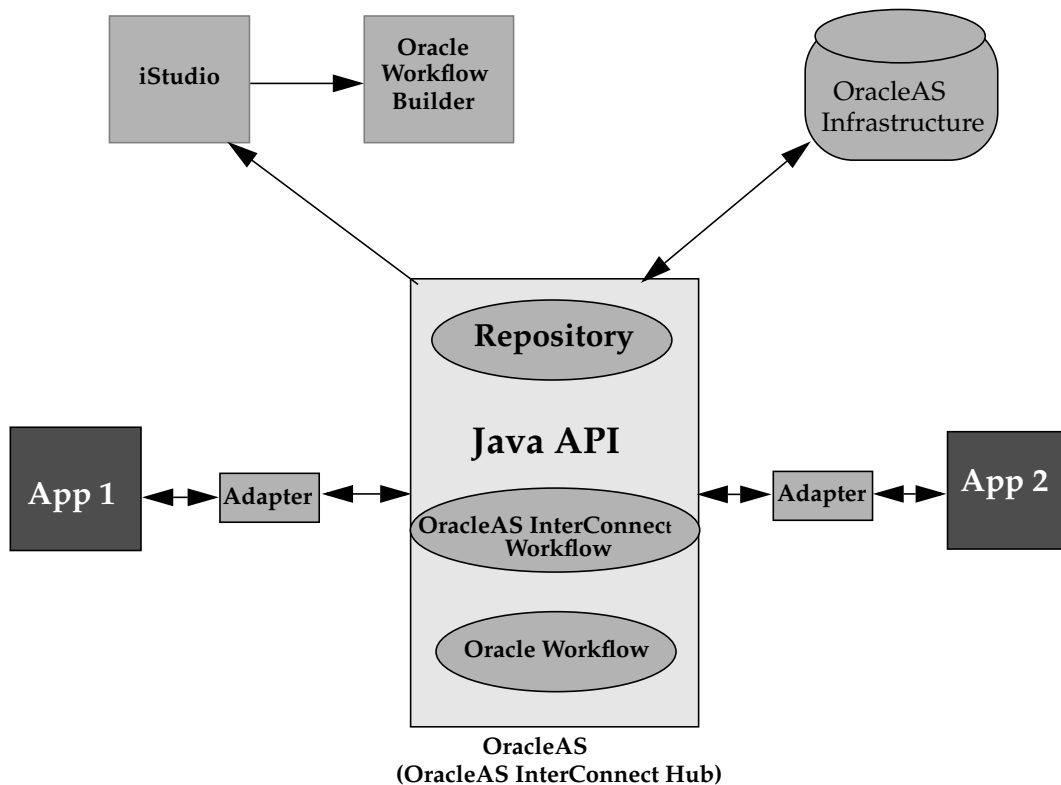
Design Time

During the design time, a business analyst uses iStudio to define the integration objects, applications that participate in the integration, and the specifications of the data exchanged between applications. All the specifications are stored as metadata in the OracleAS InterConnect Repository.

Runtime

For each application participating in a specific integration, OracleAS InterConnect attaches one or more adapters to it. At runtime, the adapters retrieve the metadata from the repository to determine the format of messages, perform transformations between the various data formats, and route the messages to the appropriate queues in the OracleAS InterConnect hub.

Figure 1–1 A graphical overview of design time and runtime phases in integration



Separation of Integration Logic and Platform Functionality

Integration using OracleAS InterConnect is a two-step process. During design time, integration logic is modeled in iStudio and captured in the repository as metadata. Metadata is created in the repository using iStudio during design time and is represented by application views, common views, and transformations. At runtime, the underlying services regard this metadata as runtime instructions to enable the conversation among participating applications. Integration has two components:

- **Integration logic**—Consists of the business rules and transformation logic necessary to integrate heterogeneous systems. Using iStudio, this integration logic can be modeled and the results stored in the repository as metadata.

- Platform functionality—Consists of the integration infrastructure provided with OracleAS InterConnect and the Oracle database. In addition, OracleAS InterConnect provides application and protocol adapters. The platform services provide the requisite infrastructure necessary for integration.

Unique Integration Methodology

iStudio exposes an integration methodology that eliminates the complexities of point-to-point custom integration solutions. The integration methodology is based on a hub-and-spoke model.

How the Hub-and-Spoke Methodology Works

An integration point is defined as an *event* that triggers communication between two or more participating applications in the integration scenario. The following are examples of such *events*:

- *Create Customer*—An integration scenario may require that customer information across two applications be synchronized in real time. Whenever a new customer is created in the application, *App1*, the customer should also be created in the application, *App2*. Therefore, *Create_Customer* is an *event* that triggers the communication between the two applications—*App1* produces the information, *App2* consumes it.
- *Get Item Info*—A user of *App1* may request information on an item stored in *App1*. The information on that item might be segmented across the two applications. To give a meaningful response to the user of *App1*, it is necessary to query *App2* for information on the item. Therefore, *Get_Item_Info* is an integration point between the two applications because it triggers communication between the two applications—*App1* produces a query, *App2* consumes it, *App2* produces the response, and *App1* consumes it.

The common view consists of a list of such integration points, each with its own associated data. Applications participate in the integration by binding to one or more of these common view integration points.

For each binding, applications have their own application view of data that needs to be exchanged. Each binding involves a mapping, or transformation, between the application view and the common view in the context of the integration point. In this model, the application views are at the spokes and the common view is the hub.

Create_Customer is an integration point. If the information to exchange is the new customer's name only, the common view has all the information potentially

captured in a name defined in an application-independent method. This information must be a superset of all the information that needs to be exchanged across App1 and App2.

Prefix, First Name, Last Name, Middle Initial, Maiden Name, Suffix is an example of a common view customer name definition.

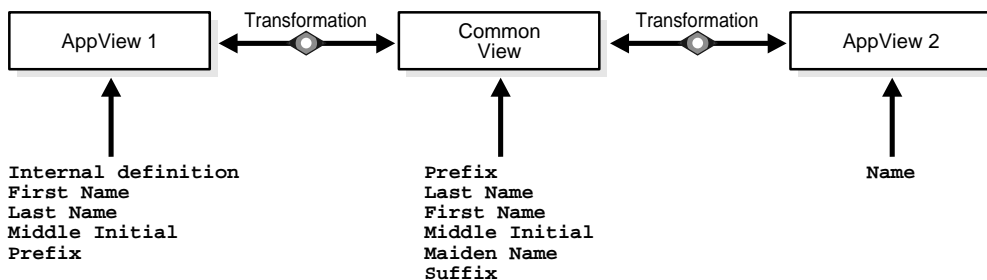
Now, App1's internal *definition* of name (App1's application view) could be First Name, Last Name, Middle Initial, Prefix.

The application view for App2 could be Name (one field that describes Last Name, First Name).

For App1, when sending this information out or publishing an event, transformations are defined from its application view to the common view. For App2, when receiving this information or subscribing to an event, transformations are defined from the common view to its application view.

Figure 1–2 illustrates this example within the hub-and-spoke model where the common view is the hub and the application views are the spokes.

Figure 1–2 OracleAS InterConnect Hub-and-Spoke Model



This hub-and-spoke model has the following advantages:

- Loosely coupled integration—Applications integrate to the common view, not with each other directly. This reduces the number of integration interfaces.
- Easy Customization—Changes in application views because of application upgrades are localized. The changes in the upgraded application should only be reflected through changes in its application view and mappings to the common view. In other words, only the spoke of the changed application should be

re-mapped to the hub. The other spokes and their relationships with the hub remain unchanged.

- **Easy Extensibility**—Applications can be added or removed from the integration scenario without affecting other applications. For example, if a new application is added to the integration scenario, it must define its spoke component (the application view) and map that component to the hub (common view) on a per integration point basis. This does not affect other applications in the integration.
- **Enhanced Reusability**—To integrate the Marketing CRM module to SAP, the integration would be from iMarketing to common view to SAP. If there is a requirement to integrate iMarketing to Peoplesoft, then the iMarketing to common view integration can be reused. Only the common view to the Peoplesoft integration needs to be built.

Integration Lifecycle Management

Managing, customizing, and evolving an integration over time is as important as creating the integration in the first place. The hub-and-spoke integration model has advantages to help achieve this goal. In addition, the OracleAS InterConnect repository, which contains all the integration logic, provides extensive services for managing changes over time. The repository provides fine-grained versioning of all OracleAS InterConnect first class objects such as events, messages, and data types. Some of the important aspects of versioning to aid the lifecycle support include:

- **Basic Versioning**—New versions of first class objects, such as messages, can be created to address changing integration needs. Different versions of the same object can co-exist in the repository. This approach has two advantages:
 - Eliminates the need for an expanded namespace to address modifications.
 - Allows related entities to be grouped together for easier management.
- **Multiple Active Versions**—Multiple versions of the same message can be active in the same integration scenario simultaneously. This helps transition an integration incrementally without requiring changes to existing messages. For example, if a purchase order definition for an application or the application view of the purchase order needs to change, a new version of the message can be created and activated for that application. Once this metadata is created, the application can smoothly transition from sending and receiving messages based on the old definitions to the new one.
- **Migration Support**—Different versions of metadata can be migrated across repositories on a first class object basis. This feature allows fine-grained control

of content in different repositories, such as a development repository and a production repository.

- **Consistency Control**—OracleAS InterConnect detects and flags metadata conflicts. This helps to prevent accidental overwriting of metadata and maintains consistency of metadata in the repository.

Using Adapters for Integration

Adapters are runtime components which process integration logic captured in the repository as runtime instructions to enable the integration. Prepackaged adapters help re-purpose applications at runtime to participate in the integration without any programming effort.

Adapters complete the following tasks:

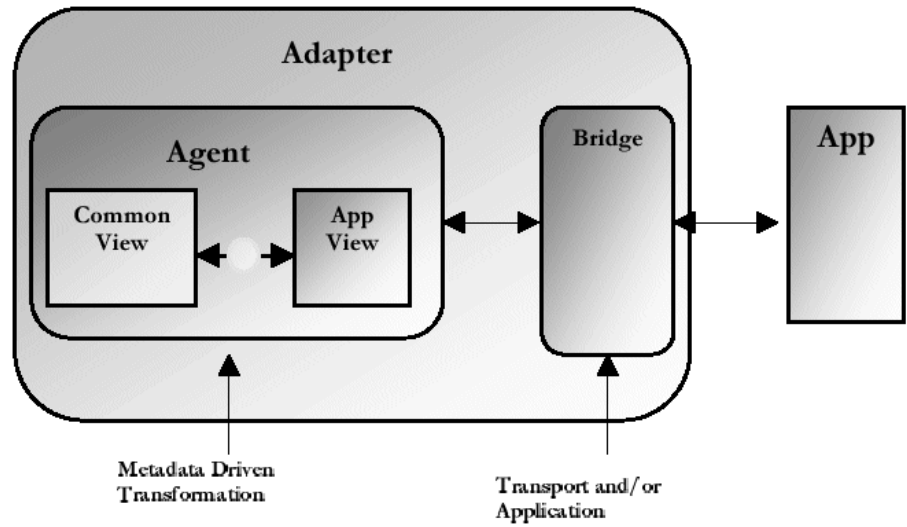
- **Application Connectivity**—Connect to applications to transfer data between the application and OracleAS InterConnect. The logical subcomponent within an adapter that handles this responsibility is called a bridge. This is the protocol/application-specific piece of the adapter that communicates with the application.

For example, the database adapter is capable of connecting to an Oracle database using JDBC and calling SQL APIs. This subcomponent only knows how to call the correct APIs.

- **Transformations**—Transform data from the application view to common view and vice versa as dictated by the repository metadata. In general, adapters are responsible for carrying out all the runtime instructions captured through iStudio as metadata in the repository. Transformations are an important subset of these instructions. The logical subcomponent within an adapter that handles the runtime instructions is called an agent. This is the generic runtime engine in the adapter that is independent of the application to which the adapter connects. It focuses on the integration scenario based on the integration metadata in the repository. There is no integration logic coded into the adapter itself. All integration logic is stored in the repository. The repository contains the metadata that drives this subcomponent.

In the database adapter example, this is the subcomponent that knows which SQL API's to call, but not how to call them. All adapters have the same agent code. It is the difference in metadata that each adapter receives from the repository that controls and differentiates the behavior of each adapter.

Figure 1-3 OracleAS InterConnect Adapter Architecture



See: *OracleAS InterConnect Installation Guide* for a complete list of OracleAS InterConnect Adapters.

Using iStudio

This chapter describes the iStudio and its concepts and discusses the following topics:

- [Overview of iStudio](#)
- [Starting iStudio](#)
- [Parts of the iStudio Window](#)
- [Using Workspaces in iStudio](#)
- [Using Projects in iStudio](#)

Overview of iStudio

iStudio is a design time integration specification tool used to help business analysts specify the integration logic at a functional level, instead of a technical coding level. iStudio exposes the integration methodology using simple wizards and reduces or eliminates the need for writing code to specify the integration logic. This reduces the total time required to complete an integration.

iStudio is a multi-user tool with fine-grained locking for all OracleAS InterConnect first class objects. This allows multiple users to work simultaneously on the same integration scenario without compromising the consistency of the metadata.

iStudio allows business analysts to complete the following tasks:

- Define the data that needs to be exchanged across applications.
- Semantically map the data across applications.
- Define the business process collaboration across applications and associate the semantic maps with business processes if required.
- Configure and deploy the integration.

iStudio is deployed as a stand-alone Java application running outside the database. iStudio runs only on Windows and can be deployed anywhere with access to the hub machine.

See Also: *Oracle Application Server InterConnect Installation Guide*

iStudio Concepts

The following concepts are described briefly:

- [Applications](#)
- [Common Views and Business Objects](#)
- [Transformations or Mappings](#)
- [Metadata Versioning](#)
- [Tracking Fields](#)
- [Content-Based Routing](#)
- [Cross Reference Tables](#)

- [Domain Value Mapping](#)
- [Routing and the Message Capability Matrix](#)

Applications

Each component integrated with OracleAS InterConnect is referred to as an application. Each application expresses interest specific messages, what its internal data type is, and how the message should be mapped to or from that internal type to the external world. iStudio also allows users to create applications.

See Also: [Chapter 3, "Creating Applications, Common Views, and Business Objects"](#)

Common Views and Business Objects

OracleAS InterConnect follows a hub-and-spoke integration methodology. The common view is the *hub view* of the integration where each spoke is the application participating in the integration. The common view consists of the following elements:

- **Business Objects**—A collection of logically related integration points. For example, Create Customer, Update Customer, Delete Customer, and Get Customer Info are all integration points that logically belong under a Customer business object.
- **Events**—An integration point used to model the Publish/Subscribe paradigm. An event has associated data which is the common view of all the data to be exchanged through this event.
- **Procedures**—An integration point used to model the Request/Reply paradigm. This is a modeling paradigm only, no actual procedures are called.
- **Common Data Types**—Used to define such data for reuse and is especially useful for defining complex hierarchical data.

See Also: [Chapter 3, "Creating Applications, Common Views, and Business Objects"](#)

Events An event is an integration point used to model the Publish/Subscribe paradigm. An event has associated data that is the common view of all the data to be exchanged through this event. In other words, the data associated with an event in the common view must be a superset of the data of participating applications.

See Also: [Chapter 4, "Using Events in iStudio"](#)

Procedures A procedure is an integration point used to model the Request/Reply paradigm. This is a modeling paradigm only, no actual procedures are called. An application can either invoke a procedure to model sending a request and receiving a reply, or implement a procedure to model receiving a request and sending a reply. Similar to events, a procedure has associated data. While an event is only associated with one data set, a procedure has two data sets—one for the request or IN data and one for the reply or OUT data.

See Also: [Chapter 5, "Using Procedures in iStudio"](#)

Transformations or Mappings

Transformations are integration points between applications. In the following example, an event is created for transferring customer names across applications:

Application View for App1 that publishes the event:

- First Name
- Last Name
- Middle Initial

Common View Event New Customer:

- Prefix
- First Name
- Last Name
- Middle Initial
- Suffix

Application View for App2 that subscribes to the event:

- Name — One field in the form of LastName, FirstName

When publishing or subscribing to the event, the application view for App1 and App2 must be mapped to the common view using transformations. There are twenty-seven built-in transformation routines provided with OracleAS InterConnect that are used to build complex mappings. In addition, using the iStudio SDK allows new transformation routines to be created using Java. These transformations can be imported into iStudio and then used identically to a built-in routine.

See Also:

- [Chapter 5, "Using Procedures in iStudio"](#) for more information on procedures
- [Appendix C, "Transformations"](#) for more information on transformations

Metadata Versioning

iStudio supports versioning for application and common data types, events, procedures, and messages.

An owner is the creator of the object and only the creator can modify the object. However, other users can create new versions or copy the original object under a new name. The owner is specified when the repository is installed.

In the following examples, metadata is created at Oracle Corporation and at the time of repository installation. OAI is specified as the owner of the metadata. The following functionality is available for versioning:

- **Automatic Versioning**—First, an event called `NewCustomerEvent` is created. When this object is created for the first time, the assigned owner is OAI and the version is V1. This event name is `NewCustomerEvent/OAI/V1`.
- **Modify Object**—The owner is the only user who can modify the contents of an event and the data associated with it. However, the owner cannot change the version number or the name of the event.
- **Create New Version**—If the owner wants to keep the original `NewCustomerEvent` but wants to create a new version of the information with modified data, the owner can create a new version. When this version is saved, there are now two objects—`NewCustomerEvent/OAI/V1` and `NewCustomerEvent/OAI/V2`.
- **Load Version**—Not all versions of objects are loaded into iStudio. To work with a specific version of an object, use the Load Version capability. When a new version is created, it becomes the current version.
- **Copy Object**—To create a `NewBigCustomerEvent` that has many common elements with `NewCustomerEvent/OAI/V1`, first load `NewCustomerEvent/OAI/V1` and copy the object in iStudio. Copying the object allows not only modifications to the data, but also modifications to the name of the event. When the name of the `NewBigCustomerEvent/OAI/V1` event has been modified, `NewCustomerEvent/OAI/V1` will coexist in the repository.

Note: Names of events must be unique.

In this example, all the metadata is built at Oracle Corporation and this metadata can be transmitted to the customer, *NewCorp*. When *NewCorp* installs the repository and specifies the owner as *NewCorp*, the metadata is in a read-only state. If *NewCorp* wants to customize *NewBigCustomerEvent/OAI/V1*, they cannot modify the existing version since the owners are different. However, they can use the other features described.

To customize the metadata, *NewCorp* must create a new version so that *NewBigCustomerEvent/OAI/V1* and *NewBigCustomerEvent/NewCorp/V2* coexist in the repository. *NewCorp* can use both events in defining messages if required and *NewCorp* can now modify the event it owns.

See Also: [Chapter 4, "Using Events in iStudio"](#)

Tracking Fields

Tracking fields are one or more application view fields in the context of a particular message. If specified in iStudio, tracking fields can be used to track messages at runtime using the Oracle Enterprise Manager. Tracking is executed only from the perspective of the sending application.

For example, if *App1* publishes a new purchase order and specifies the *PO_order* number field as the tracking field, then the user can log in to the runtime console and specify the message to track, or *New Purchase Order* in this case. The user is then prompted to enter the purchase order number to display the corresponding tracking information.

Content-Based Routing

Using the wizards in iStudio, messages can be routed to specific applications based on business rules or message content.

See Also: [Chapter 6, "Enabling Infrastructure"](#)

Cross Reference Tables

Keys for corresponding entities created in different applications can be correlated through cross referencing in iStudio.

See Also: [Chapter 6, "Enabling Infrastructure"](#)

Domain Value Mapping

Code tables can be mapped across systems using domain value mapping in iStudio.

See Also: [Chapter 6, "Enabling Infrastructure"](#)

Routing and the Message Capability Matrix

In the OracleAS InterConnect hub, Advanced Queues in the database are used to store, route, and forward messages from the sending application adapters to the receiving application adapters. The following paradigm is used for routing messages. The sending adapters evaluate who the recipients are based on metadata.

1. Every adapter has one or more queues where it receives messages.
2. The Message Capability Matrix allow queues to be specified for receiving messages on a per message per receiving application basis.

Note: By default, there is only one queue named the `oai_hub_queue`. This queue is used for all messages for all applications. This queue does not need to be changed unless the single queue implementation turns out to be a performance bottleneck.

Starting iStudio

To log into iStudio, the database and the repository must be running. To log in to iStudio:

- From the Windows Start menu, select OracleAS InterConnect, then select iStudio.

When iStudio starts, the last opened project is automatically loaded into the default workspace.

See Also: ["Creating a New Project"](#) on page 2-15

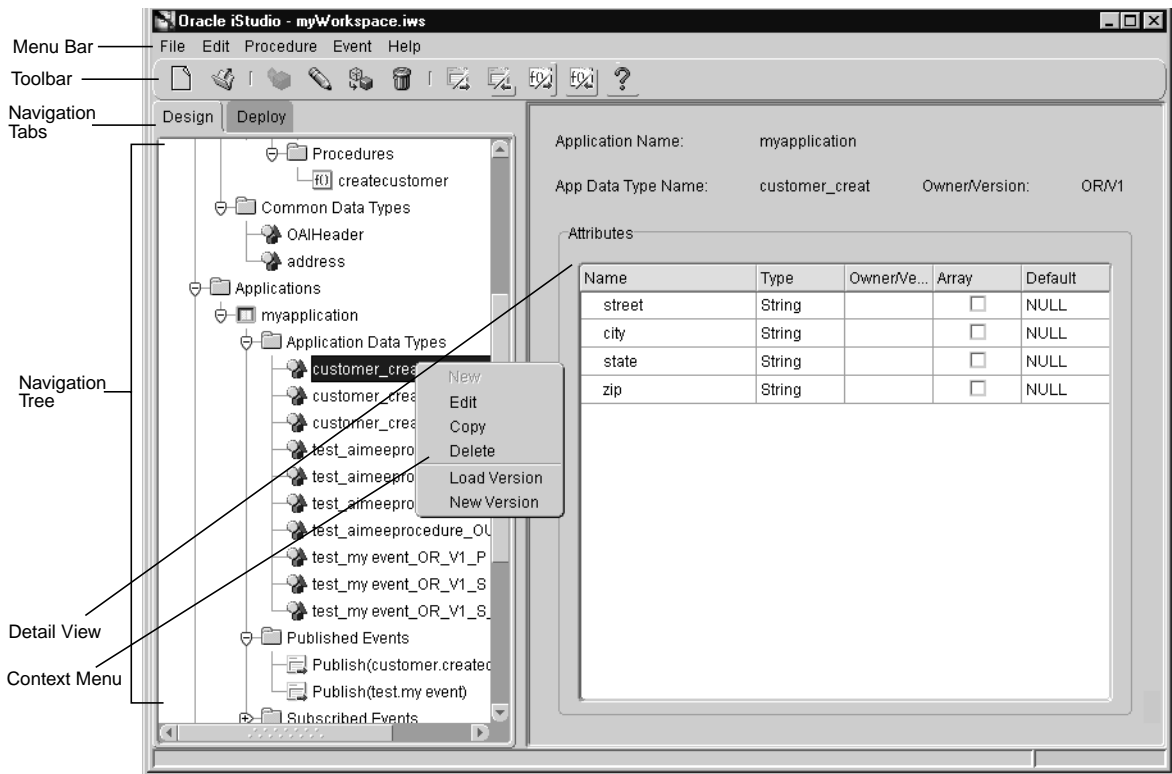
Parts of the iStudio Window

The main iStudio window has the following parts:

- [Menu Bar](#)
- [Toolbar](#)
- [Design Navigation Tree](#)
- [Deploy Navigation Tree](#)
- [Context Menus](#)
- [Detail View](#)

When iStudio is started, the main window displays:

Figure 2–1 OracleAS InterConnect iStudio



Menu Bar

The menu bar provides access to all commands. Click each menu to display its commands. Click a command to execute it. There are five menus:

- File Menu
- Edit Menu
- Procedure Menu
- Event Menu
- Help Menu

File Menu

Use the File menu to create new projects and workspaces, open existing projects and workspaces, or reload existing projects. You can also create such objects as events, procedures, and common data types from the File menu. Commands include:

- New Project...—Creates a new project.
- Open Project...—Opens an existing project. In the Open dialog, select the directory and project, then click Open.
- New Workspace...—Creates a new workspace.
- Open Workspace...—Opens an existing workspace. In the Open dialog, select the directory and workspace, then click Open.
- Reload Project—Reloads a project. When Reload Project is selected, a list of current projects displays. Select the project to Reload from the list.
- Migrate—Migrates objects from one repository to another.
- New—Creates a new object in iStudio. When New is selected, a list of available objects displays. Select the object to create. If some objects are grayed-out, then they are not allowed to be created.
- Export PL/SQL—Exports SQL-stored procedure stubs generated by iStudio.
- Push Metadata—Pushes metadata to adapters.
- Exit—Leaves iStudio.

Edit Menu

Use the Edit menu to edit, copy, or delete selected objects. If an object is selected and the Edit menu is not available, that selected object cannot be edited. Commands include:

- Edit—Edits a selected object. The type of editing depends on the object selected.
- Copy—Copies a selected object.
- Delete—Deletes a selected object.
- Rename—Renames a selected application.
- Version—Creates a new version of or load a selected object.
- Domain Value Map—Adds or removes applications from a domain value map.
- Cross Reference Table—Adds or removes applications from a cross reference table.
- Workflow—Deploys events to Oracle Workflow or edits Oracle Workflow configuration information.

Procedure Menu

Use the Procedure menu to invoke or implement procedures. Commands include:

- Invoke—Invokes a selected procedure by launching the Invoke Wizard.
- Implement—Implements a selected procedure by launching the Implement Wizard.

Event Menu

Use the Event menu to publish or subscribe events. Commands include:

- Publish—Publishes a selected event by launching the Publish Wizard. An event must be created.
- Subscribe—Subscribes to a selected event by launching the Subscribe Wizard. An event must be created.









Help Menu




The Help menu provides links to online help. Commands include:

- Contents—Opens the User's Guide.
- About...—Displays version information for iStudio.

Toolbar

The toolbar is made up of icons that represent frequently used commands. To display a caption describing the icon, pause the cursor on the icon. The following functions are provided:

Function	Icon	Description
New Project		Creates a new project in iStudio.
Open Project		Opens an existing project in iStudio.
Create Integration Object		Creates a new integration object.
Create Like		Creates a new integration object similar to an existing object. This icon is enabled only when an object is selected in the Navigator.
Edit Integration Object		Edits a selected integration object.
Delete Integration Object		Deletes a selected integration object. This button is enabled only when an integration object is selected in the Navigator.
Publish Event		Publishes a selected event. This button is enabled only when an integration object is selected in the Navigator.
Subscribe Event		Subscribes to a selected event. This button is enabled only when an integration object is selected in the Navigator.

Function	Icon	Description
Invoke Procedure		Invokes a selected procedure. This button is enabled only when an integration object is selected in the Navigator.
Implement Procedure		Implements a selected procedure. This button is enabled only when an integration object is selected in the Navigator.
Help		Displays the help file.

Design Navigation Tree

The Design Navigation tree displays the hierarchical tree of all objects used in the design phase of an opened project. Each object type in the Deploy Navigation tree is identified by an icon and name. A container is represented by a folder icon and is a logical grouping of one specific type of object, such as Business Objects and Application Data Types.

The objects are grouped as follows:

- Common Views
- Applications
- Workflow
- Enabling Infrastructure

Deploy Navigation Tree

The Deploy Navigation tree displays the hierarchical tree of all objects used in the deploy phase of an opened project. Each object type in the Design Navigation tree is identified by an icon and name. A container is represented by a folder icon and is a logical grouping of one specific type of object, such as Process Bundles.

The objects are grouped as follows:

- Applications
- Workflow

Context Menus

As in other Windows applications, you can right-click an object to pop up a context menu; that is, a shortcut menu relating to the object right-clicked.

Navigation Tree	Selected Item	Context Menu Options
Design	Object, such as Common View, Application, Business Objects, and Common Data Types	New, Edit, Copy, Delete
	Container object, such as an existing event or procedure	New, Edit, Copy Delete, Load Version, New Version
	Workflow object	New, Edit, Copy, Delete, Launch WF Builder, Launch WF Home Page
Deploy	Object such as Applications	New, Edit, Copy, Delete
	Workflow object	New, Edit, Copy, Delete, Deploy, Edit Configuration, Launch WF Home Page
	Container object, such as an existing routing object	New, Edit, Copy, Delete, Create Partition

Detail View

To the right of the Navigation tree is the Detail View, composed of one or more property sheets displaying information about the object selected. Often, these property sheets may be edited.

Using Workspaces in iStudio

A workspace stores user settings and preferences, such as application login credentials and last opened project. Inside a workspace, users can work on multiple projects.

Creating a New Workspace

To create a new workspace:

1. From the File menu, select New Workspace. The New Workspace Dialog displays.

2. Enter a name for the workspace in the Workspace Name field.
3. Click OK.

Opening an Existing Workspace

To open an existing workspace:

1. From the File menu select Open Workspace. The Open dialog displays.
2. Enter the name and path to an existing workspace or select the workspace to open.



3. Click Open. The selected workspace displays in iStudio.

Using Projects in iStudio

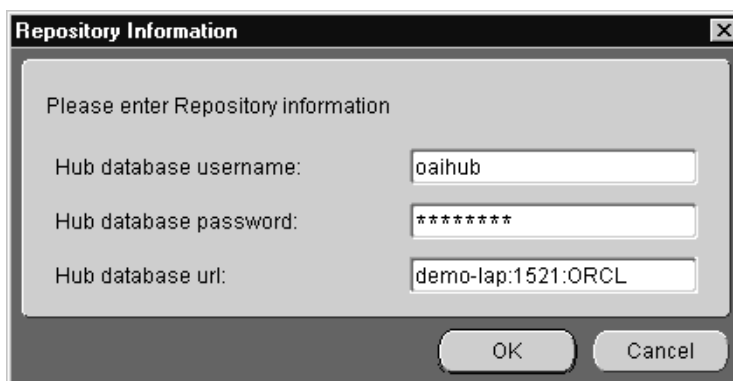
A project in iStudio encapsulates all the integration logic for one integration scenario. An integration scenario is defined as a set of two or more applications integrated with each other using OracleAS InterConnect. One project corresponds to one repository. For example, a user may have a development integration environment and a production integration environment. These are two separate projects and must, therefore, be self-contained in their own separate repositories.

Since iStudio is a multi-user tool, multiple users can work on the same project, simultaneously, without jeopardizing the integrity of the metadata. To create a project in iStudio, the repository must be running.

Creating a New Project

To create a new project in iStudio:

1. From the File menu, select New Project. The New Project Dialog displays.
2. Enter the project name and click OK. The Repository Information dialog displays:



The screenshot shows a dialog box titled "Repository Information" with a close button in the top right corner. The dialog contains the text "Please enter Repository information" and three input fields. The first field is labeled "Hub database username:" and contains the text "oaihub". The second field is labeled "Hub database password:" and contains "*****". The third field is labeled "Hub database url:" and contains "demo-lap:1521:ORCL". At the bottom of the dialog are two buttons: "OK" and "Cancel".

3. Enter information in the following fields:
 - Repository Name—The name of the repository server.
 - Hub database username—The name of the hub database user. The default username is oaihub.
 - Hub database password—The password associated with the hub database user. The default password is set when OracleAS InterConnect is installed.
 - Hub database url—Information of the following form:
`machine name:port number:database sid.`
4. Click OK.

Opening an Existing Project

To open an existing project:

1. From the File menu, select Open Project. The Open dialog displays:



2. Enter the name and path to an existing project or select the workspace to open.
3. Click Open. The selected project displays in iStudio.

Creating Applications, Common Views, and Business Objects

This chapter describes how to create and manage applications, common views, and business objects using iStudio.

This chapter discusses the following topics:

- [Overview of Applications](#)
- [Overview of Common Views and Business Objects](#)

Overview of Applications

Each component integrated with OracleAS InterConnect is referred to as an application. Each application expresses interest in specific messages, what its internal data type is, and how the message should be mapped to or from that internal type to the external world.

Application View

Each application has its own application view of data that allows it to participate in the integration. The application view of data uses transformations to insert into the common view.

After creating an application in iStudio, events and procedures can be created and used with the application.

See Also:

- [Chapter 4, "Using Events in iStudio"](#) for information on events
- [Chapter 5, "Using Procedures in iStudio"](#) for more information on procedures

Application Data Types

Application data types have the same function as Common Data Types but relate to a particular application.

Creating an Application

To create an application:

1. From the File menu, select New, then select Application. The Create Application dialog displays.
2. Enter a name for the application in the Application Name field.
3. Click OK.

The application created displays in the Design Navigation Tree under the Applications node.

Overview of Common Views and Business Objects

The common view is the *hub view* of the integration where each spoke is an application wanting to participate in the integration. After defining a common view by creating a business object and common data types, existing events can be published or subscribed to, and procedures can be invoked or implemented.

See Also: [Chapter 2, "Using iStudio"](#)

Defining Common Views

When defining a common view, business objects and common data types must be created.

Creating Business Objects

To create a new business object:

1. From the File menu select New, then select Business Object. The Create Business Object dialog displays.
2. Enter a name for the business object in the Business Object Name field.
3. Click OK. The business object displays in the Design Navigation Tree under the Common View node.

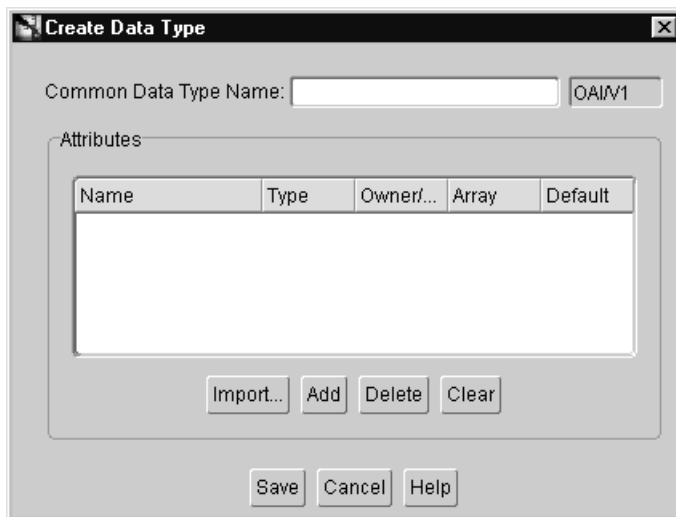
Creating Common Data Types

When creating the data associated with an event or a procedure, it is possible to define the data once and reuse it for different integration points. Common data types are used to define such data for reuse and is especially useful for defining complex hierarchical data.

For example, a purchase order contains a header object and an array of line item objects. In addition, the header object contains two address objects: `Bill_To` and `Ship_To`. Therefore, the purchase order can be defined once and used for other purchase order-related integration points such as `Create_Purchase_Order`, `Update_Purchase_Order`, and `Get_Purchase_Order`. Moreover, `Address` can be defined once and used in the `Bill_To` and `Ship_To` addresses.

To create a common data type:

1. From the File menu select New, then select Common Data Type. The Create Data Type window displays:



Enter a name for the common data type in the Common Data Type Name field.

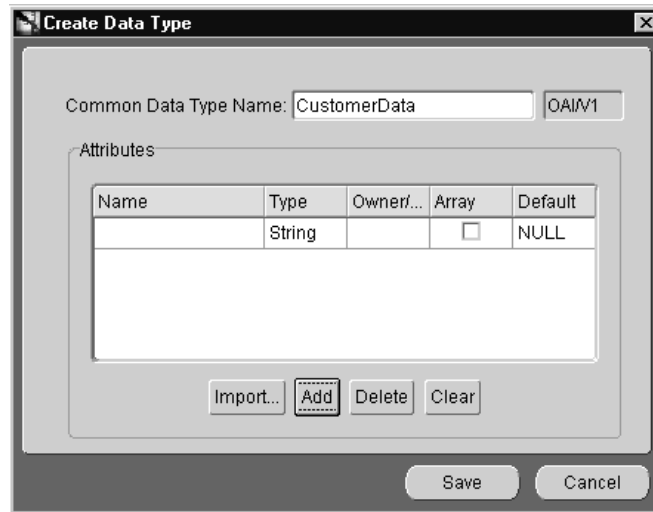
The owner and version number of the common data type display next to the common data type name. This field cannot be edited.

2. Specify the attributes for this common data type using one of the following methods:
 - Add attributes one by one.
 - Import attributes from already existing application native data types or APIs.

Adding Attributes

To add attributes:

1. On the Create Data Type dialog, click Add. A new entry displays in the attribute list:



Specify the following by editing the information directly in the attribute list entry.

Name—The name of the attribute.

Type—The type of the attribute. Select the type by clicking on the Type column in the attribute entry. A dropdown list displays. The attribute can be of primitive type (string, integer, float, double, date) or another common data type used to build hierarchical data types.

Array—Check this box if it the attribute is a collection instead of a single attribute.

Default—The default value of the field in case it is not populated at runtime.

2. Click Save. Repeat the above steps for adding other attributes.

Importing Attributes

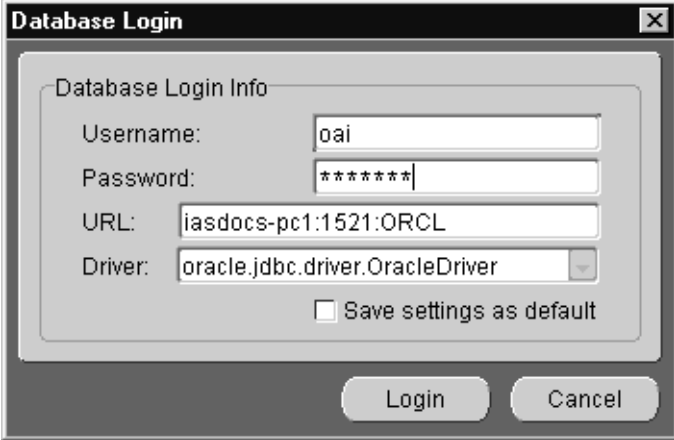
To import attributes:

1. On the Create Data Type dialog, click Import. Attributes can be imported from the following sources:
 - Common Data Types
 - Database
 - XML
 - D3L

The following example utilizes the Database import facility.

See Also: [Appendix B, "Using the Data Definition Description Language"](#)

2. Click Database. The Database Login dialog displays:



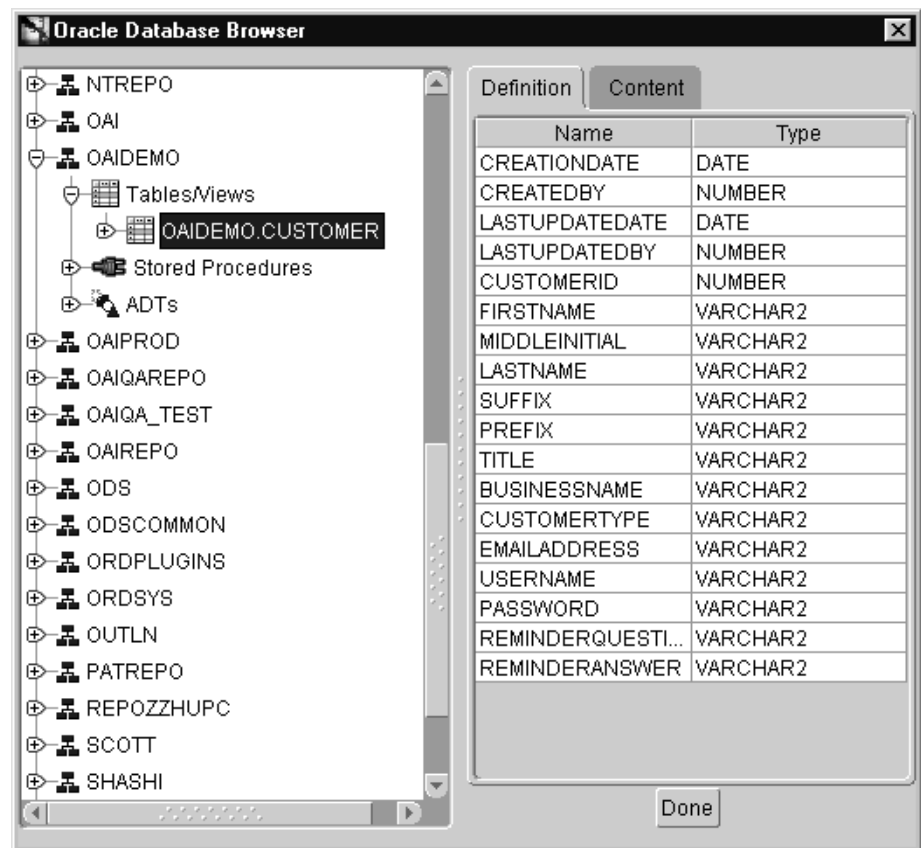
3. Enter information in the following fields:
 - User Name**—The database log in name.
 - Password**—The database log in password.
 - URL**—The machine name: port number: database SID.
 - Driver**—The JDBC driver used to connect to the database.

Save settings as default—Check this box to save the settings for the workspace.

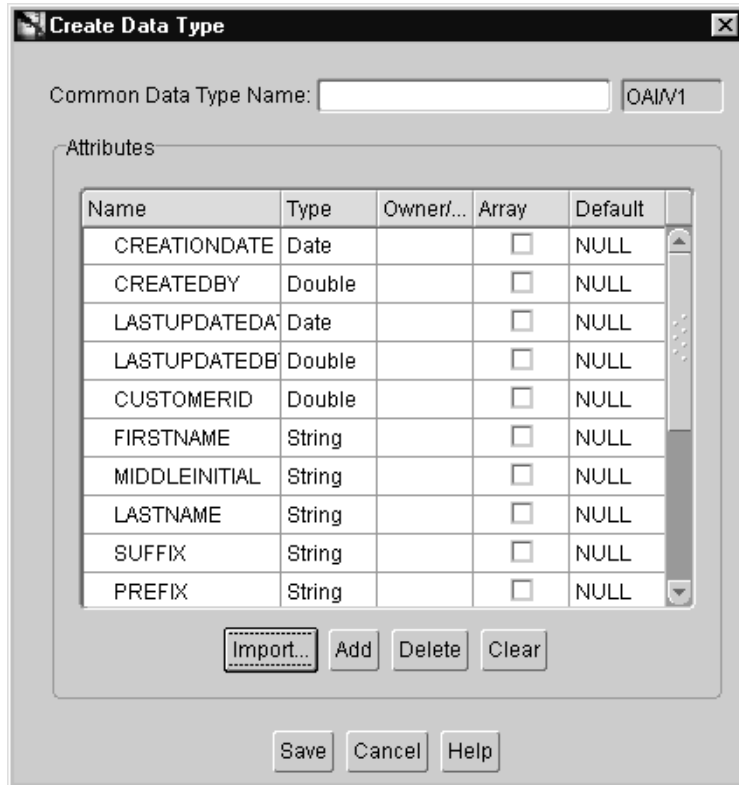
4. Click Login.

After logging in, the database tables and arguments display in the Database Browser Window.

Select the fields to add. To select a range of fields, press Shift when clicking the mouse button. To select multiple items, press Control while clicking the mouse button.



5. Click Done to import the attributes into the common data type. The selected attributes display on the Create Data Type dialog:



Deleting and Clearing Attributes

To delete a selected attribute:

- On the Create Data Type dialog, select the attribute to be deleted and click Delete.

To clear all attributes:

- On the Create Data Type dialog, click Clear.

Using Events in iStudio

This chapter describes how to use iStudio to create, publish, and subscribe to events. This chapter discusses the following topics:

- [Overview of Events](#)
- [Creating Events](#)
- [Publishing and Subscribing to an Event](#)

Overview of Events

An event is an integration point used to model the Publish/Subscribe paradigm. An event has associated data that is the common view of all the data to be exchanged through this event. In other words, the data associated with an event in the common view must be a superset of the data of participating applications.

For example, App1 and App2 publish customer names and App3 subscribes to it. If App1 publishes First Name, Last Name, and Middle Initial, and App2 publishes First Name, Last Name, Prefix, and Suffix, the event could be defined as follows:

```
New Customer Event
Prefix
First Name
Last Name
Middle Initial
Suffix
```

Note: Standard application-independent definitions can be used for event-associated data in the common view such as Open Applications Group XML business object definitions.

See Also: ["How the Hub-and-Spoke Methodology Works"](#) on page 1-9

Event Maps

Event maps allow application data to be mapped to an OracleAS InterConnect event without the application having to know about the OracleAS InterConnect event itself. For example, if an application is publishing a `Create_Customer` event, it does not have to explicitly state that the message it is publishing corresponds to an OracleAS InterConnect `Create_Customer` event. Instead, using iStudio, certain fields in the application view can be associated to help OracleAS InterConnect determine which event the message maps.

In addition, if an application publishes exactly the same structure of data for two or more events, event maps help OracleAS InterConnect distinguish which message corresponds to which event. For example, an application publishes the same Customer Application Data Type regardless of whether it is a `Create_Customer` or an `Update_Customer` event. Through event map, OracleAS InterConnect can

determine which messages correspond to `Create_Customer` and `Update_Customer`.

Creating Events

To create an event:

1. From the File menu, click New, then select Event. The Create Event dialog displays:



Enter the information in the following fields:

Business Object—The name of the category to which the event belongs. Select a category from the drop down list.

Event Name—The name of the event. Only alphanumeric characters can be used.

OAI/V1—The owner and version number of the Business Object. This field cannot be edited.

2. Add or import attributes to this event.

See Also:

- ["Adding Attributes"](#) on page 3-5
- [Appendix B, "Using the Data Definition Description Language"](#)
- ["Deleting and Clearing Attributes"](#) on page 3-8

3. Click Save.

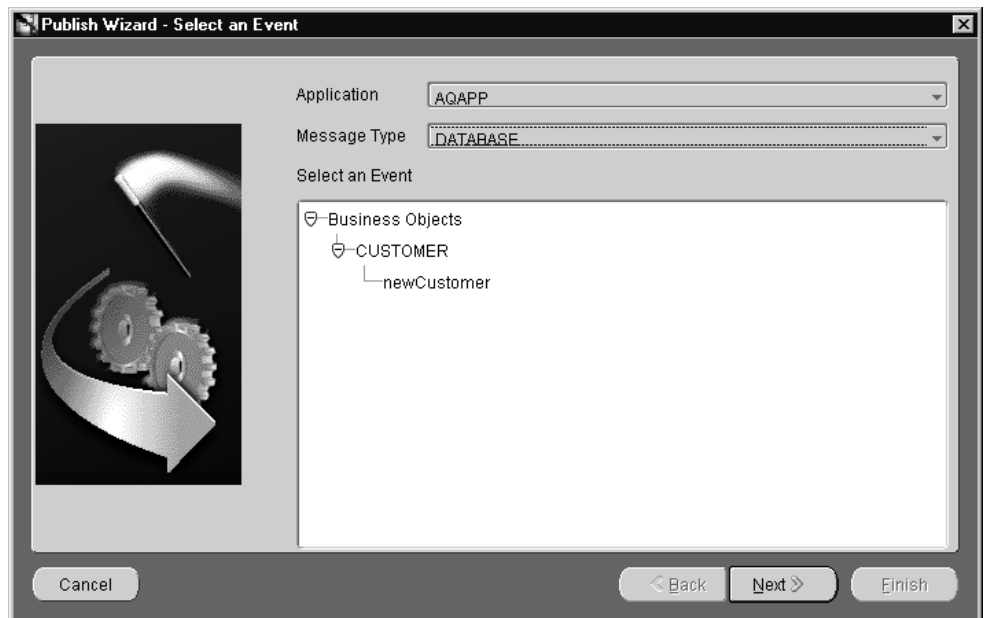
Publishing and Subscribing to an Event

The publish/subscribe paradigm is an existing technology for exchanging information between a provider (publisher) and a set of users (subscriber). This paradigm is one of the most common ways for adapters to communicate with each other through the hub.

Publishing an Event

Publishing an event in an application in iStudio involves using the Publish Wizard. To start the Publish Wizard:

1. In the Design Navigation tree, expand the Application node. Select and expand the Application node to display the Published Events leaf. Right-click Published Events and select Publish. The Publish Wizard displays:



- a. Enter information in the following fields:

Application—The name of the application that is publishing the event.

Message Type—This field specifies the mode of communication between OracleAS InterConnect and the application. Select from the following message types:

Database—OracleAS InterConnect communicates with the application using the database.

Generic—OracleAS InterConnect communicates with application using a user-defined bridge.

XML—OracleAS InterConnect communicates with the application using XML data described through a data type definition (DTD) using the FTP, SMTP, HTTP, MQ Series, or user-defined adapters.

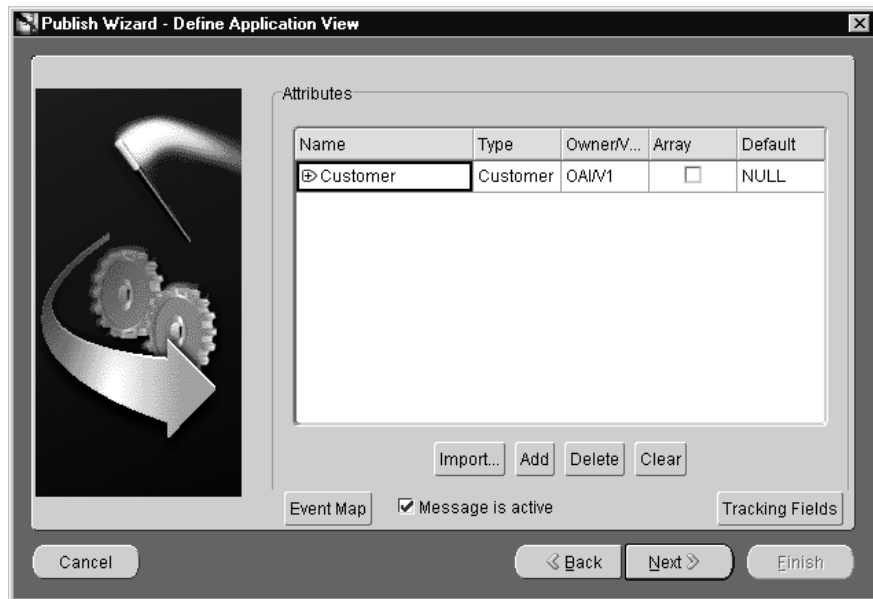
AQ—OracleAS InterConnect communicates with the application through Oracle Advanced Queues using the Advanced Queue adapter. The payload can be Oracle Objects where fields may be XML or RAW XML.

D3L—The adapter communicates with the application using D3L.

- b. Select the event name.
- c. Click Next.

2. Define Application View Page

After clicking Next on the Select Event page, the Define Application View page displays:

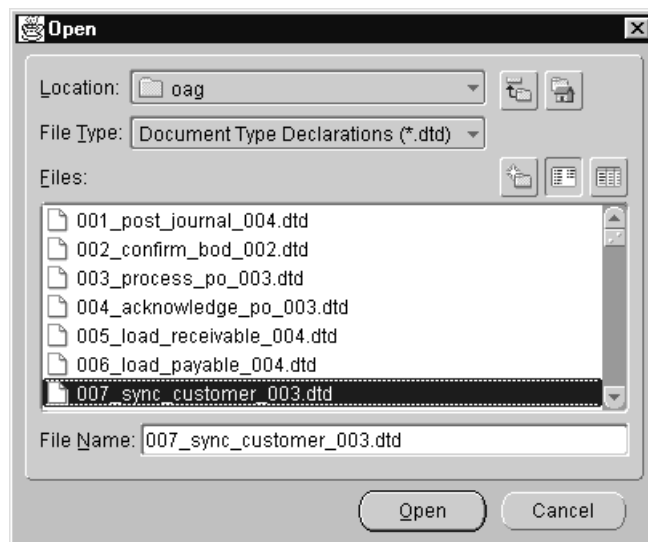


Once an event is selected to publish, the application view is defined. The application view page is initially an empty table. Define the attributes using Add, or import the definitions from a database or an API Repository using Import.

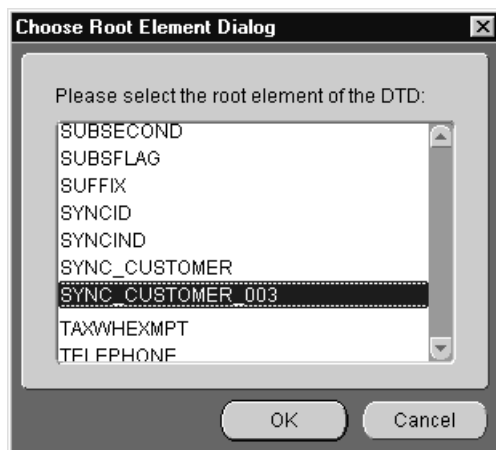
- a. Add or import attributes by clicking Add or Import.

See Also:

- ["Adding Attributes"](#) on page 3-5
 - ["Importing Attributes"](#) on page 3-6
 - ["Deleting and Clearing Attributes"](#) on page 3-8
 - [Appendix B, "Using the Data Definition Description Language"](#)
- b. To import an XML DTD, a file dialog is displayed when the Import button is pressed and XML is selected:

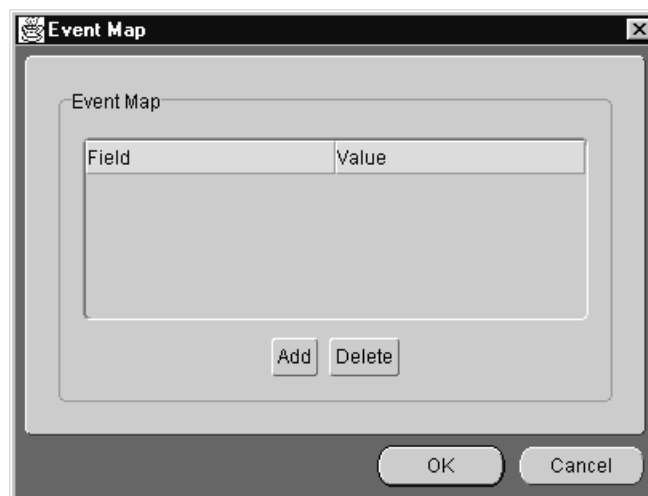


- c. Select a DTD file and click Open. The Choose Root Element dialog displays:



- d. Select a root DTD element and click OK.
- e. If this is a XML type message, the Event Map button is enabled. To define the event map, click Event Map.

The Event Map dialog displays:



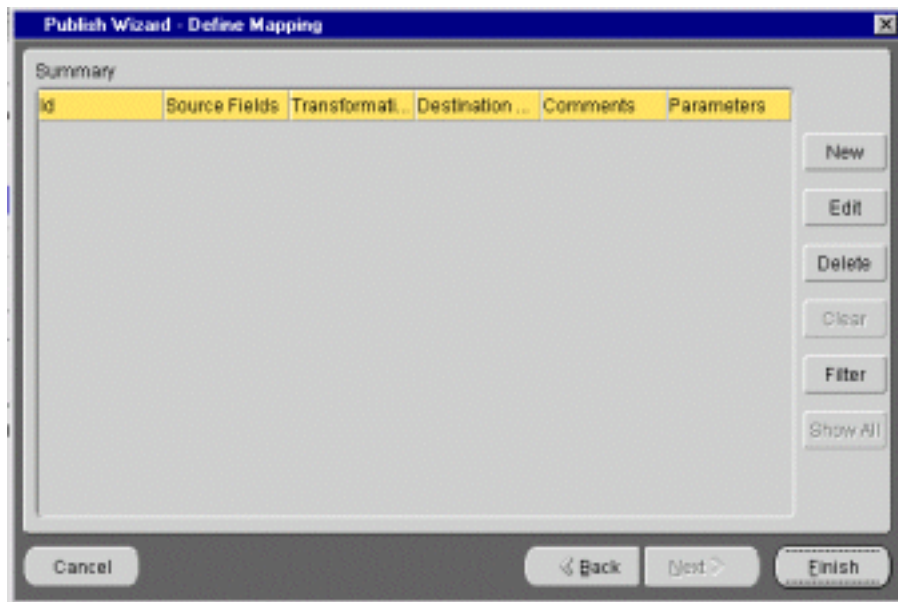
- f. Click Add to add an event map attribute. The New Event Map dialog displays:



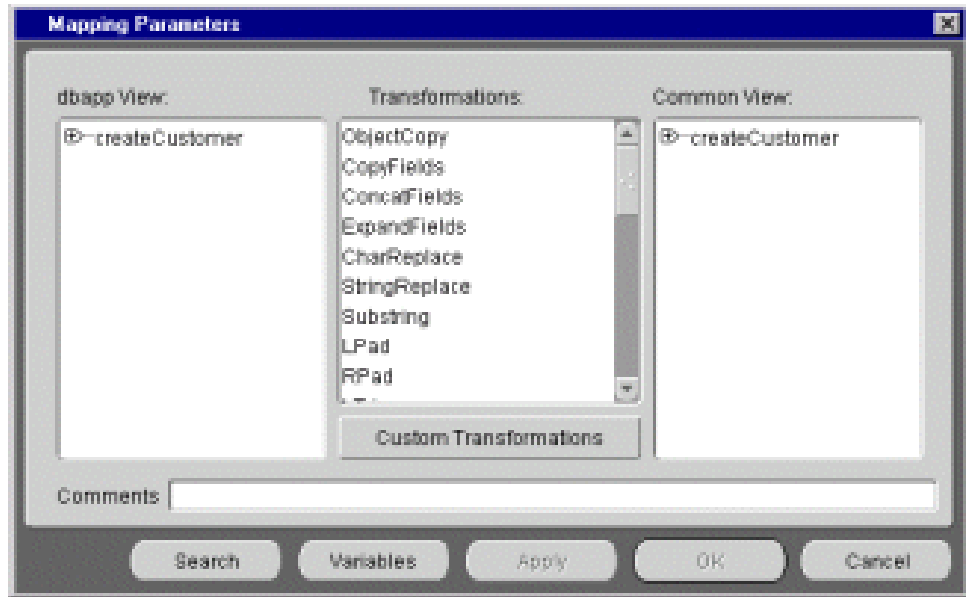
- g. Expand the tree and select an attribute and enter a value in the Value field.
- h. Click OK on the Add Event Map dialog to return to the Event Map dialog.
- i. Click OK to return to the Publish Event Wizard.
- j. Click Next.

3. Define Mapping Page

Mapping involves copying the individual fields or simple shape-change transformations. After clicking Next on the Define Application View page, the Define Mapping page displays:



Click New to define new mappings. The Mapping Parameters dialog displays:



To map fields in the application view to fields in the common view, use a transform. For example, to map fields in the `FirstName` and `LastName` in the common view to `Name` in the application view, use the `ExpandFields` transform.

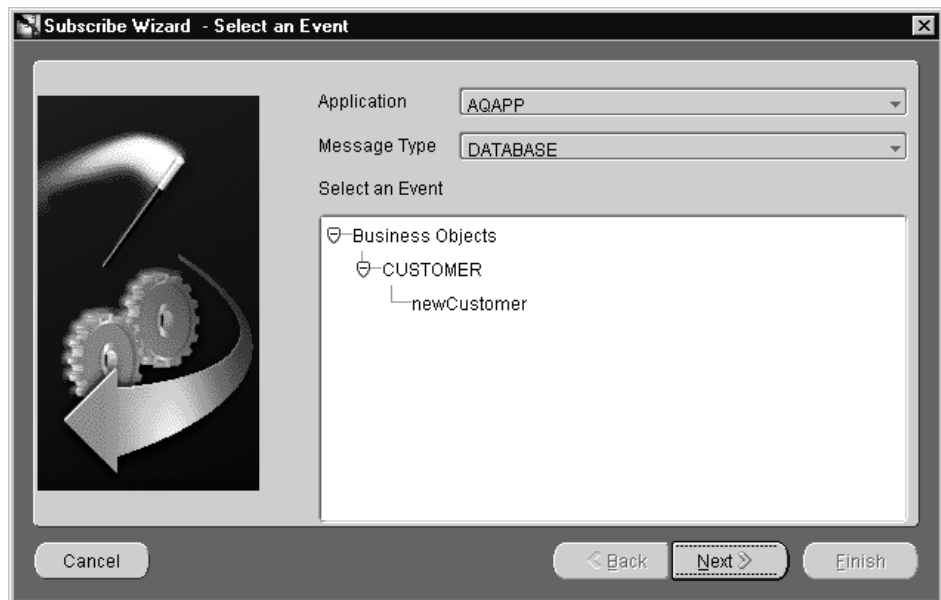
See Also:

- ["Invoking a Procedure"](#) on page 5-5 for detailed instructions using the Mapping dialog
 - ["Adding Custom Transformations"](#) on page 6-11 for more information on creating custom transformations
4. Click OK to return to the Publish Event Wizard.
 5. Click Finish.

Subscribing to an Event

Subscribing to an event in an application in iStudio involves using the Subscribe Wizard. To subscribe to an event in an application:

1. In the Design Navigation tree, expand the Application node. Select and expand the Application node to display the Subscribed Events leaf. Right-click Subscribed Events and select Subscribe. The Subscribe Wizard displays:



- a. Use this page to enter information in the following fields:

Application—The name of the application subscribing to the event.

Message Type—This field specifies the mode of communication between OracleAS InterConnect and the application. Select from the following message types:

Database—OracleAS InterConnect communicates with the application using the database.

Generic—OracleAS InterConnect communicates with application using a user-defined bridge.

XML—OracleAS InterConnect communicates with the application using XML data described through a DTD using the FTP, SMTP, HTTP, MQ Series, or user-defined adapters.

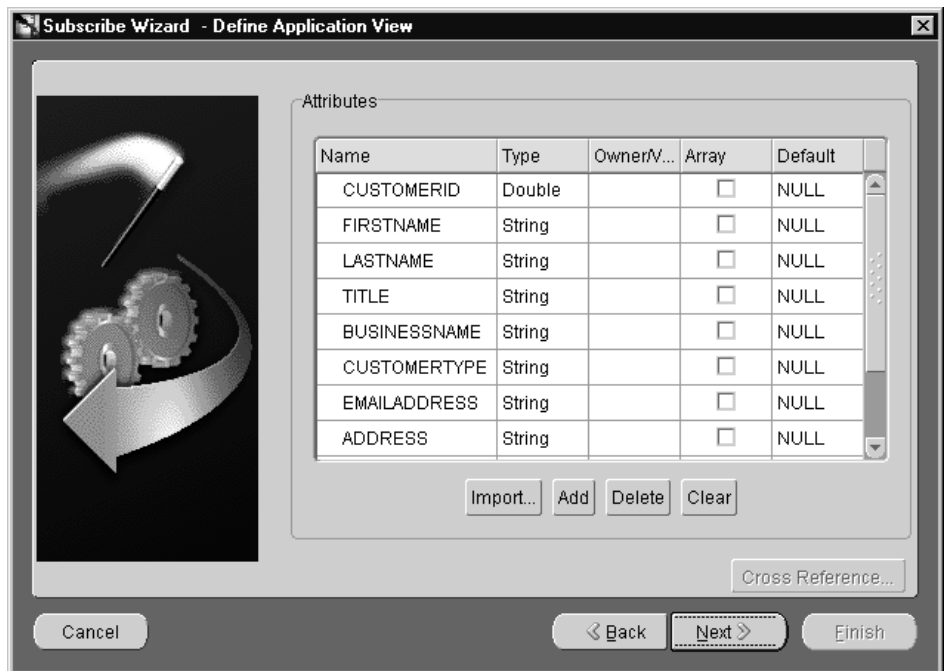
AQ—OracleAS InterConnect communicates with the application through Oracle Advanced Queues using the Advanced Queue adapter. The payload can be Oracle Objects where fields may be XML or RAW XML.

D3L—The adapter communicates with the application using D3L.

- b. Select the event to subscribe to and click Next.

2. Define Application View Page

After selecting the event to subscribe to, the Define Application View page displays:



Once an event is selected to subscribe to, the application view is defined. The application view page is initially an empty table. Define the attributes using

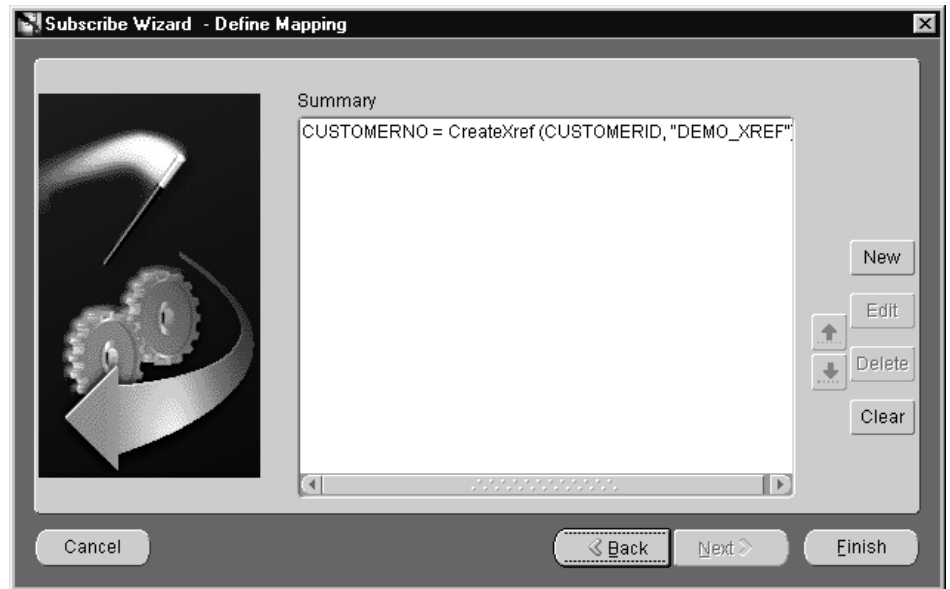
Add or import the definitions from a database or an API Repository using Import.

- a. Add or import attributes by clicking Add or Import.

See Also:

- ["Adding Attributes"](#) on page 3-5
 - ["Importing Attributes"](#) on page 3-6
 - ["Deleting and Clearing Attributes"](#) on page 3-8
 - [Appendix B, "Using the Data Definition Description Language"](#)
- b. To populate and look up cross reference tables, click Cross Reference... The Cross Reference dialog displays. When finished, click OK to return to the Subscribe Wizard.
 - c. Click Next.
- 3. Define Mapping Page**

Mapping can either involve copying the individual fields or simple shape change transformations. After clicking Next on the Define Application View page, the Define Mappings page displays:



- a. Click New to define mappings.

To map fields in the application view to fields in the common view, use a transform. For example, to map fields in the `FirstName` and `LastName` in the application view to `Name` in the common view, use the `ExpandFields` transform.

See Also: ["Invoking a Procedure"](#) on page 5-5

- b. Click Finish.

Using Procedures in iStudio

This chapter describes using iStudio to create, invoke, and implement procedures. This chapter discusses the following topics:

- [Using Procedures](#)
- [Invoking and Implementing a Procedure](#)
- [Exporting Stored Procedures](#)

Using Procedures

A procedure is an integration point used to model the Request/Reply paradigm. This is a modeling paradigm only, no actual procedures are called. An application can either invoke a procedure to model sending a request and receiving a reply, or implement a procedure to model receiving a request and sending a reply. Similar to events, a procedure has associated data. While an event is only associated with one data set, a procedure has two data sets—one for the request or IN data and one for the reply or OUT data.

For example, if a `Get_Address` procedure is defined so that the request contains the social security number, `SSN`, for a person and the reply contains the address in four fields—`Street`, `City`, `Zip`, `State`, then the procedure is defined as follows:

```
get Address Procedure
  SSN IN
  Street OUT
  City OUT
  Zip OUT
  State OUT
```

Procedures can be used to implement both synchronous and asynchronous Request/Reply.

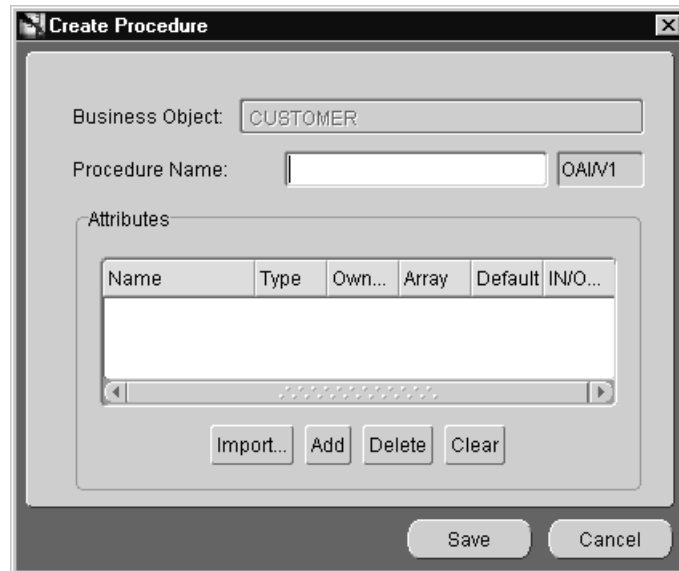
Note: Standard application-independent definitions can be used for procedure-associated data in the common view such as Open Applications Group XML business object definitions.

See Also: ["How the Hub-and-Spoke Methodology Works"](#) on page 1-9

Creating a Procedure

To create a procedure:

1. From the File menu, select New, then select Procedure. The Create Procedure dialog displays:



Enter information in the following fields:

Business Object Name—The name of the category to which the procedure belongs. Select from the drop down list.

Procedure Name—The name of the procedure. Only alphanumeric characters can be used.

OAI/V1—The owner and version number of the procedure. This field cannot be edited.

2. Add or import attributes to this procedure.

See Also:

- ["Adding Attributes"](#) on page 3-5
- ["Importing Attributes"](#) on page 3-6
- ["Deleting and Clearing Attributes"](#) on page 3-8
- [Appendix B, "Using the Data Definition Description Language"](#)

3. Click Save.

Invoking and Implementing a Procedure

An adapter can activate a procedure which is developed by a user or provided as part of an adapter package, as part of the processing of an integration event or common view.

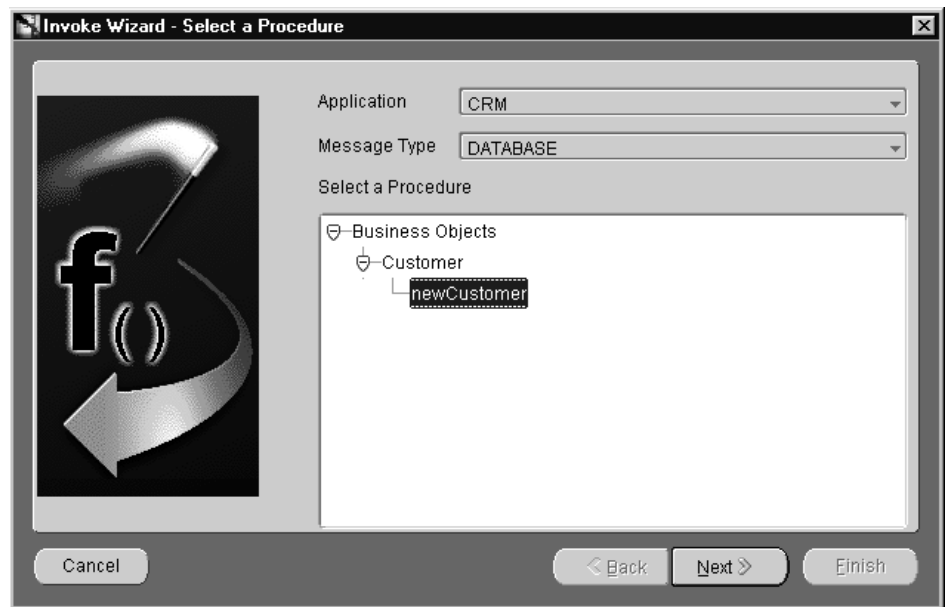
A common view procedure is an event that can be invoked or implemented through the use of a request/reply paradigm, also known as synchronous invocation. The procedure is modeled as an integration business event with the parameter for the invocation and return of the result once the procedure has been executed.

The following example describes how to model an integration event or common view as a procedure.

Invoking a Procedure

Invoking a procedure in iStudio involves using the Invoke Wizard. To start the Invoke Wizard:

1. In the Design Navigation tree, expand the Application node. Select and expand the Application node to display the Invoked Procedures leaf. Right-click Invoke Procedures and select Invoke. The Invoke Wizard displays:



- a. Use this page to enter information for the following fields:

Application—The name of the application invoking the procedure. Select an application from the drop down list.

Message Type—This field specifies the mode of communication between OracleAS InterConnect and the application. Select from the following message types:

Database—OracleAS InterConnect communicates with the application using the database.

Generic—OracleAS InterConnect communicates with the application using a user-defined bridge.

XML—OracleAS InterConnect communicates with the application using XML data described through a DTD using the FTP, SMTP, HTTP, MQ Series, or user-defined adapters.

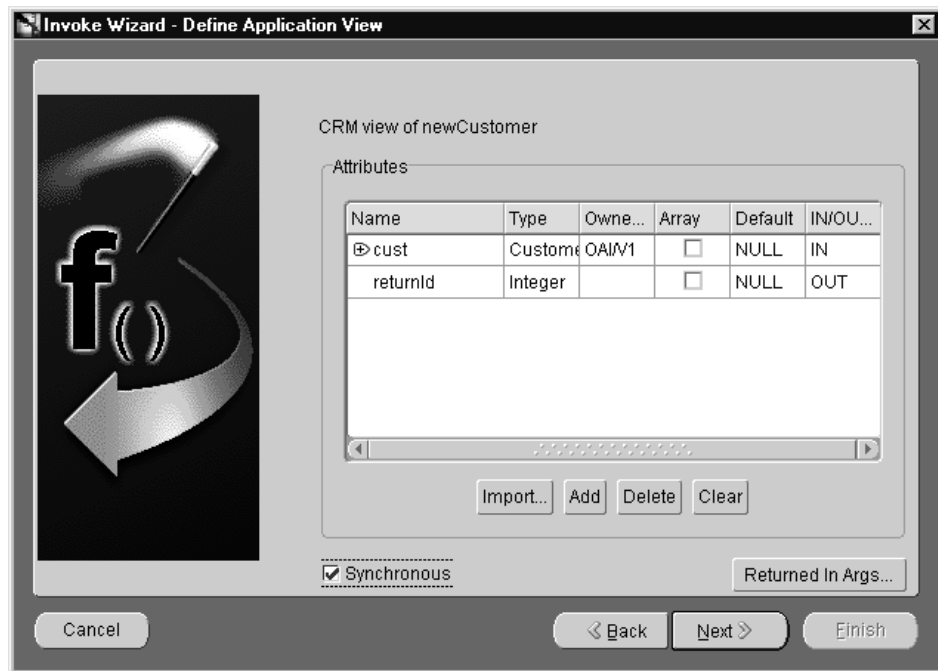
AQ—OracleAS InterConnect communicates with the application through Oracle Advanced Queues using the Advanced Queue adapter. The payload can be Oracle Objects where fields may be XML or RAW XML.

D3L—The adapter communicates with the application using D3L.

- b. Select the procedure to invoke in the Select a Procedure box.
- c. Click Next.

2. Define Application View Page

After clicking Next on the Select a Procedure page, the Define Application View page displays:



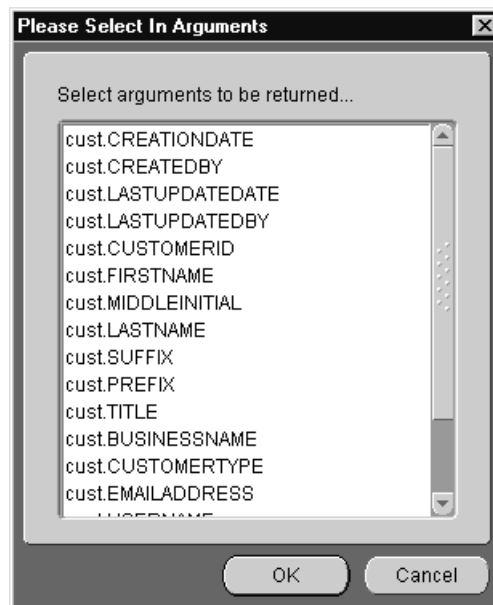
Once a procedure is selected to invoke, the application view is defined. The application view page is initially an empty table. Define the attributes using

Add or import the definitions from a database or an API Repository using Import.

- a. Add or import attributes by clicking Add or Import.

See Also:

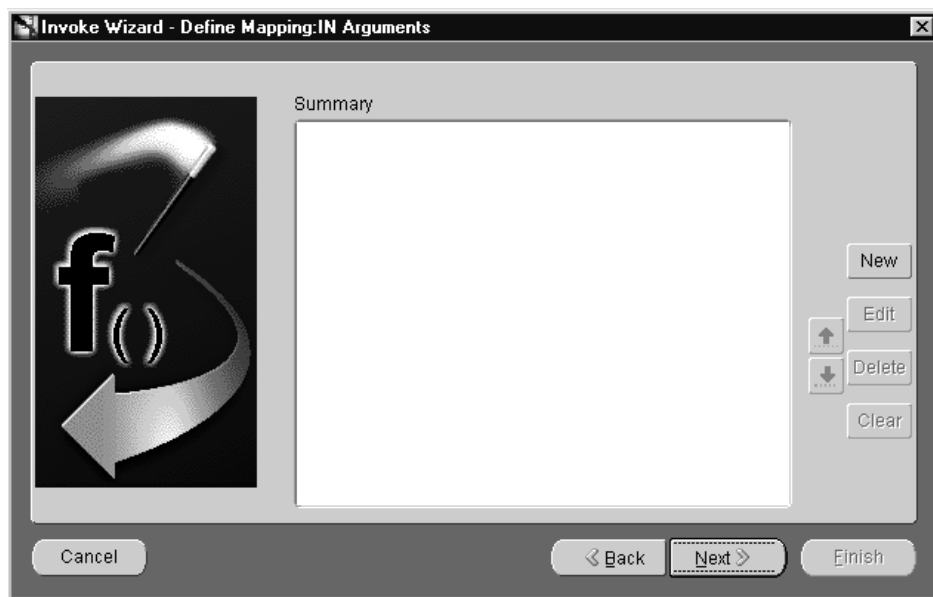
- ["Adding Attributes"](#) on page 3-5 for more information on adding attributes
 - ["Importing Attributes"](#) on page 3-6 for more information on importing attributes
 - ["Deleting and Clearing Attributes"](#) on page 3-8 for more information on deleting attributes
 - [Appendix B, "Using the Data Definition Description Language"](#)
- b. Check the Synchronous box if this is a synchronous invoke. For example, if the request will block until a reply is received.
 - c. To specify IN arguments to be returned, click Returned In Args. The Please Select In Arguments dialog displays:



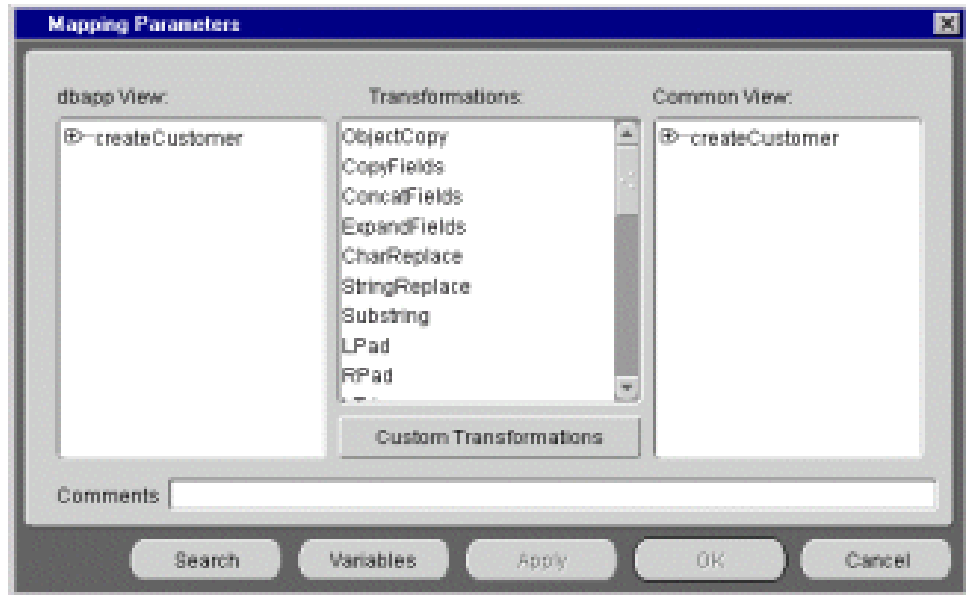
- d. Select the input and output arguments to be returned. Use the left mouse button to select multiple arguments. Only non user-defined input arguments are shown for selection.
- e. Click OK to return to the Define Application View page.
- f. Click Next.

3. Define Mapping IN Arguments Page

Mapping arguments involves copying the individual fields or simple shape-change transformations. After clicking Next on the Define Application View page, the Define Mapping IN Arguments page displays:



- a. Click New to define mappings. The Mapping Parameters page displays:



To map fields in the application view to fields in the common view, use a transform. For example, to map fields in the `FirstName` and `LastName` in the common view to `Name` in the application view, use the `ExpandFields` transform.

See Also:

- ["Invoking a Procedure"](#) on page 5-5 for detailed instructions using the Mapping dialog
- ["Adding Custom Transformations"](#) on page 6-11 for more information on creating custom transformations

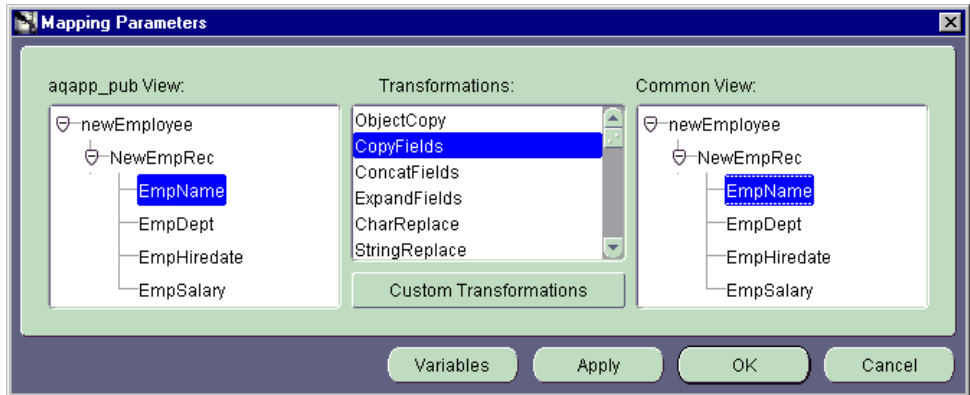
The following steps illustrate this example:

1. Select fields to map from in the application view. Use the left mouse button to select multiple fields in a view.
2. Select the transformation, for example, `ExpandFields`.

3. Select the fields to map to in the common view. Use the left mouse button to select multiple fields in a view.
4. Click Apply to confirm selection and continue specifying additional mappings.
5. When all mappings have been made, click OK.

See Also: ["Adding Custom Transformations"](#) on page 6-11

The transformation may have parameters. If the Apply button is clicked, the Mapping Parameters dialog displays:

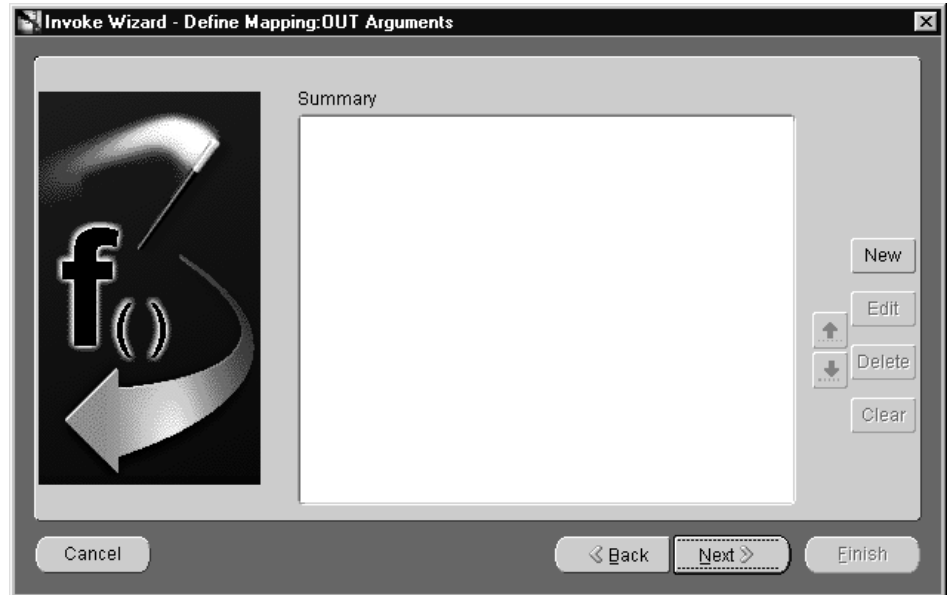


In the Parameters field, enter the values for the transformation parameters. For example, a blank value indicates a value for the separator parameter.

- b. Click Next.

4. Define Mapping OUT Arguments Page

Mapping arguments involves copying the individual fields or simple shape-change transformations. Use this page to map the common view return arguments to the application view return arguments.



- a. Click New to define mappings.

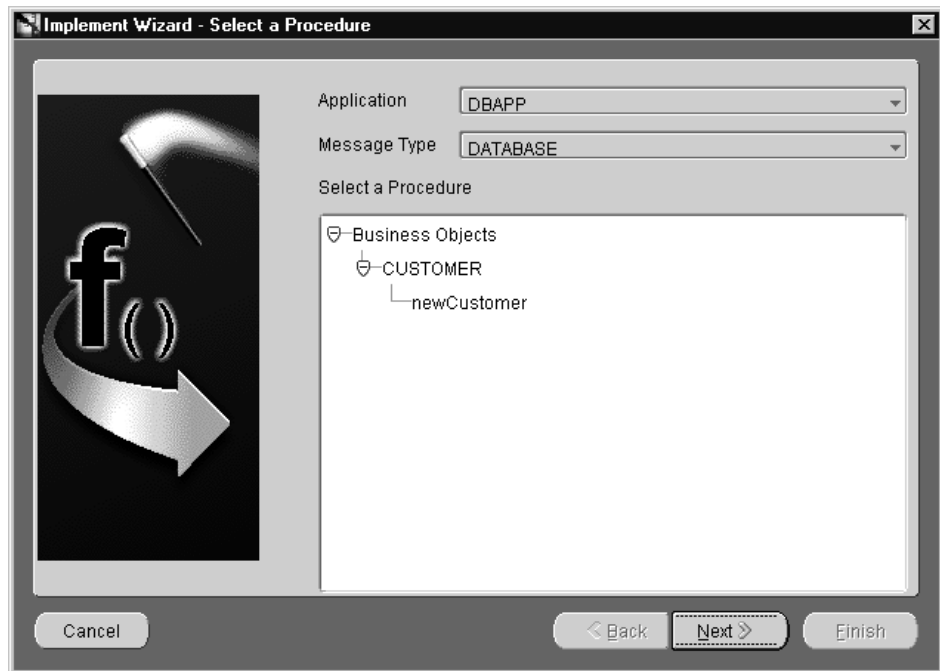
See Also: ["Define Application View Page"](#) on page 5-6

- b. Click Finish.

Implementing a Procedure

Implementing a procedure in iStudio involves using the Implement Wizard. To start the Implement Wizard:

1. In the Design Navigation tree, expand the Application node. Select and expand the Application node to display the Implemented Procedures leaf. Right-click Implemented Procedures and select Implement. The Implement Wizard displays:



Use this page to select a procedure to implement.

- a. Select information for the following fields:

Application—The name of the application implementing the procedure. Select an application from the drop down list.

Message Type—This field specifies the mode of communication between OracleAS InterConnect and the application. Select from the following message types:

Database—OracleAS InterConnect communicates with the application using the database.

Generic—OracleAS InterConnect communicates with the application using a user-defined bridge.

XML—OracleAS InterConnect communicates with the application using XML data described through a DTD using the FTP, SMTP, HTTP, MQ Series, or user-defined adapters.

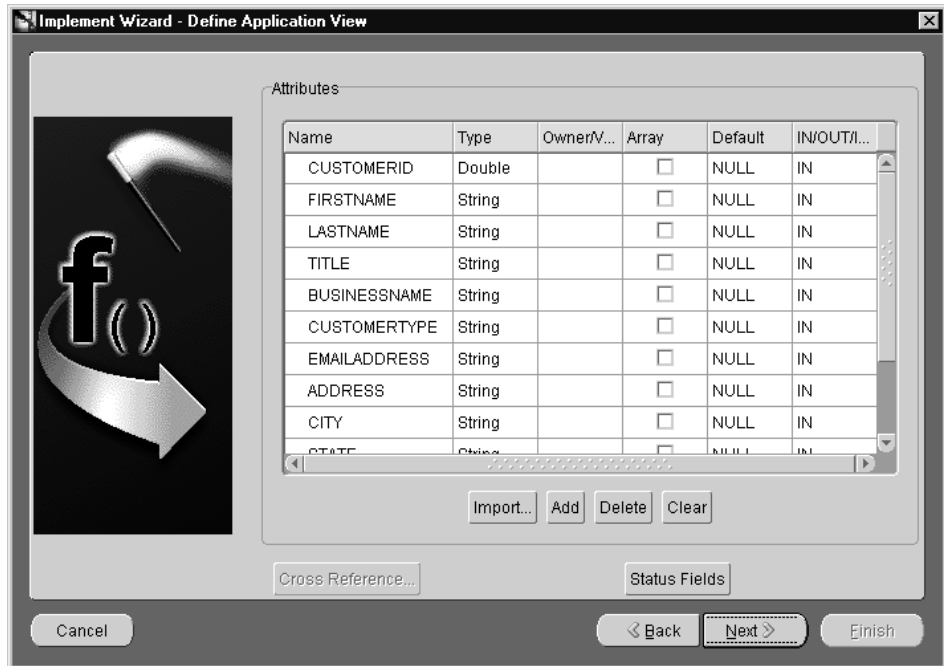
AQ—OracleAS InterConnect communicates with the application through Oracle Advanced Queues using the Advanced Queue adapter. The payload can be Oracle Objects where fields may be XML or RAW XML.

D3L—The adapter communicates with the application using D3L.

- b. Select the procedure to invoke.
- c. Click Next.

2. Define Application View Page

After selecting the procedure to implement, the Define Application View page displays:



Initially, this page is an empty table, used to define the application view. Attributes can be defined by using Add. Attribute definitions can be imported from a database or an API Repository by using Import.

- a. Add or import attributes by clicking Add or Import.

See Also:

- ["Adding Attributes"](#) on page 3-5
- ["Importing Attributes"](#) on page 3-6
- ["Deleting and Clearing Attributes"](#) on page 3-8
- [Appendix B, "Using the Data Definition Description Language"](#)

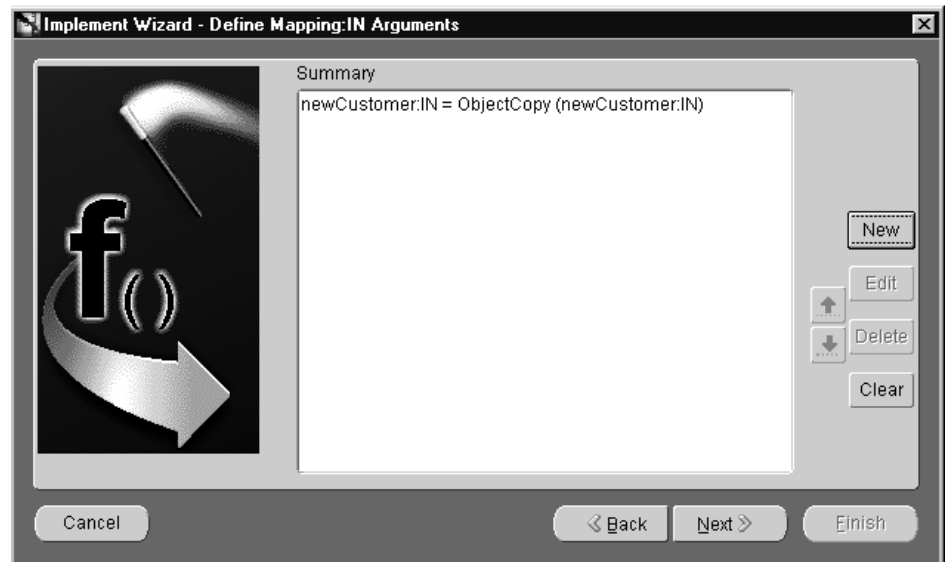
- b. Click Cross Reference... to populate cross reference tables.

See Also: ["Populating Cross Reference Tables"](#) on page 6-8

- c. Click Next.

3. Define Mapping IN Arguments Page

Mapping may involve copying individual fields, or simple shape-change transformations. After clicking next on the Define Application View page, the Define Mapping IN Arguments page displays:



Use this page to define mapping IN arguments.

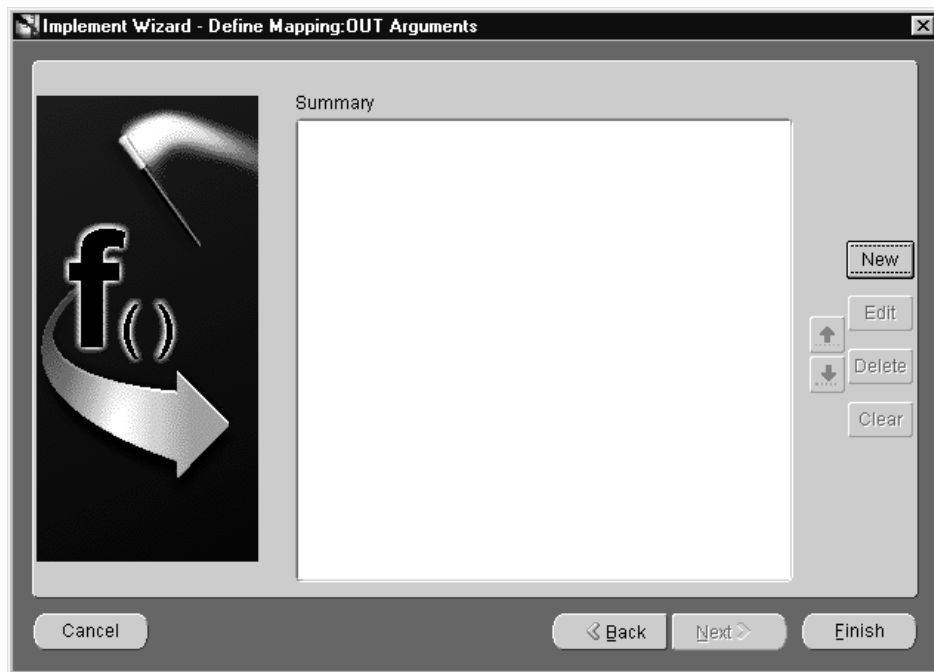
- a. Click New to define mappings.

See Also: ["Define Application View Page"](#) on page 5-6

- b. Click Next.

4. Define Mapping OUT Arguments Page

After clicking Next on the Define Mapping IN Arguments page, the Define Mapping OUT Arguments page displays:



Use this page to define mapping OUT arguments.

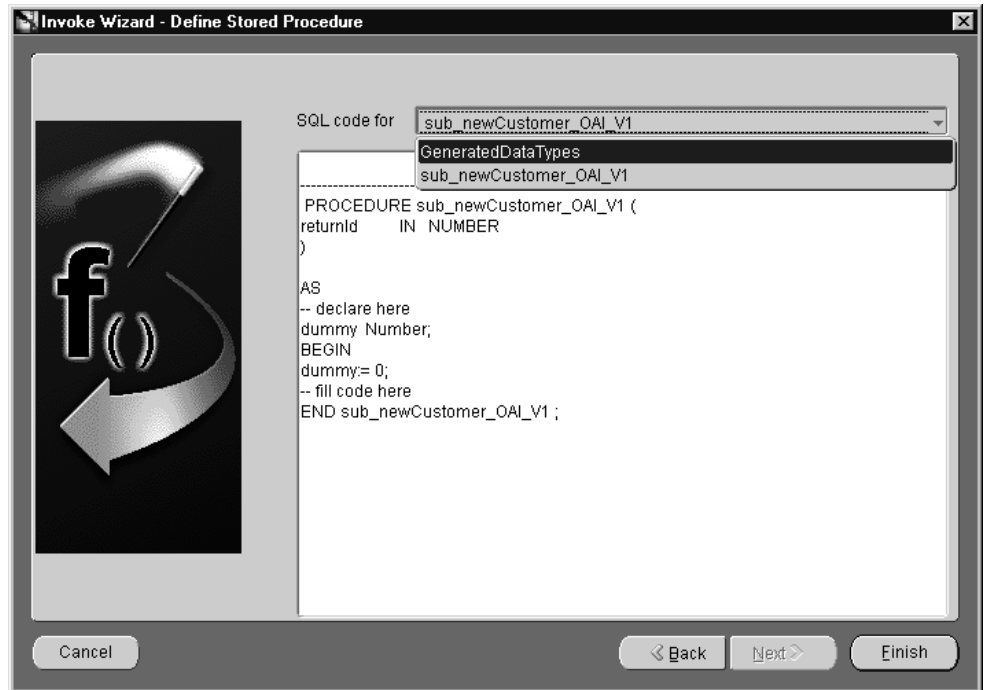
- a. Click New to define mappings.

See Also: ["Define Application View Page"](#) on page 5-6

- b. Click Next.

5. Define Stored Procedure Page

After clicking Next on the Define Mapping OUT Arguments page, the Define Stored Procedure page displays:



If the message type selected was database, the data is received by a stored procedure. In this stored procedure, the action performed when the values are returned to the application can be specified. The adapter invokes the stored procedure at runtime with the appropriate data.

The following arguments will be returned:

- All OUT arguments.
- All IN arguments specified to be returned as part of the reply.
 - a. Select a generated procedure from the SQL code for drop down list.
 - b. Click Finish.

Exporting Stored Procedures

iStudio generates stored-procedure stubs to enable an application to interface with the OracleAS InterConnect run-time easily. These stubs are exported to a file using the export functionality.

To export stored procedures:

1. Select File from the menu bar, then select Export. The Export Application dialog displays:



2. Select the messages to export stored procedures. Messages can be filtered as follows:
 - Export all messages—Select Applications at the top of the directory.
 - Export all messages of a certain type for all applications—Check All Applications, then select one or more types of messages to export.
 - Export all messages for a specific application—Select the application name.
 - Export all messages of a certain type for a specific application—Select the type under the application name in the directory.
 - To export specific messages—Select the messages by name. To select more than one message or class of messages click the application.

3. Enter the name of the file to contain the exported stored procedures in the File Prefix field. The name generates multiple files.
To view the directory page, click Browse.
4. Click OK. The stored procedure is now exported.

Enabling Infrastructure

This chapter describes the enabling infrastructure tasks in iStudio. This chapter discusses the following topics:

- [Enabling Infrastructure](#)
- [Working with Content-Based Routing](#)
- [Working with Cross Reference Tables](#)
- [Working with Domain Value Mappings](#)
- [Cross Reference Table Walk-Through](#)
- [Domain Value Mappings Walk-Through](#)

Enabling Infrastructure

The following topics discuss concepts related to the Enabling Infrastructure branch of the Design Navigation tree.

Content-Based Routing

Messages routed to specific applications based on business rules or message content.

Cross Reference Tables

Creating a cross reference table correlates a unique key in one application with a corresponding key in the common view. In this process, three separate events are addressed:

- Inserting a record.
- Updating a record.
- Deleting a record.

Domain Value Maps

Creating a domain value map correlates a value in one application to a value in a second application. In this process, two separate events are addressed:

- Inserting a record.
- Updating a record.

Working with Content-Based Routing

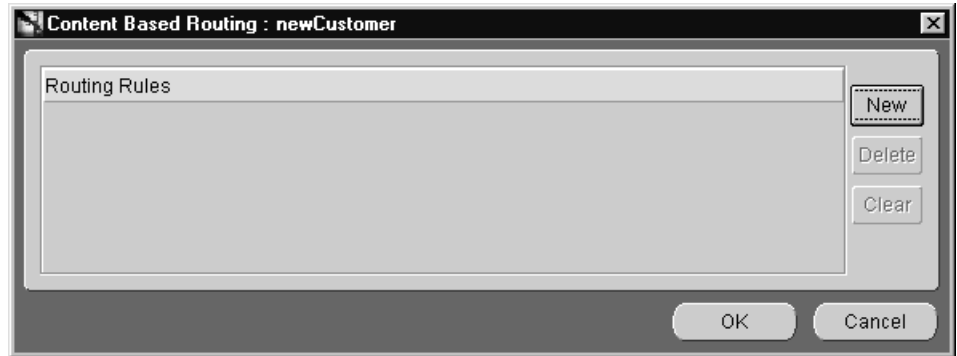
Messages are routed to specific applications based on business rules or message content. For example, a procurement system can route fulfillment requests to different fulfillment centers based on originating location or item requested.

Modifying Content-Based Routing

To modify content based routing for an event or procedure:

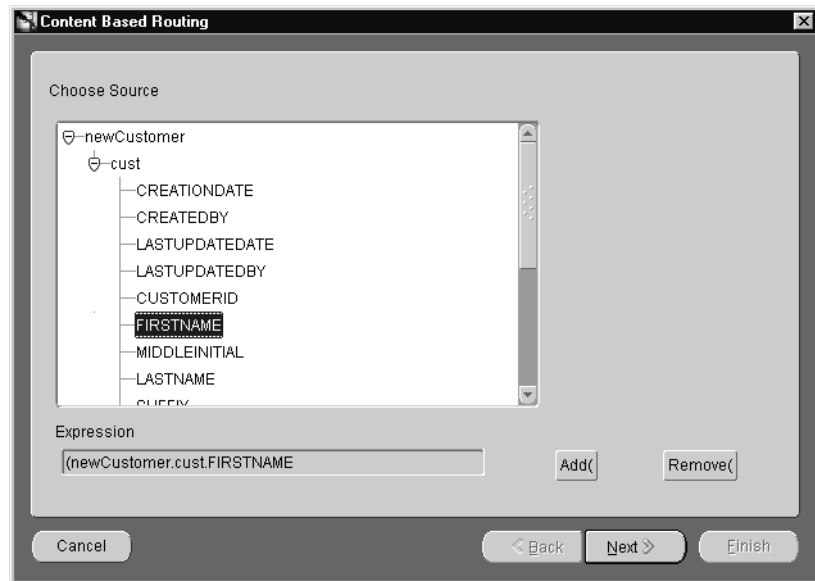
1. Right-click the event or procedure under the Content-Based Routing node in the Design Navigation tree, then click Edit. The Content Based Routing Rules dialog displays.

2. Click New to display the Content Based Routing Wizard. This wizard provides a series of pages to follow for editing content based routing.



3. Choose Source Page

When the Content Based Routing Wizard starts, the Choose Source page displays:



Choose the source event attribute and click Next.

4. Choose Operator Page

After the source event is selected, the Choose Operator page displays:

Select an operator from the drop-down list and click Next.

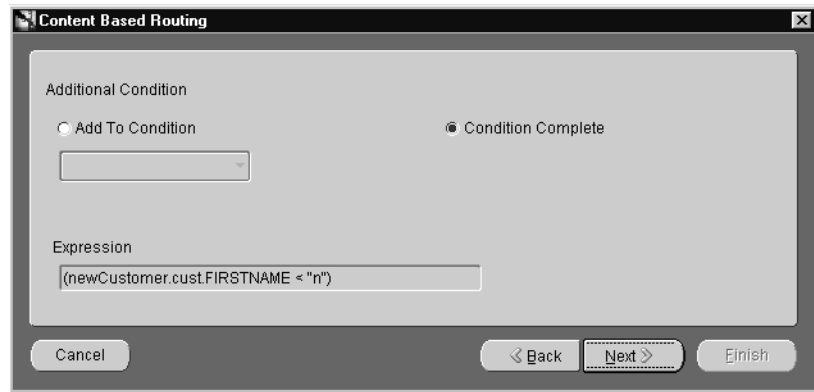
5. Choose Value Page

After selecting an operator, the Choose Value page displays:

- a. Use the Enter Value field to compare a value in an attribute to a literal. Enter a value in the text field.
Use Select Attribute to compare one value in an attribute to another. Select an attribute from the navigation tree.
- b. Click Next.

6. Additional Condition Page

After entering a value or selecting an attribute, the Additional Condition page displays:

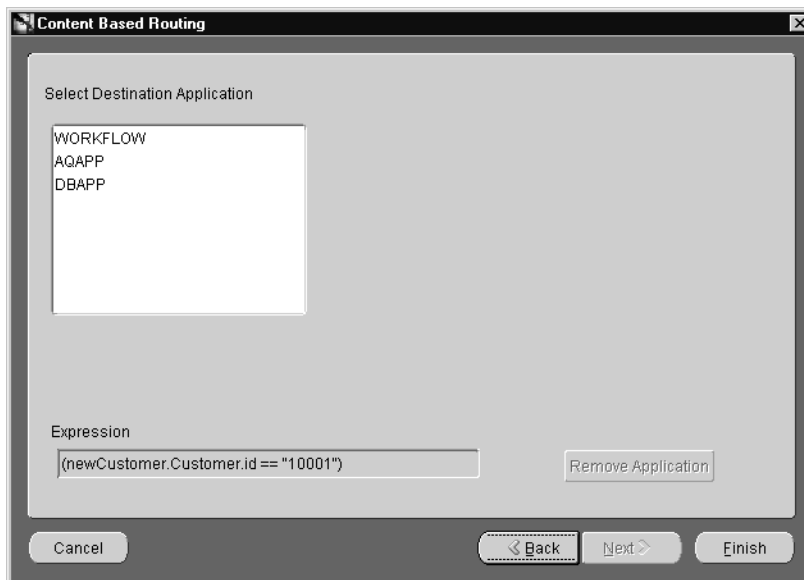


The screenshot shows a dialog box titled "Content Based Routing" with a close button (X) in the top right corner. The main area is titled "Additional Condition" and contains two radio buttons: "Add To Condition" (unselected) and "Condition Complete" (selected). Below the radio buttons is a text input field. Underneath is the "Expression" section with a text input field containing the expression `(newCustomer.cust.FIRSTNAME < "n")`. At the bottom of the dialog are four buttons: "Cancel", "Back" (with a left arrow), "Next" (with a right arrow), and "Finish".

- a. Select Add to Condition to have complex evaluations, such as `Age > 30, Salary <= 2,000 and State = CA.`
Select Condition Complete to continue to the next page.
- b. Click Next.
Depending on which radio button is selected, the Choose Source page or Select Destination Application page displays.

7. Select Destination Application Page

In this example, the Select Destination page is used:



Select an application from the Select Destination Application box and click Finish.

The Content Based Routing Rule is created and displays on the Content Based Routing dialog.

8. Click OK.

Working with Cross Reference Tables

Keys for corresponding entities created in different applications can be correlated through cross referencing in iStudio. For example, a purchase order created in a procurement system has a native id `PurchaseOrder`. It is then routed to a fulfillment system. The purchase order is created in the fulfillment system with native id `Order_ID`. `PurchaseOrder` and `Order_ID` must be cross referenced so OracleAS InterConnect can correlate communication about this same logical entity in two different systems without each system knowing the native ids of the other.

Creating Cross-Reference Tables

Creating a cross reference in iStudio creates a table in the repository schema. To create a cross reference table:

1. From the File menu, click **New**, then select **Cross Reference Tables**. The **Create Cross Reference Table** dialog displays.
2. Enter a name for the cross reference table in the **Table Name** field and click **OK**.

Adding Applications to Cross Reference Tables

To add applications to the cross reference table:

1. In the **Design Navigation tree**, select the appropriate cross reference table and right-click to display the context menu.
2. From the context menu, select **Add App**. The **Add Application to Cross Reference Table** dialog displays.
3. From the drop down list, select an application name.
4. Click **OK**.

Removing Applications From Cross Reference Tables

To remove applications from a cross reference table:

1. In the **Design Navigation tree**, select the appropriate cross reference table and right-click to display the context menu.
2. From the context menu, select **Remove App**. The **Remove Application from Cross Reference Table** dialog displays.

3. From the drop down list, select an application name.
4. Click OK.

Populating Cross Reference Tables

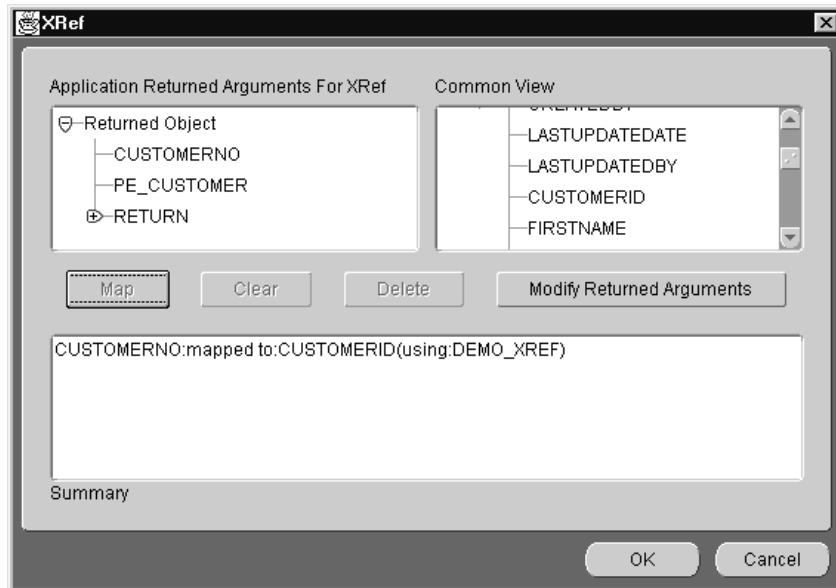
To populate the cross reference tables, returned arguments must first be defined. Returned arguments are the arguments returned by the subscribe or implement code for populating the cross reference table.

Use the Subscribe Wizard to access the correct page for populating cross reference tables.

See Also: ["Subscribing to an Event"](#) on page 4-12

To populate cross reference tables:

1. Click Cross Reference... on the Define Application View page on the appropriate wizard. The XRef dialog displays:



The Application Returned Arguments box displays the returned arguments. This information is initially populated with any OUT arguments from the application view.

2. Click Modify Return Arguments to modify the returned arguments list.
3. Select the corresponding attributes in the Application Returned Arguments For XRef and Common View windows, then click Map.
4. Specify the Cross Reference Table name to be populated using these attributes' values and click OK.

Working with Domain Value Mappings

Code tables can be mapped across systems using domain value mapping in iStudio. For example, a purchase order in a procurement system has a purchase order status field with possible domain values of `Booked` and `Shipped`. The corresponding field in a fulfillment system has the domain value set of 1 and 2. OracleAS InterConnect creates mappings such as `booked=1` and `shipped=2` so these values can be correlated at runtime without each system knowing the domain value set of the other.

Creating a Domain Value Mapping Table

To create a domain value mappings table:

1. Select File menu from the menu bar, select New, then select Domain Value Mapping. The Create Domain Value Mapping dialog displays.
2. Enter a name for the domain value map in the Map Name field and click OK.

Adding Applications to Domain Value Mappings

To add applications to domain value mappings:

1. In the Design Navigation tree, select a domain value mapping and right-click to display the context menu.
2. From the context menu, select Add App. The Add Application to Domain Value Map dialog displays.
3. Select an application name from the drop down list and click OK.

Removing Applications From Domain Value Mappings

To remove applications from the domain value mappings:

1. In the Design Navigation tree, select a domain value mapping and right-click to display the context menu.

2. From the context menu, select Remove App. The Remove Application from Domain Value Mapping dialog displays.
3. Select the Application Name to remove from the drop down list and click OK.

Modifying Domain Value Mappings

To modify data domain value mappings:

1. In the Design Navigation tree, select a domain value mapping and right-click to display the context menu.
2. From the context menu, select Edit Domain Value Map. The Edit Domain Value Map dialog displays.
3. Click Add to add mappings or Import to import mappings.
4. Click OK.

Deleting Domain Value Mappings

To delete a selected domain value mapping:

- Select the domain value mapping to delete and click Delete.

Deleting Domain Value Mapping Tables

To delete the domain value mapping table:

1. Select the domain value mapping table to be deleted and right-click to display the context menu.
2. From the context menu, select Delete.
3. In the Confirm Delete dialog, click Yes.

Modifying Attribute Mappings

To modify a selected attribute mapping, use the Define Mapping page on the Publish Wizard:

See Also: ["Publishing an Event"](#) on page 4-5

1. Select a mapping and click Edit.
2. Edit the appropriate fields and click OK.

Removing Attribute Mappings

In the Publish Wizard, use the Define Mapping page to remove attribute mappings.

- To remove a mapping, delete the attribute and click Remove.
- To remove all mappings, click Clear.

Adding Custom Transformations

Use the Define Mapping page of the Publish Wizard to create custom transformations.

See Also: ["Publishing an Event"](#) on page 4-5

To create custom transformations:

1. Click Custom Transformations on the Mapping dialog. The Transformations dialog displays:



2. To add user-defined transformations, click Add.
3. Click OK.

Deleting Custom Transformations

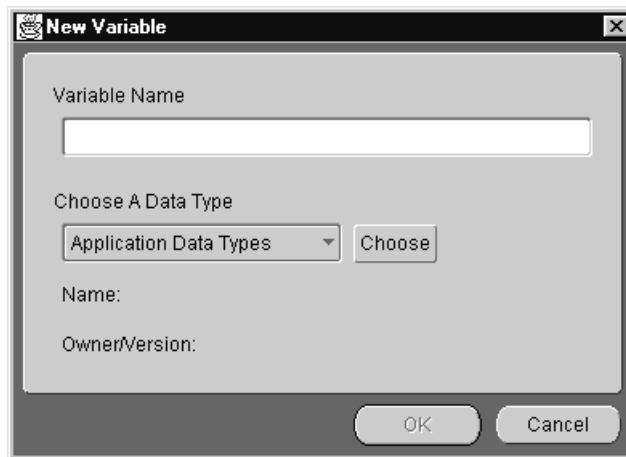
Using the Transformation dialog, you can delete custom transformations. To delete a selected transformation:

1. Select the transformation to delete and click Delete.
2. Click OK on the Transformation dialog.

Adding Mapping Variables

To add mapping variables, use the Mapping dialog.

1. On the Mapping dialog, click Variables. The Variables dialog displays.
2. Click Add to add a variable. The New Variable dialog displays:



The image shows a 'New Variable' dialog box. It has a title bar with the text 'New Variable' and a close button (X). The main area contains a text input field labeled 'Variable Name'. Below that is a section titled 'Choose A Data Type' which includes a dropdown menu currently showing 'Application Data Types' and a 'Choose' button. Underneath are labels for 'Name:' and 'Owner/Version:'. At the bottom of the dialog are two buttons: 'OK' and 'Cancel'.

3. Enter a name for the variable in the Variable Name field.
4. Select a data type from the drop down list and click Choose. The data type for the variable must be previously defined.
5. Click OK to return to the New Variable dialog.
6. Click OK on the New Variable dialog.

Deleting a Mapping Variable

To delete a mapping variable, use the Mapping dialog:

1. Select the mapping variable to delete and click Delete.
2. Click OK.

Cross Reference Table Walk-Through

The following set of steps is a walk-through involving cross-reference tables.

1. Create a cross reference table using iStudio. Creating a cross reference node creates a table in the repository schema.
2. Add applications to the cross reference table. This defines the cross reference.
3. Create a cross reference in the mapping in the subscribing application. This step involves the Subscribe Wizard.
4. Define the return argument(s) for the key using the XRef dialog. The code must return the new key from the subscribing application. This key and the publishing application key will then be inserted into the cross reference table.

See Also: ["Creating Cross-Reference Tables"](#) on page 6-7

5. Map the common view key to the application key using a transformation in the subscribing application.
6. Set the table name and options after the mapping is applied.
7. Define syntax for the mapping.
8. Define a stored procedure.
9. Map the common view key to the application key using a transformation in the subscribing application.
10. Set the table name after the mapping is applied.

See Also: [Chapter 4, "Using Events in iStudio"](#)

Domain Value Mappings Walk-Through

The following set of steps is a walk-through involving domain value mappings.

1. Create the domain value mapping.
2. Add applications and values to the mapping.

See Also: [Chapter 6, "Enabling Infrastructure"](#)

3. Create a mapping using a transformation in the subscribing application.
4. Set the table name and options after the mapping is applied.
5. Define syntax for the mapping.

See Also: [Chapter 4, "Using Events in iStudio"](#)

Using Oracle Workflow

This chapter discusses using Oracle Workflow to apply business logic to an integration. Topics include:

- [Oracle Workflow Overview](#)
- [OracleAS InterConnect Integration with Oracle Workflow](#)
- [Using Oracle Workflow to Apply Business Logic](#)
- [Design Business Process](#)

Oracle Workflow Overview

Oracle Workflow is integrated with OracleAS InterConnect and is used for business process collaborations across two or more applications. A business process collaboration is defined as the conversation between two or more applications in the context of a business process.

OracleAS InterConnect leverages the robust design time and runtime Oracle Workflow business process definition and execution support to make business processes explicit and manageable.

Note: Knowledge of Oracle Workflow, its tools, and its Business Event System is required to utilize OracleAS InterConnect and Oracle Workflow for business process collaboration. For more information on Oracle Workflow, see *Oracle Workflow Administrator's Guide*.

Oracle Workflow Solves Common Business Problems

The following are some of the common business problems that can be solved using Oracle Workflow.

Error Management

If there is a problem in a conversation between two or more applications, the errors arising from this problem can be centrally managed and appropriate remedial actions can be defined. For example, it may be required to keep the data of an order entry system synchronize with a backend ERP system. Consider that a new purchase order is created in the order entry system but the ERP system is down at the time the purchase order is created. At a later time, the ERP system comes back up and an attempt is made to create a corresponding new purchase order through messaging using OracleAS InterConnect. This attempt fails. To deal with this scenario, the integrator can utilize Oracle Workflow to send a compensating message to the order entry system to undo the creation of the purchase order and notify the user who created the order.

In the example above, OracleAS InterConnect and Oracle Workflow can be used to model the following for every purchase order that is over \$50,000:

- Send a notification to a named approver and wait for approval.
- If approved, send the message to the ERP system. Otherwise send a message to the order entry system to rollback the order creation.

Message Junctions

Fan-in and fan-out of messages can be effectively modeled using OracleAS InterConnect and Oracle Workflow. Fan-in messages involve combining two or more messages into one. Fan-out messages involve splitting one message into two or more.

For an example of fan-in messaging, consider the following. A global organization has a centralized Human Resources Enterprise Resource Planning (ERP) application in the United States. Each country has one or more local systems that capture local employee information. If a new employee joins the Japanese branch of this organization, data is entered into a local human resources application and a local Benefits application. Each entry launches a message for adding this information to the centralized system. However, the centralized system needs data from both systems combined and will only commit the data if it was entered successfully in both of the local systems connecting to OracleAS InterConnect. Using Oracle Workflow, this process can be modeled so that OracleAS InterConnect routes messages from both local systems to Oracle Workflow, Oracle Workflow waits until it receives both messages, combines the data, and launches a single message to be delivered by OracleAS InterConnect to the centralized human resources system.

Stateful Routing

OracleAS InterConnect provides extensive support for stateless routing through event-based and content-based routing features. Using Oracle Workflow, stateful routing can be accomplished. In other words, the decision to route can be based on more than the event or the content of the message.

Composite Services

Combining all of the examples, an internal (organization focused) or external (customer/partner focused) service can be built through a well-defined set of business processes involving communication between two or more applications. For example, a brick-and-mortar retail company wants to provide an on-line procurement service to their customers. Behind the user interface are several business processes controlling communication across several internal applications to deliver a robust, performant service to the customer.

Note: The ability to define explicit business process collaborations is a feature, not a requirement for completing integrations. It is not necessary to utilize Oracle Workflow for integration if the business process definition is simple enough to be implicitly captured in the messaging through the core functionality in iStudio.

OracleAS InterConnect Integration with Oracle Workflow

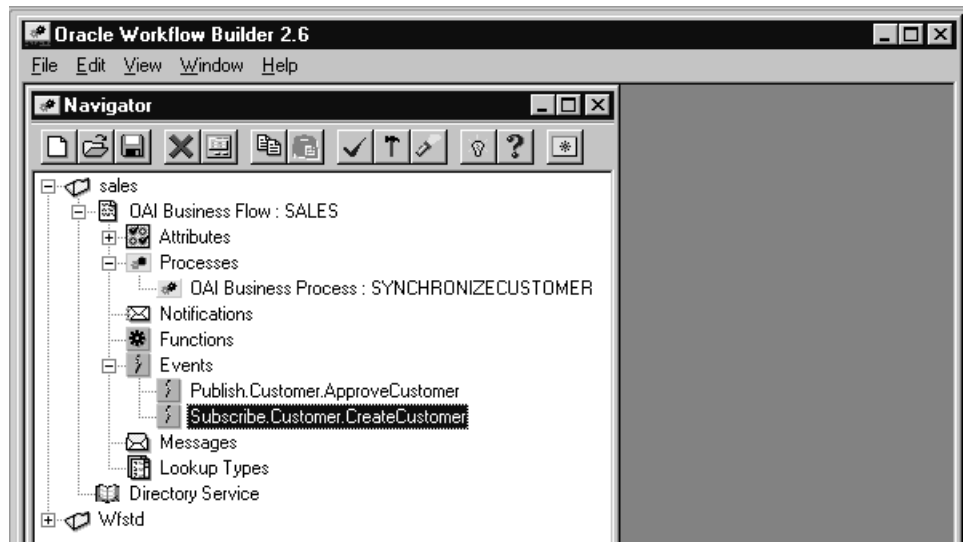
The following describes how OracleAS InterConnect and Oracle Workflow are integrated.

Design Time Tools

During design time, business process and event definitions in iStudio can be deployed to Oracle Workflow. Consequently, Oracle Workflow tools can be launched from within iStudio to graphically create process diagrams in the context of enterprise integration through OracleAS InterConnect.

Using iStudio, the following tools can be launched:

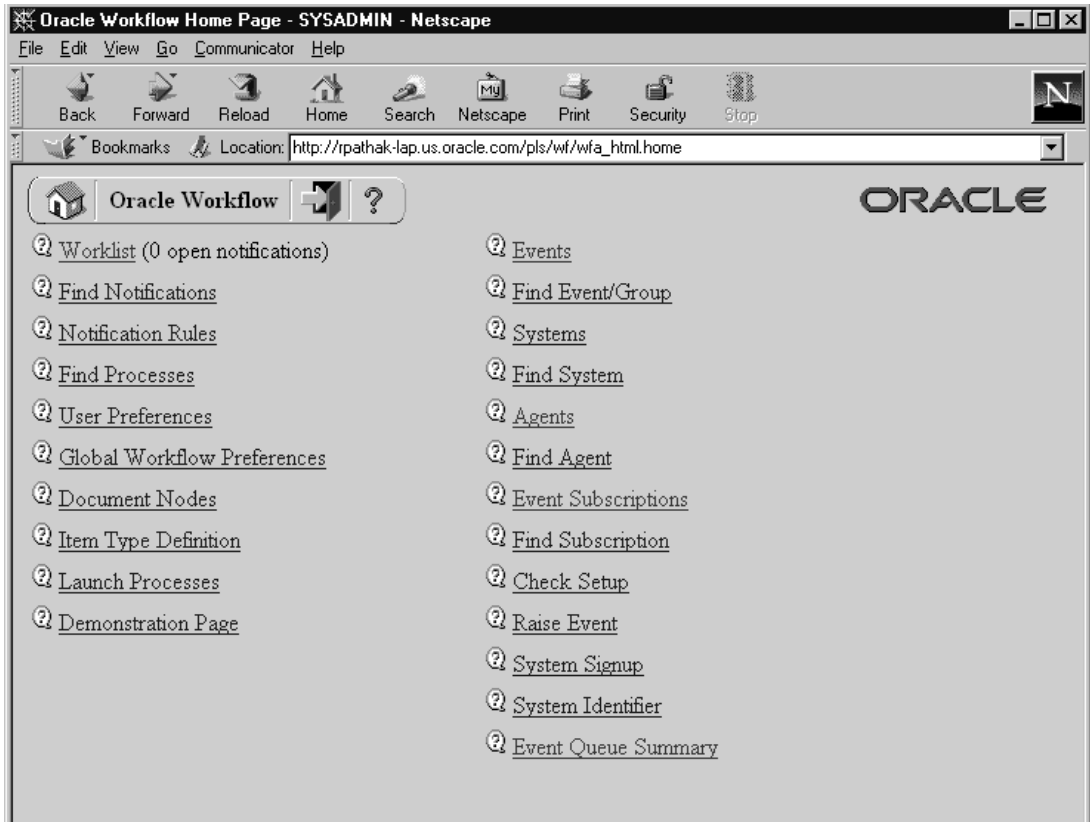
- Oracle Workflow Builder—Use this tool to complete business process definitions defined and deployed through iStudio. This is accomplished by using Oracle Workflow Builder to create process diagrams.

Figure 7–1 Oracle Workflow Builder

- Oracle Workflow Home Page—Use this tool for centralized access to the web-based features of Oracle Workflow. The Business Event System management and administration is the most important feature exposed through this home page.

For more information on the Business Event System, see *Oracle Workflow Administrator's Guide*.

Figure 7-2 Oracle Workflow Home Page



Runtime

At runtime, OracleAS InterConnect integrates with the Business Event System of Oracle Workflow. The Business Event System is an application service that uses the Advanced Queueing infrastructure to communicate business events between systems. OracleAS InterConnect registers itself as an external system in Business Event System so the following conditions exist:

- Messages can flow from applications through OracleAS InterConnect, in the common view format, to the Business Event System. This will either trigger or continue Oracle Workflow business processes as defined using iStudio (Business Processes) and described using Oracle Workflow Builder (Process Diagrams).
- Messages can flow from the Business Event System to OracleAS InterConnect in the common view format to applications to either continue or end Oracle Workflow business processes.

At runtime, Oracle Workflow is integrated with OracleAS InterConnect at the hub. Messages are passed back and forth between OracleAS InterConnect and the Business Event System of Oracle Workflow via Advanced Queues. The OracleAS InterConnect Oracle Workflow Communication Infrastructure facilitates this communication.

At design time, to keep the integration methodology consistent, iStudio reuses the messaging paradigms of publish/subscribe and request/reply to specify communication between OracleAS InterConnect and Oracle Workflow. Therefore, for messages inbound into Oracle Workflow, the iStudio user can specify, in the context of a business process, which events Oracle Workflow is subscribing to and which procedures Oracle Workflow is implementing. For outbound messages, events that Oracle Workflow can publish and procedures it can invoke can be specified.

See Also: ["Using Oracle Workflow to Apply Business Logic"](#) on page 7-8

Using Oracle Workflow to Apply Business Logic

This section describes using OracleAS InterConnect with Oracle Workflow. There are three broad steps:

1. [Install Oracle Workflow Components](#)
2. [Design Business Process](#)
3. [Deploy Business Processes for Runtime](#)

Install Oracle Workflow Components

You need to install the necessary Oracle Workflow components and perform required configuration tasks following the procedures described in *Oracle Application Server InterConnect Installation Guide*.

Design Business Process

To deploy the business process:

- Design process bundles using iStudio.
- Deploy process bundles from iStudio to a `.wft` file.
- Complete process diagrams in Oracle Workflow Builder by launching Oracle Workflow Builder from iStudio and using the deployed `.wft` file.

Deploy Business Processes for Runtime

To deploy business processes for runtime:

- Deploy events to the Business Event System from iStudio.
- Deploy a process diagram from a file to the database using Oracle Workflow Builder.

Design Business Process

The following concepts discuss how iStudio and Oracle Workflow work together in OracleAS InterConnect. In addition, these topics discuss how to use iStudio and Oracle Workflow step by step during design time for business process collaborations across applications.

Process Bundle

A process bundle is a set of logically related business processes. This maps one-to-one with an Oracle Workflow item.

See Also: ["Business Process"](#) on page 7-9

Business Process

A business process is a set of OracleAS InterConnect common view events or procedures that must be routed to and from Oracle Workflow in one Oracle Workflow business process. These events and procedures manifest themselves as Oracle Workflow business events and can be used to define a process diagram in Oracle Workflow Builder. This maps one-to-one with an Oracle Workflow business process.

Activity

Activities in iStudio allow the user to define the common view events and procedures that must be a part of an Oracle Workflow business process. The following are types of activities in iStudio:

- **Publish Event**—Oracle Workflow publishes an OracleAS InterConnect common view event. At deployment time, a business event corresponding to the common view event is created in the Business Event System.
- **Subscribe Event**—Oracle Workflow subscribes to an OracleAS InterConnect common view event. At deployment time, a business event corresponding to the common view event is created in the Business Event System.
- **Invoke Procedure**—Oracle Workflow invokes an OracleAS InterConnect common view procedure. At deployment time, two business events corresponding to the common view procedure are created in the Business Event System, one event for sending the request and the other for receiving the reply.
- **Implement Procedure**—Oracle Workflow implements an OracleAS InterConnect common view procedure. At deployment time, two business events corresponding to the common view procedure are created in the Business Event System, one event for receiving the request and the other for sending the reply.

The following table describes how iStudio and Oracle Workflow concepts are mapped.

iStudio Concept	Oracle Workflow Concept	Mapping
Process Bundle	Item	One-to-one.
Business Process	Business Process	One-to-one.
Common View Event	Business Event	One-to-one. ¹
Common View Procedure	Business Event	Two business events per procedure.
Publish Activity	Send Event Activity	One-to-one.
Subscribe Activity	Receive Event Activity	One-to-one.
Invoke Activity	Send Event Activity (for the request) Receive Event Activity (for the reply)	
Implement Activity	Receive Event Activity (for the request) Send Event Activity (for the reply)	

¹ Only for all events that are part of a business process in iStudio. Events that are part of the common view but not part of a business process are not instantiated as Oracle Workflow business events. All common view events need not be part of business processes. In other words, depending on the integration, some common view events could be exchanged directly between applications without involving Oracle Workflow using the core functionality of OracleAS InterConnect. Other events may need to be part of an explicit business process. It is the latter set of events that become business events in Oracle Workflow. The same is true for common view procedures.

Creating a Process Bundle

To create a process bundle using iStudio:

1. From the project tree, click Workflow and expand the subtree.
2. Right-click on Process Bundles and select New. The Create Process Bundle dialog displays.
3. Enter the name of the process bundle in the Process Bundle Name field and click OK.

Creating a Business Process

To create a business process:

1. From the project tree, expand the process bundle for which the business process is to be created.
2. Right-click on Business Processes and select New. The Create Business Process dialog displays.
3. Enter a name for the business process in the Business Process Name field and click OK.

Populating a Business Process with Activities

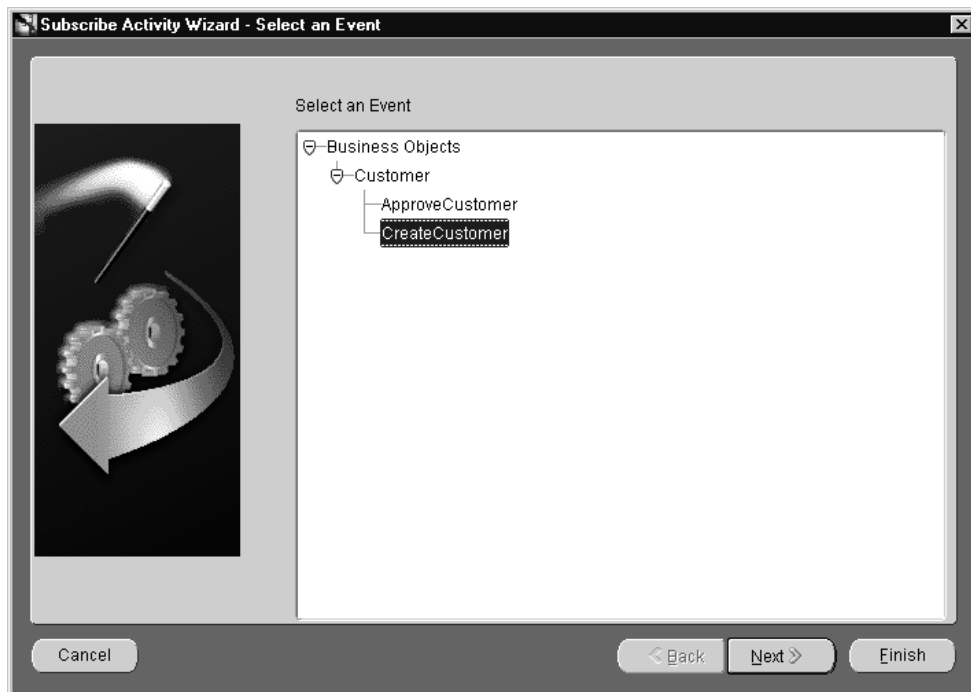
To populate a business process with activities:

1. From the project tree, select the business process to populate.
2. Right-click the business process and in the context menu, select the activity to be part of the business process. Choose from the following activities:
 - New Publish Activity—Oracle Workflow sends a message to OracleAS InterConnect in the context of the business process.
 - New Subscribe Activity—Oracle Workflow receives a message from OracleAS InterConnect.¹
 - New Invoke Activity—Oracle Workflow sends a request message to OracleAS InterConnect and receives a reply.
 - New Implement Activity—Oracle Workflow receives a request from OracleAS InterConnect and sends a reply.

¹ This example is used for explaining the steps. The steps are similar regardless of the selection.

The Subscribe Activity Wizard displays.

3. Select an event for the activity.



4. Click Finish.

Repeat these steps for adding other activities to the process.

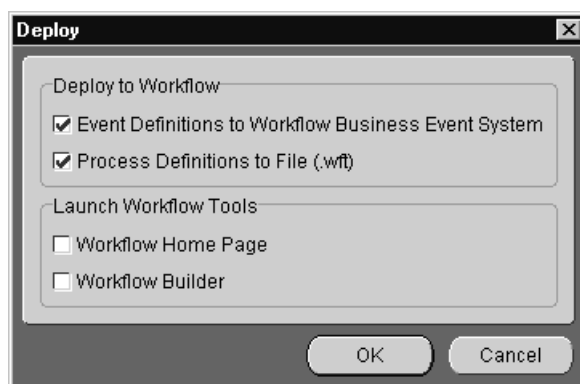
Note: When you create multiple activities under a business process, the list of activities is unordered. For example, the order in which the activities are added is not important. The order can be later decided in Oracle Workflow Builder through a process diagram.

Deploying to Oracle Workflow

After populating business processes with activities, this information must be deployed to Oracle Workflow to graphically model a business process. To deploy this information to Oracle Workflow:

1. On the Deploy tab in iStudio, right click on Workflow and select Deploy.

The Deploy dialog displays:



2. There are two sets of information that need to be deployed. They can be done independently or together:
 - Oracle Workflow Business Events—Business Events need to be created in the Business Event System of Oracle Workflow. This is a requirement for runtime only. Using Oracle Workflow Builder, users can decide not to deploy these until all design time work, including modeling the business process, is complete.

Note: iStudio checks if an event is already deployed before deploying it. Therefore, users can safely decide to re-deploy all events at anytime. However, deploying events after all the design time work has been completed saves you the effort of redeploying events repeatedly.

To check if events have been deployed, launch the Oracle Workflow Home page.

See Also: ["Launching the Oracle Workflow Home Page"](#) on page 7-16 for more information on the Oracle Workflow Home page.

- Oracle Workflow Process Definitions through .wft file generation—Information about business processes captured in iStudio provides a foundation for building process diagrams in Oracle Workflow Builder. Deploying process definitions is required for design time.

Note: When deploying process definitions, iStudio prompts for a filename. If an existing file is specified, iStudio will **overwrite** the file. Therefore, if there are existing process definitions in a file modified using Oracle Workflow Builder, do not select that filename as the target, otherwise all modifications made will be lost.

By default, both choices are selected. The dialog also allows the following to be automatically launched:

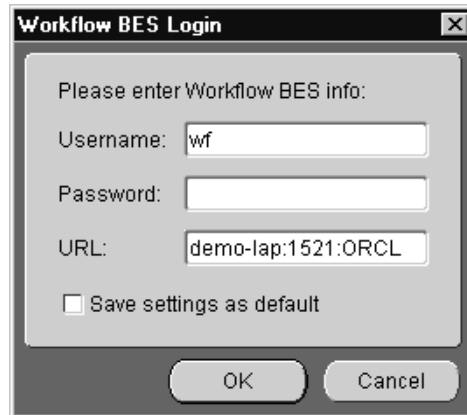
- Oracle Workflow Builder—Defines business process diagrams.
- Oracle Workflow Home Page—Verifies Business Event deployment.

By default, these choices are unselected. Choose to launch these tools with deployment or complete this task at a later time on the Design tab.

See Also: ["Launching Oracle Workflow Tools"](#) on page 7-16

3. Select the appropriate choices and click OK.

If deploying event definitions to the Oracle Workflow Business Event System is selected, the following dialog displays:



4. Enter the required information based on the selections made during Oracle Workflow installation and click OK.

If Deploying Process Definitions to a .wft file was selected, a file dialog displays:



5. Enter the name and location of the file to create and click OK.

Note: When deploying process definitions, iStudio prompts for a filename. If an existing file is specified, iStudio will **overwrite** the file. Therefore, if there are existing process definitions in a file modified using Oracle Workflow Builder, do not select that filename as the target, otherwise all modifications made will be lost.

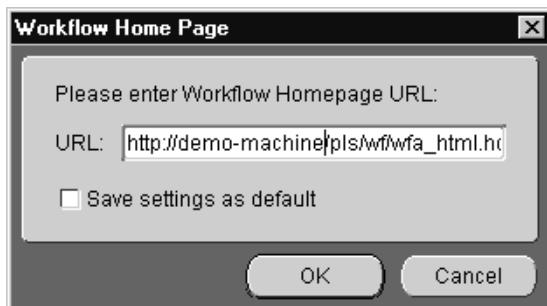
Launching Oracle Workflow Tools

The following topics discuss how to launch Oracle Workflow tools in iStudio.

Launching the Oracle Workflow Home Page

To launch the Oracle Workflow Home page:

1. On the Design tab in iStudio, right click on Workflow and select Launch WF Home Page. The Oracle Workflow Home Page dialog displays:



2. Make sure the URL is correct and click on OK.

The Username and Password Required dialog displays:



3. Enter the login information for the Oracle Workflow Home Page and click OK.
The Oracle Workflow Home page is launched using the default browser.

Launching Oracle Workflow Builder

To launch Oracle Workflow Builder:

1. On the Design tab in iStudio, right click on the process bundle to view in Oracle Workflow Builder and select Launch Workflow Builder. The Deploy dialog displays:



2. In the Deploy dialog, select an existing `.wft` file name to load into Oracle Workflow Builder.

Note: The assumption is that a process definition has already been deployed to a file.

Oracle Workflow Builder is launched depending on which process definition file is selected.

Note: To launch Oracle Workflow Builder outside of a specific OracleAS InterConnect process bundle, right-click on Workflow and select Launch Workflow Builder.

Modifying Existing Oracle Workflow Processes

When modifying existing Oracle Workflow processes, do not add, modify, or remove OracleAS InterConnect event activities directly in Oracle Workflow Builder. Always make all event-related process changes in iStudio, redeploy to the file, and import in Oracle Workflow Builder.

For example, the following steps were completed to create an integration-related Oracle Workflow business process:

1. Create a process bundle in iStudio and create business processes with some activities.
2. Deploy to the `my_process_bundle.wft` file.
3. Import the file into Oracle Workflow Builder.
4. Make **non-event** related modifications to the process in Oracle Workflow Builder, such as adding other activities like notifications and decision functions to complete the business process.
5. Save the modified process to `my_process_bundle.wft`.

At this point, you need to make some event-related modifications to the business process. For example, add two new events to the business process with the following steps:

6. Using iStudio, make the additions to the particular business process.

7. Deploy to a different file such as `changes_to_my_process_bundle.wft`. Do not deploy to `my_process_bundle.wft` because any non-event-related modifications made through Oracle Workflow Builder will be lost.
8. Launch Oracle Workflow Builder and import both `my_process_bundle.wft` and `changes_to_my_process_bundle.wft`.
9. Drag the required modifications from the process representing `changes_to_my_process_bundle.wft` to the process representing `my_process_bundle.wft`.
10. Save the modified process to `my_process_bundle.wft`.

The `my_process_bundle.wft` file now contains the updated process definition with both the event and the non-event modifications.

Runtime System Concepts and Components

This chapter describes the runtime concepts, components, processes, and component configuration of OracleAS InterConnect in the following topics:

- [Integration Architecture](#)
- [Features](#)
- [Components](#)
- [Real Application Clusters Configuration](#)

Integration Architecture

OracleAS InterConnect runtime is an event-based distributed messaging system. An event is any action that initiates communication through messaging between two or more applications integrated through OracleAS InterConnect. The messaging system can be deployed both within an enterprise or across enterprise boundaries.

The runtime enables inter-application communication through hub and spoke integration. This methodology keeps the applications decoupled from each other by integrating them to a central hub rather than to each other directly. The applications are at the spokes of this arrangement and are unaware of the applications with which they are integrating with. To them, the target of a message (or the source) is the hub. Since each application integrates with the hub, translation of data between the application and hub (in either direction) is sufficient to integrate two or more applications.

Features

The OracleAS InterConnect runtime features are as follows:

- [Messaging Paradigms](#)
- [Message Delivery](#)
- [Message Retention](#)
- [Routing Support](#)
- [Error Management](#)
- [Scalability and Load Balancing](#)
- [File Persistence](#)

Messaging Paradigms

OracleAS InterConnect runtime supports three major messaging paradigms:

- Publish/Subscribe
- Request/Reply (synchronous and asynchronous)
- Point-to-Point

These paradigms can be configured on a per integration point basis.

See Also: [Chapter 1, "Getting Started with OracleAS InterConnect"](#)

Message Delivery

The following are features of message delivery:

- **Guaranteed Delivery**—All messages are guaranteed to be delivered from the source application(s) to the destination application(s).
- **Exactly Once Delivery**—The messages are neither lost nor duplicated. The destination application(s) will receive each sent message exactly once.
- **In Order Delivery**—The messages are delivered in the exact same order as they were sent. This is applicable only when there is one instance of the adapter running per application serviced.

Message Retention

Messages remain in the runtime system until they are delivered. Advanced Queues in the hub provide the message retention. The messages are deleted when each application that is scheduled to receive a specific message has received that message. For auditing purposes, you can configure the system to retain all messages even after they have been delivered successfully to each application.

Routing Support

The current version of OracleAS InterConnect has significant improvements over the previous releases for configuring your routing needs. Routing is a function of the Advanced Queues in the hub. By default, `oai_hub_queue` is the only multiconsumer Advanced Queue configured to be the persistent store for all messages for all applications. This will handle all your standard as well as content based routing needs. Moreover, this queue is created automatically when you install the repository in the hub. The only reason to change this configuration is if Advanced Queues becomes a performance bottleneck. For most scenarios, this is unlikely because most of the message processing is done in the adapters, not in the hub.

See Also: ["Scalability and Load Balancing"](#) on page 8-5

Content-Based Routing

Content-based routing allows you to route messages to specific destination applications based on message content. For example, an electronic funds transaction settlement application is designed to transmit bank transactions with a specific bank code to identify the destination bank system. When the electronic funds transfer application publishes each message at runtime, the Oracle Application InterConnect runtime component determines the bank code value based on objects stored in the repository, and routes the message to the appropriate recipient system.

Error Management

This release has significant improvements to deal with error conditions in your integration environment:

- **Central Logging**—If there is an error in the system, it is logged centrally in the repository.
- **Central Monitoring**—These errors can be monitored through the runtime management console.

Resubmission

You can resubmit errored-out messages again into your integration environment for processing after modifying them (if required) using the runtime management console.

Tracing

You can modify the `.ini` files of adapters to turn up the tracing level to troubleshoot errors. You can view the tracing logs by opening up log files through the runtime management console.

Tracking

Messages can be tracked by specifying tracking fields using iStudio. The runtime system checkpoints state at certain pre-defined points so that you can monitor where the message is currently in the integration environment. This tracking capability can be utilized through the runtime management console.

Scalability and Load Balancing

At runtime, it is possible that the adapter attached to a particular application becomes a performance bottleneck. You can detect this by monitoring the message throughput information through the runtime console.

See Also: [Chapter 9, "Runtime Management"](#)

OracleAS InterConnect addresses adapter scalability through a well-defined methodology.

Multiple adapters can be attached to one application to share the message load. This can be done in several ways depending upon the needs of your integration environment. For example, Application A publishes three different kinds of events—EventA, EventB, and EventC. Three potential scenarios should be examined to determine how one or more adapters should be attached to the application to meet performance objectives:

Scenario 1

- Requirement

The order in which the messages are sent by Application A must be adhered to, strictly, for the life of all messages. For example, if Application A publishes messages in a specific order, they must be received by the subscribing applications in the exact same order across all the different event types.

- Solution

In this case, you cannot add more than one adapter to Application A for load balancing.

Scenario 2

- Requirement

The order in which messages are sent by Application A must be adhered to but not across different event types. For example, Application A publishes the following messages in order: M1_EventA, M2_EventB, M3_EventA. M1 and M3 must be ordered with respect to each other because they correspond to the same event type. However, M2 has no ordering restrictions with respect to M1 and M3. In addition, EventA messages are transformation, size, and computation heavy and EventB and EventC messages are very light.

- Solution

In this case, you can create message partitions in the Message Capability Matrix. `Partition1` can process `EventA` messages, and `Partition2` can process `EventB` and `EventC` messages. When you install the adapters, you specify not only the application it is attached to but also the partition it uses. These message partitions can be used to effectively load balance message processing.

See Also: [Chapter 1, "Getting Started with OracleAS InterConnect"](#)

Scenario 3

- Requirement

There is no message order dependency, even within the same event type.

- Solution

Since there are no ordering restrictions, two approaches for load balancing can be employed:

- No message partitions are created. One or more adapters are added utilizing the entire Message Capability Matrix. This means that at runtime any one of the adapters would be available to receive any message, though only one of them would actually receive the message. Which adapter receives the message will be determined solely by which adapter is the first to request the next message for processing.
- Message Partitions can be created based on projections of the number of messages for a particular event type. For example, if there will be three times as many `EventA` messages than `EventB` or `EventC` messages, you could create two partitions—one for handling `EventA` messages, and the other for handling the other two event types.

File Persistence

Adapters deliver messages from applications to the integration hub and vice versa. By default, adapters store messages in local file persistence during processing. This makes adapters stateful. If your customized scenario requires OracleAS InterConnect to be deployed in a fail safe environment, stateful adapters become cumbersome to manage. If an adapter fails in the middle of processing a message, a new adapter instance that takes over the failed adapter must somehow read the message from the failed adapter's local file persistence.

To overcome this problem, OracleAS InterConnect provides a feature to make adapters stateless for fail safe environments by turning off file persistence through parameters in the `adapter.ini` file.

Note: There are two parameters associated with file persistence, `agent_pipeline_to_hub` and `agent_pipeline_from_hub`. You should set both parameters to `false` to make adapters stateless.

[Table 8–1](#) Summarizes file persistence parameters.

Table 8–1 *File Persistence Parameters*

Parameter	Description
<code>agent_pipeline_to_hub=false</code>	Turn off local file persistence when sending messages from application to the hub.
<code>agent_pipeline_from_hub=false</code>	Turn off local file persistence when sending messages from the hub to the application.

Components

There are six major components in the runtime system:

- [Adapters](#)
- [Repository](#)
- [Advanced Queues](#)
- [Oracle Workflow](#)

Adapters

Prepackaged adapters help re-purpose applications at runtime to participate in the integration without any programming effort.

Agent and Bridge Combination

Adapters are the run-time component for OracleAS InterConnect. Adapters have the following responsibilities:

- **Application Connectivity**—Connect to applications to transfer data between the application and OracleAS InterConnect. The logical sub-component within an adapter that handles this responsibility is called a bridge. This is the protocol/application-specific piece of the adapter that knows how to communicate with the application. For example, the database adapter is capable of connecting to an Oracle database using JDBC and calling SQL APIs. This sub component does not know which APIs to call, only how to call them.
- **Transformations**—Transform data from the application view to common view and vice versa as dictated by the repository metadata. In general, adapters are responsible for carrying out all the runtime instructions captured through iStudio as metadata in the repository. Transformations are an important subset of these instructions. The logical sub component within an adapter that handles the runtime instructions is called an agent. This is the generic runtime engine in the adapter that is independent of the application to which the adapter connects. It focuses on the integration scenario based on the integration metadata in the repository. There is no integration logic coded into the adapter itself. All integration logic is stored in the repository. The repository contains the metadata that drives this sub component. In the database adapter example, this is the sub-component that knows which SQL APIs to call, but not how to call them. All adapters have the same agent code. It is the difference in metadata that each adapter receives from the repository that controls and differentiates the behavior of each adapter.

Adapters can be configured to cache the metadata at runtime to address performance needs. There are three settings for caching metadata:

- **No Caching**—For each message, the adapter will query the repository for metadata. This setting is recommended for an early or unstable integration development environment.
- **Demand Caching**—The adapter will query the repository only once for each message type and then cache that information. For subsequent messages of the same type, it will use the information from the cache. This setting is recommended for a stable integration development environment.
- **Full Caching**—At start-up time, the adapter will cache all its relevant metadata. This setting is recommended for a production environment.

The following adapters are available with OracleAS InterConnect:

- **OracleAS InterConnect Adapter for DB**—Allows you to communicate to an Oracle database using JDBC.

- OracleAS InterConnect Adapter for AQ—Allows you to communicate to an Advanced Queue with Raw, XML, and ADT payloads.
- OracleAS InterConnect Adapter for HTTP—Allows you to communicate across firewalls with an HTTP or HTTPS server using XML payloads.
- OracleAS InterConnect Adapter for FTP—Enables an Oracle FTP Application to be integrated with other applications using OracleAS InterConnect. This adapter is useful in all enterprise application integration scenarios involving the FTP transport protocol or local file system. The Advanced Queuing adapter can monitor incoming messages which are in the form of FTP files placed in a remote FTP server or on local file systems. The Advanced Queuing adapter is also capable of sending messages to remote FTP servers via proxy host. The payload for this adapter can be XML data or D3L data.
- OracleAS InterConnect Adapter for MQ Series—Enables OracleAS InterConnect to send messages to and receive messages from the MQ Series queues and topics. This allows an application that uses IBM's MQ Series as its messaging technology to be integrated with other applications using OracleAS InterConnect. Therefore, the Advanced Queuing adapter is useful in all enterprise application integration scenarios involving MQ Series-based applications.
- OracleAS InterConnect Adapter for SMTP—The SMTP adapter enables an SMTP application to be integrated with other applications using OracleAS InterConnect. This adapter is useful in all enterprise application integration (EAI) environments where e-mail uses the Internet Message Access Protocol 4 (IMAP4) and SMTP transport protocols.
- OracleAS InterConnect Adapter for PeopleSoft 7.5x
- OracleAS InterConnect Adapter for PeopleSoft 8
- OracleAS InterConnect Adapter for SAP R/3
- OracleAS InterConnect Adapter for CICS
- OracleAS InterConnect Adapter for Siebel 2000
- OracleAS InterConnect Adapter for Siebel 7
- OracleAS InterConnect Adapter for J.D. Edwards OneWorld XE

See Also: Adapter-specific installation and user's guides for details on respective OracleAS InterConnect Adapters

Repository

The repository consists of two components:

- **Repository Server**—A Java application that runs outside the database. It provides CORBA services to create/modify/delete metadata at design time using iStudio and query during runtime using adapters. Both adapters and iStudio act as CORBA clients to communicate with the repository server.
- **Repository Database**—The repository server stores metadata in database tables. The server communicates to the database using JDBC.

Adapters have the ability to cache metadata. If the repository metadata is modified after adapters have cached metadata, those adapters are automatically notified by the repository server to purge their caches and re-query the new metadata.

Advanced Queues

Advanced Queues provide the messaging backbone for OracleAS InterConnect in the hub. In addition to being the store and forward unit, they provide message retention, auditing, tracking, and guaranteed delivery of messages.

See Also: *Oracle Database Application Developer's Guide* for information on Advanced Queues

Oracle Workflow

Oracle Workflow facilitates integration at the business process level through its Business Event System. OracleAS InterConnect and Oracle Workflow are integrated to leverage this facility for business process collaborations across applications.

Real Application Clusters Configuration

Introduction

Real Application Clusters (RAC) harnesses the processing power of multiple interconnected computers, unites the processing power of each component to create a robust computing environment. In RAC environment, all active instances can concurrently execute transactions against a shared database. RAC coordinates each instance's access to the shared data to provide data consistency and data integrity. It features balance of workloads among the nodes by controlling multiple server connections during period of heavy use and provide persistent, fault tolerant connections between clients and RAC database.

See Also: The following documentation for additional information on RAC:

- *Oracle9i Real Application Clusters Administration*
- *Oracle Application Server 10g Concepts*

OracleAS InterConnect Adapters Supporting RAC

OracleAS InterConnect adapters leverage RAC technology, provide consistent and uninterrupted services without having to restart the adapters, if an instance fails, and provide guaranteed message delivery. OracleAS InterConnect adapters connect to the first of the listed available nodes in `adapter.ini` and `hub.ini` files.

See Also: OracleAS InterConnect adapters installation documentation for details on `adapter.ini` and `hub.ini` files associated with specific adapters

If one of the nodes fails, database connection is re-established to the next available node in the `adapter.ini` or `hub.ini` file list mentioned above, recursively, until a successful connection is made transparent to the user.

The hub connections for all adapters are RAC enabled. The spoke connections for DB and AQ adapters are RAC enabled.

See Also: Section Support for Oracle Real Application Clusters in the *Oracle9i Application Developer's Guide—Advanced Queuing*

Configuration

The `adapter.ini` and `hub.ini` files must be populated with the information about the host, port, and instance for all the nodes. Additional sets of parameters which specifies the number of nodes are also required to be populated. All the existing entries remain the same except a new entry for each node is added. [Table 8–2](#) describes the additional sets of parameters which specify the number of nodes required to be populated.

Table 8–2 Additional Parameters for RAC Configuration

File Name	Parameter
hub.ini	host_num_nodes hub_hostx hub_portx hub_instancex—where x varies between 2 and the number of nodes.
adapter.ini for the Advanced Queueing adapter	ab_bridge_num_nodes aq_bridge_host aq_bridge_port aq_bridge_instance
adapter.ini for the Database adapter	db_bridge_num_nodes db_bridge_schemal_hostx db_bridge_schemal_portx db_bridge_schemal_instancex—where x varies between 2 and the number of nodes.

Sample hub.ini File

The following are sample hub.ini files.

```

hub_username=oaihub904
hub_password=oaihub904
hub_use_thin_jdbc=true
hub_host=dlsun1312
hub_instance=iasdb
hub_port=1521
hub_num_nodes=2
hub_host2=vindaloo
hub_instance2=orcl
hub_port2=1521

```

Sample Database Adapter adapter.ini File that Shows the Spoke Database Entry

The following are sample `adapter.ini` files for the Database adapter that shows the spoke database entry.

```
db_bridge_schema1_host=dlsun1312
db_bridge_schema1_port=1521
db_bridge_schema1_instance=iasdb
db_bridge_num_nodes=2
db_bridge_schema1_host2=vindaloo
db_bridge_schema1_port2=1521
db_bridge_schema1_instance2=orcl
```

Runtime Management

This chapter describes how to manage OracleAS InterConnect components using the Oracle Enterprise Manager Central Console.

This chapter discusses the following topics:

- [Introduction to Runtime Management](#)
- [Features](#)

Introduction to Runtime Management

The Runtime Management Console is an Oracle Enterprise Manager Console-based tool that allows you to manage your integration components at runtime. The OracleAS InterConnect components that can be managed through the Console are as follows:

- Adapters
- Repository Server

To be able to manage your integration components, you must

- Install the Oracle Enterprise Manager Server on your hub machine. When the management server is started it detects the repository in the hub and all associated adapters.
- Install the Oracle Enterprise Manager Console on the machine you wish to manage your integration components from.

See Also:

- *Oracle Application Server Installation Guide*
- *Oracle Enterprise Manager Concepts Guide*, Release (9.2.0)

Starting Oracle Enterprise Manager Console

To start the Oracle Enterprise Manager Console:

1. Start the Oracle Enterprise Manager Console with the following command:

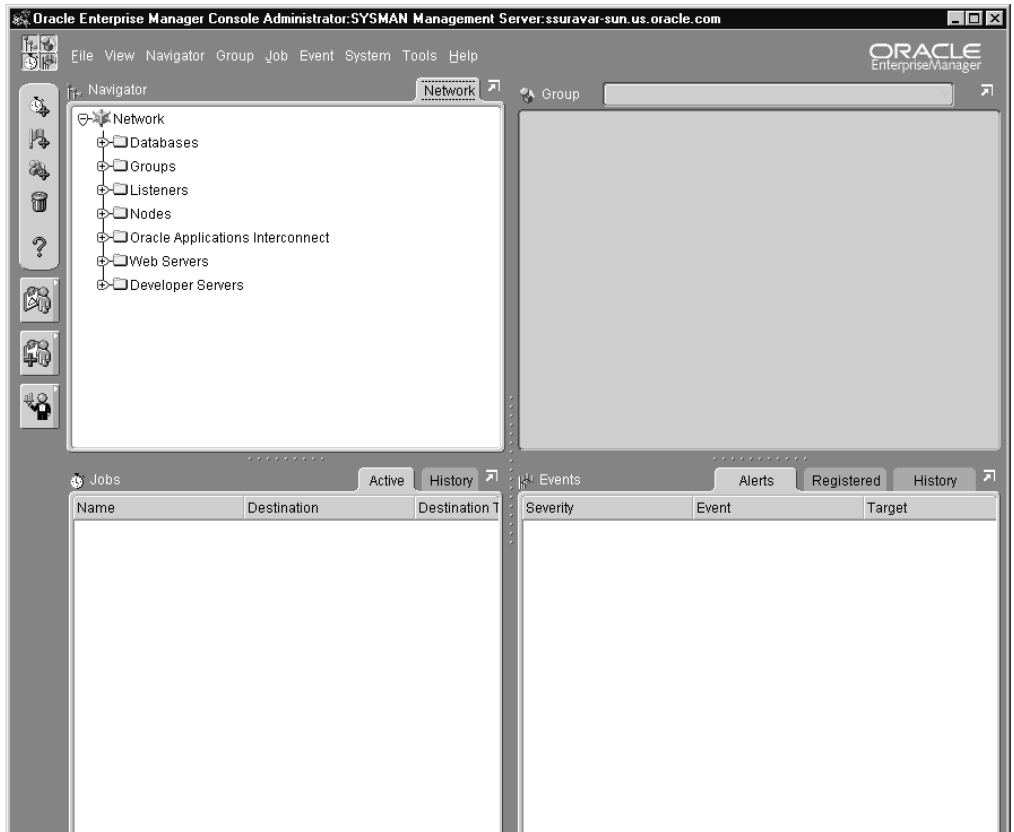
(On UNIX) `$ oemapp console`

(On Windows) `c:\oemapp console`

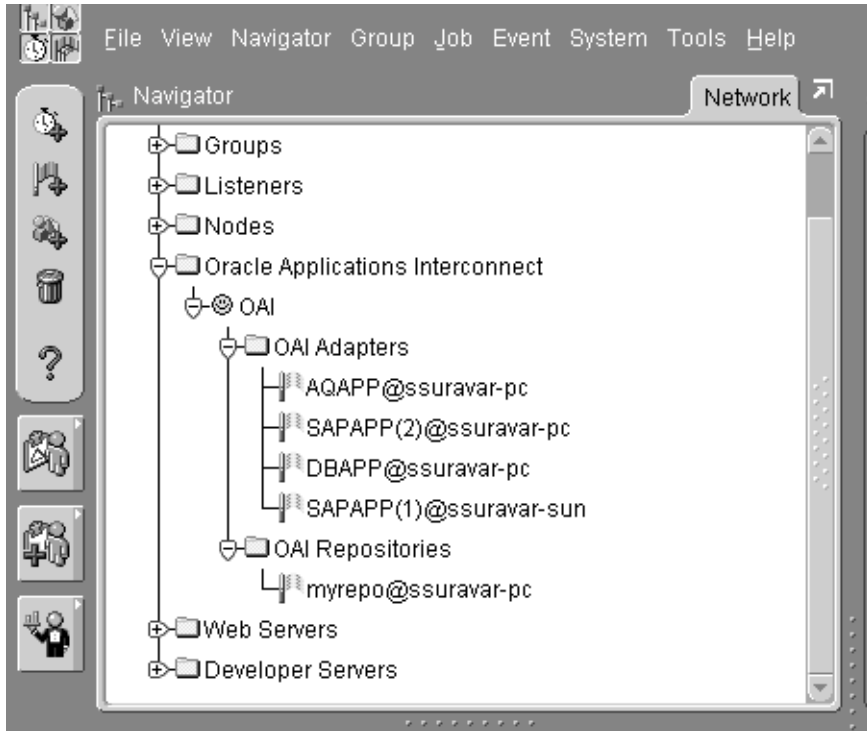
You may also start the Oracle Enterprise Manager Console from the Windows Services panel.

2. On the Oracle Enterprise Manager Login page, enter the Administrator name, password, and the Management Server that you want to connect to. Click **OK**.

3. The Console displays.



- Expand the OracleAS InterConnect subtree, a list of adapters and the repository associated with the particular OracleAS InterConnect hub displays.



- Right click any of the adapters or repositories to manage or to use its features.

Features

The following are runtime features for adapters and the repository.

Common Features for Adapters and the Repository

The following features are provided for both adapters and repository:

- **Get State**—Shows the component's status. Red indicates the component is stopped. Green indicates the component is running. Yellow indicates the component state is unknown.
- **Start**—Starts the component.

- Stop—Stops the component.
- Alert On Shutdown—Allows you to send a page or email if that particular component goes down. Refer to the *Oracle Enterprise Manager Administrator's Guide* for setting up the alert.
- Get Ini File—Allows you to browse and modify the `.ini` file for that particular component. If you make changes, the tool prompts you to restart the component. You must restart the component for the changes to take effect.
- Get Log File—Allows you to browse the log files for a particular component. Repositories have one log file each. Adapters have multiple log files. This feature allows you to select the log file you want to view.
- Get Alive Time—Gets the duration that the component has been up for since it was last started.
- Remove—Removes the component from the list of manageable components for this session of the Oracle Enterprise Manager Console.

Repository Specific Features

- Get Repo Sessions—Gets a list of all components connected to the repository, such as adapters and iStudio.

Adapter Specific Features

- Get Throughput—Monitor the adapter throughput in terms of messages/second. The steps to get throughput information are as follows:
 1. Right-click the adapter for which to get throughput information and select Get Throughput.
 2. On the dialog that displays, click Get Throughput. Throughput information is displayed for inbound and outbound message separately. At startup time the adapter starts recording throughput information by default. To record throughput information in a particular interval, click disable timer, then click enable timer. This restarts the recording process and allows new throughput information to be available by clicking Get Throughput.
- Track Messages—Track particular messages at runtime. During design time, in iStudio, you can specify fields in the application view for publish event or invoke procedure with which to track the message. For example, a customer related message could be tracked by an application view field, such as social

security number. The following steps utilize this functionality in the runtime management console:

1. Right-click on any adapter attached to the application that is publishing the event or invoking the procedure to track. On the context menu, select Track Messages. A list of trackable events and procedures is displayed.
 2. Select the event or procedure to track.
 3. A dialog displays asking for the value(s) of the selected tracking field(s) for that particular event or procedure in iStudio. Therefore, in the customer related message example, the dialog asks for the social security number.
- Get Errors—Displays all errors associated with the adapter. Certain errors have a message payload associated with them. These errors are marked with a yes in the Editable column. To modify and resubmit a message:
 1. Double-click the description field for such an error. Depending on the processing state at which the error occurred, a dialog displays an XML message in the application or common view.
 2. If necessary, edit the message. If this is not necessary, go to the next step.
 3. Click Resubmit. The message now re-enters the integration system for reprocessing.

Integration Scenario

This appendix provides an integration scenario and model based on a fictitious company, Acme, Inc. using OracleAS InterConnect. This appendix discusses the following topics:

- [Integration Scenario Overview](#)
- [Modeling the Integration](#)
- [Implementing the Scenario](#)
- [Modeling Business Logic in Oracle Workflow](#)
- [Deployment](#)
- [Conclusion](#)

Integration Scenario Overview

Each division of Acme, Inc. has multiple Order Fulfillment Systems which are a legacy from various mergers and acquisitions. Maintaining the parts of these systems such as platforms, software, training, etc. is costly and time consuming for Acme. In addition, the lack of integration between the systems prevents business analysis on the enterprise level.

Acme has created a new centralized system and the first phase of the integration project is to synchronize one of the legacy systems with the new system. The first test is to synchronize the Purchase Order information, where all purchase orders from the legacy system are to be reflected in the new system.

The New Centralized System

This new order fulfillment system operates on an Oracle9i database and uses the OracleAS InterConnect Database Adapter to communicate with this system.

The Legacy System

The legacy order fulfillment system operates on an Oracle8i database and uses the OracleAS InterConnect Advanced Queueing Adapter to communicate with this system.

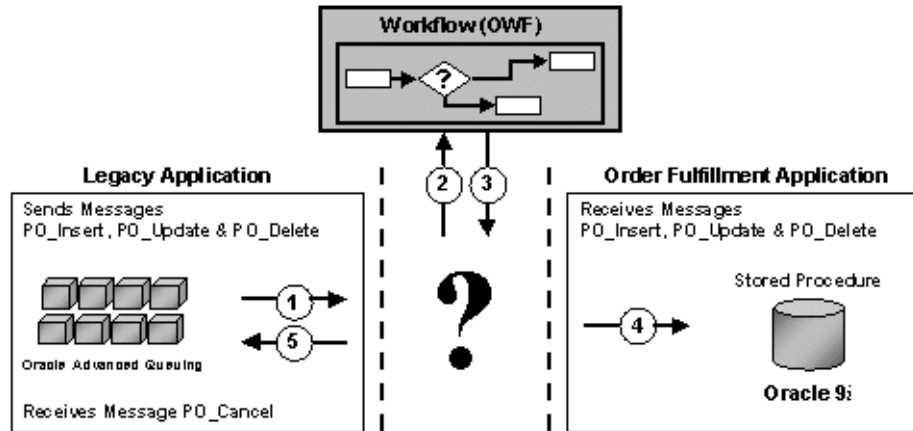
On the Purchase Order table in this system a database trigger queues the changed records. OracleAS InterConnect is configured to listen to that queue to accomplish this integration.

Note: There are many methods available to capture changes to in a system. These methods include but are not limited to database triggers, a batch process using interface tables, and the use of database log files.

The Integration Scenario

Figure A-1 illustrates the integration scenario:

Figure A-1 Integration Scenario



The question mark above is the task on hand:

How can we accomplish this task?

The first step in any integration scenario is to model the integration.

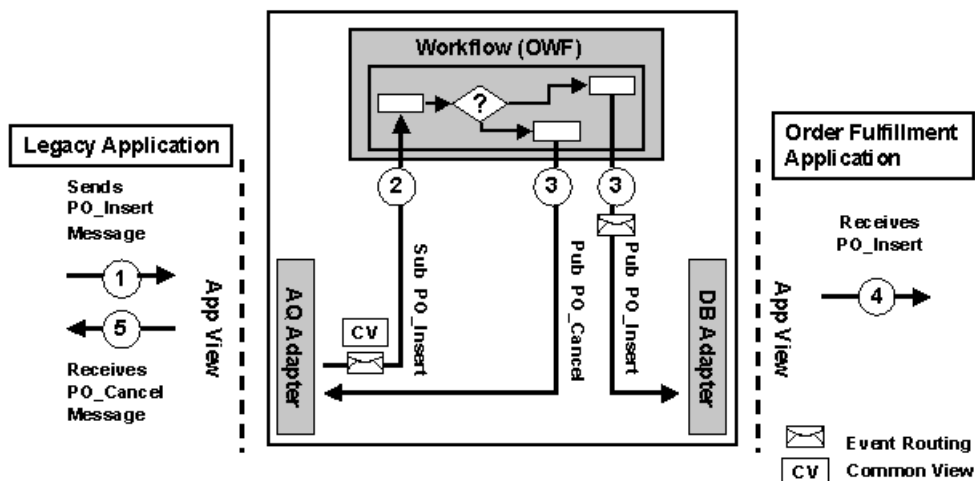
- Legacy System
 - Any changes to the Purchase Order table in the Legacy application are published via a database trigger. An administrator must approve all changes such as insert, update, & delete before they are applied to the Order Fulfillment System
- Order Fulfillment System
 - If a change is approved, it is sent to the Order Fulfillment System. If a change is rejected, then a cancellation notification is sent back the Legacy system.
- Additional Issues
 - The process must be non intrusive, the user cannot alter the structure of either system.

- Synchronization of the primary keys of each system must be maintained by the integration platform.
- The integration must be scalable and support addition of systems.

Modeling the Integration

Figure A-2 illustrates how OracleAS InterConnect integrates with the scenario in Figure A-1.

Figure A-2 Integration Modeling



Now that the integration scenario has been defined:

How are we going to accomplish this task?

1. The Legacy Application publishes the insert, update, and delete messages onto a queue. To receive or send messages onto that queue, the OracleAS InterConnect Adapter for AQ (Advanced Queueing adapter) is used.

The Order Fulfillment Application is a standard Oracle database and uses the OracleAS InterConnect Adapter for DB (Database adapter).

2. All messages are routed to Oracle Workflow to apply user-defined logic.

Integration Modeling using iStudio

The following list of steps outlines the process to accomplish this integration.

1. Review Legacy System Database Trigger
2. Create a Project
3. Create the Common View Business Object
4. Create Business Object Events
5. Create Applications
6. Create a Cross Reference Table
7. Create Publish Events
8. Subscribing to Events
9. Create Content Based Routing
10. Create an Oracle Workflow Process Bundle
11. Deploy the Process Bundle to Oracle Workflow
12. Creating Objects in Oracle Workflow for Modeling
13. Deployment
14. Creating Objects in Oracle Workflow for Modeling
15. Modeling Business Logic in Oracle Workflow
16. Deployment
 - Setting Queues
 - Pushing Metadata
 - Exporting and Installing Code

Implementing the Scenario

The following sections describe implementing the integration scenario.

Review Legacy System Database Trigger

The source system uses Oracle8i Advanced Queuing to publish changes to the purchase order table. To be non-intrusive, the user creates a database trigger on the purchase order table. When a record is updated, inserted, or deleted and then committed, the trigger enqueues the appropriate payload. The OracleAS InterConnect Advanced Queuing Adapter is configured to listen on this queue.

The following code is for the database trigger.

```
CREATE OR REPLACE TRIGGER AQAPP.ENQUEUE_PO
    AFTER INSERT OR DELETE OR UPDATE ON AQAPP.PURCHASE_ORDER FOR EACH ROW
DECLARE
    qname                VARCHAR2(20)        := 'OUTBOUND_QUEUE';
    enqueue_options      DBMS_AQ.ENQUEUE_OPTIONS_T;
    message_properties   DBMS_AQ.MESSAGE_PROPERTIES_T;
    msgid                RAW(16);
    recip_agent          SYS.AQ$_AGENT;
    raw_payload          RAW(32767);
    payload              VARCHAR2(256);
BEGIN
    IF INSERTING THEN
        payload := '<?xml version="1.0" standalone="no"?>' ||
            '<PO_Insert>' ||
            '<id>' || :new.id || '</id>' ||
            '<action>' || 'I' || '</action>' ||
            '<item>' || :new.item || '</item>' ||
            '<amount>' || :new.amount || '</amount>' ||
            '<quantity>' || :new.quantity || '</quantity>' || '</PO_Insert>';

    ELSIF DELETING THEN
        payload := '<?xml version="1.0" standalone="no"?>' ||
            '<PO_Delete>' ||
            '<id>' || :old.id || '</id>' ||
            '<action>' || 'D' || '</action>' || '</PO_Delete>';

    ELSIF UPDATING THEN
        payload := '<?xml version="1.0" standalone="no"?>' ||
            '<PO_Update>' ||
            '<id>' || :old.id || '</id>' ||
            '<action>' || 'U' || '</action>' ||
            '<item>' || :new.item || '</item>' ||
            '<amount>' || :new.amount || '</amount>' ||
            '<quantity>' || :new.quantity || '</quantity>' ||
            '<last_updated>' || :new.last_updated || '</last_updated>' || '</PO_Update>';
```

```

END IF;

raw_payload := UTL_RAW.CAST_TO_RAW( payload );

DBMS_AQ.ENQUEUE( queue_name      => qname
                 ,enqueue_options => enqueue_options
                 ,message_properties => message_properties
                 ,payload          => raw_payload
                 ,msgid           => msgid );

EXCEPTION
    WHEN OTHERS THEN NULL;

END;

```

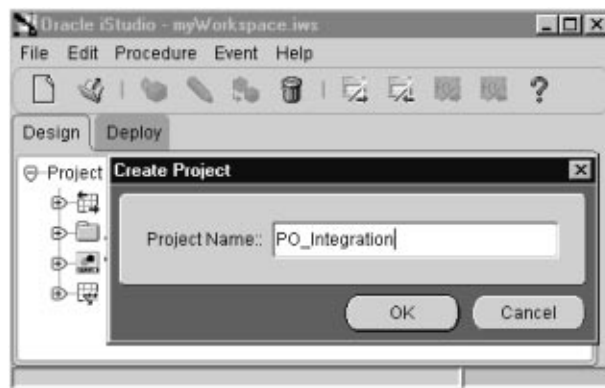
Create a Project

A project is a container for the integration logic pertaining to an integration scenario. The following steps describe creating the PO_Integration project using iStudio.

1. From the File menu, select New Project. The Create Project dialog displays.
2. Enter PO_Integration in the Project Name field and click OK. The Repository Information dialog displays.
3. Enter the correct repository information and click OK.

See Also: ["Creating a New Project"](#) on page 2-15

Figure A-3 *Creating a Project*



Create the Common View Business Object

Each application has its own semantics and syntax. In order to integrate the data from multiple sources, a common view that is semantically compatible is required. The common views are either events or procedures and are grouped in a business object, located under the Common Views node in iStudio. In this scenario, all events are grouped under the `Purchase_Order` business object.

The following steps describe creating the `Purchase_Order` business object.

1. From the File menu select New, then select Business Object. The Create Business Object dialog displays.
2. Enter `Purchase_Order` in the Business Object Name field and click OK.

See Also: ["Creating Business Objects"](#) on page 3-3

Create Business Object Events

In order to integrate data between two or more systems, a semantically compatible view, or common view, is required. In this scenario, the insert, updated, delete, and cancel events are grouped under the `Purchase_Order` business object. The following four events must be created:

- `PO_Cancel`
- `PO_Insert`
- `PO_Update`
- `PO_Delete`

Note: When an event is created, a Common Data Type representing its structure is automatically created. This common data type can then be reused to define the structure of other events.

The following steps describe creating the `PO_Insert` event using an XML DTD (eXtensible Markup Language Data Type Definition). The user can also use the database or other common data type to describe the structure of the event.

1. From the File menu, click New, then select Event. The Create Event dialog displays.
2. Select `Purchase_Order` as the Business Object.
3. Enter `PO_Insert` in the Event Name field.
4. Click Import and select XML.
5. Select the predefined file, `PO_Insert_CV.dtd` in the Open dialog and click Open.
6. Select `PO_Insert` in the Choose Root Dialog and click OK to return to the Create Event dialog.
7. Click OK.

See Also: ["Creating Events"](#) on page 4-3

Use the same steps for the `PO_Update`, `PO_Delete`, and `PO_Cancel` events, substituting the following correct XML DTD for each event. The `PO_Cancel`, `PO_Delete`, `PO_Insert`, and `PO_Update` events appear in the Design Object Navigator under the Events node as shown in [Figure A-4](#).

DTD Code

Each event has its own XML DTD. The following code is listed for each event.

- `PO_Cancel`

```
<!ELEMENT PO_Cancel (id, action, item, amount, quantity)>
<!ELEMENT id          (#PCDATA)>
<!ELEMENT action      (#PCDATA)>
<!ELEMENT item        (#PCDATA)>
<!ELEMENT amount      (#PCDATA)>
<!ELEMENT quantity    (#PCDATA)>
```

- `PO_Update`

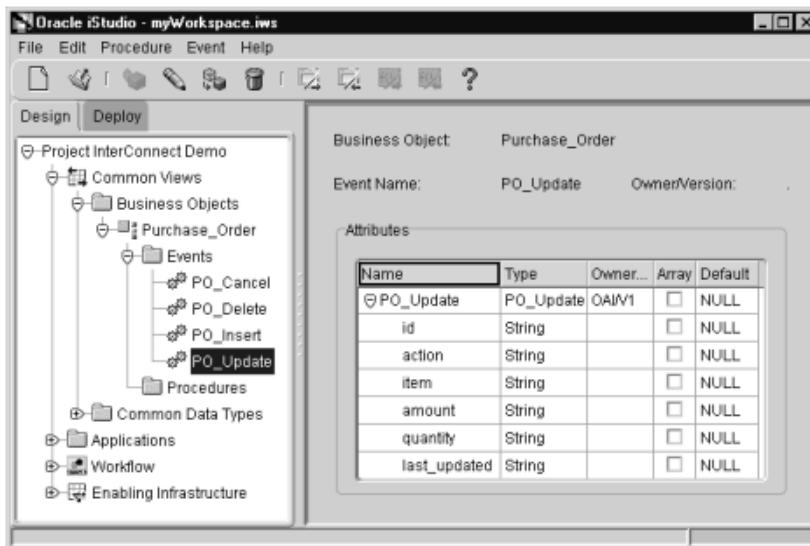
```
<!ELEMENT PO_Update (id, action, item, amount, quantity, last_updated)>
<!ELEMENT id          (#PCDATA)>
<!ELEMENT action      (#PCDATA)>
<!ELEMENT item        (#PCDATA)>
<!ELEMENT amount      (#PCDATA)>
<!ELEMENT quantity    (#PCDATA)>
<!ELEMENT last_updated (#PCDATA)>
```

- PO_Delete


```
<!ELEMENT PO_Delete (id, action)>
<!ELEMENT id          (#PCDATA)>
<!ELEMENT action      (#PCDATA)>
```
- PO_Insert


```
<!ELEMENT PO_Insert (id, action, item, amount, quantity)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
```

Figure A-4 Completed Event Node in iStudio



Note: The Business Object and Events display in the Object Navigator under node Common Views as shown in [Figure A-4](#).

Create Applications

An application in iStudio represents an instance of an adapter communicating with an application. When the user installs an adapter, a unique name is supplied and, in iStudio, this name is used as the name of the application. This scenario demonstrates creating the AQAPP and DBAPP applications.

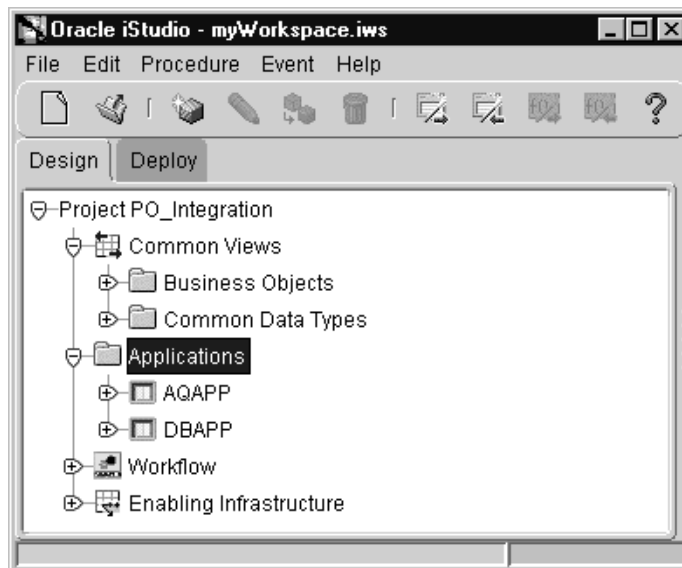
See Also: ["Creating an Application"](#) on page 3-2

The following steps describe creating the AQAPP application using iStudio.

1. From the File menu, select New, then select Application. The Create Application dialog displays.
2. Enter AQAPP in the Application Name field and click OK.

Complete the same steps to create the DBAPP application. The AQAPP and DBAPP applications appear in the Design Object Navigator under the Applications node as shown in [Figure A-5](#).

Figure A-5 AQAPP and DBAPP Applications in iStudio



Create a Cross Reference Table

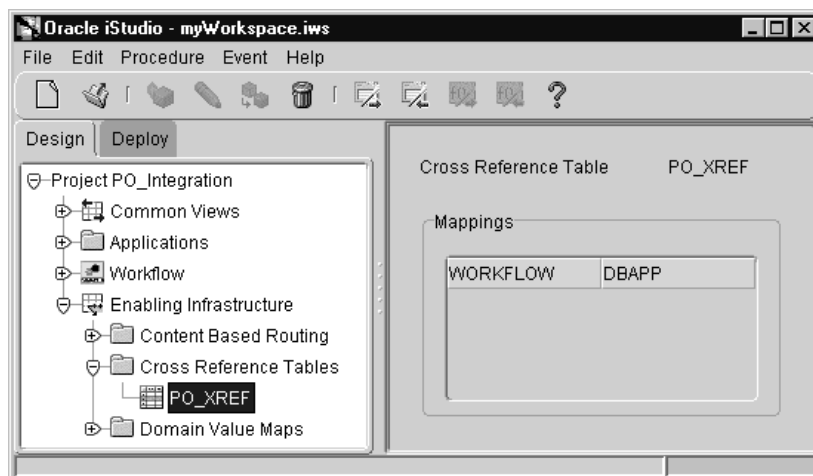
Each system has its own unique identifier or primary key. In most cases, an administrator does not allow any changes to the structure of their systems. Therefore, using a cross reference table, the keys of both systems can be maintained and cross referenced for subsequent updates and deletes.

The following steps describe creating the `PO_XREF` cross reference table using iStudio. The table is automatically created in the repository schema and is referenced by the subscribing application. The `WORKFLOW` and `DBAPP` applications are added to the table, as the publisher and subscriber respectively.

1. From the File menu, click New, then select Cross Reference Tables. The Create Cross Reference Table dialog displays.
2. Enter `PO_XREF` in the Table Name field and click OK.
3. Right click on the `PO_XREF` in the Navigator and add the `WORKFLOW` and `DBAPP` applications. The `PO_XREF` cross reference table appears in the Design Object Navigator under the Cross Reference Tables node as shown in [Figure A-6](#).

See Also: ["Creating Cross-Reference Tables"](#) on page 6-7

Figure A-6 *PO_XREF Cross Reference Table in iStudio*



Create Publish Events

The database trigger in the Legacy Application, `AQAPP`, publishes messages when records are inserted, updated, or deleted on the purchase order table. This process happens outside the OracleAS InterConnect environment. The OracleAS InterConnect Advanced Queueing adapter is configured to read these messages. The publish events under the iStudio application will:

- Map the application view to the common view.
- Perform transformations.
- Publish the new event to subscribers in the OracleAS InterConnect environment.

The following steps describe how the message received from the Legacy Application queue is processed.

See Also: ["Publishing an Event"](#) on page 4-5

Step 1 Starting the Publish Wizard

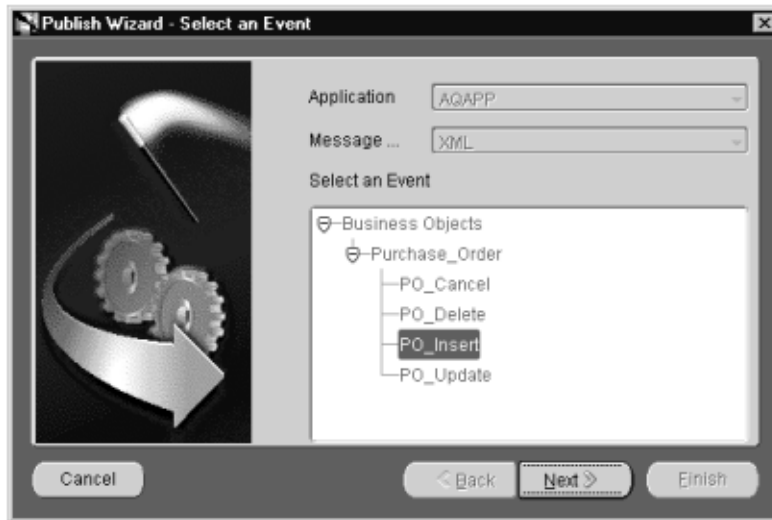
To start the Publish Wizard:

1. Expand the Applications node in the Design Object Navigator.
2. Select and expand the `AQAPP` application.
3. Select the published events node.
4. Right-click Published Events and select New. The Publish Wizard displays.

Step 2 Using the Publish Wizard to Publish the `PO_Insert` Event

When the Publish Wizard starts the following pages display.

1. Select an Event Page
 - a. Enter information in the following fields:
 - * **Application**—Select `AQAPP` for the application.
 - * **Message Type**—Select XML for the message type.
 - b. Expand the Business Objects tree in the Select an Event box and drill down to `PO_Insert`.
 - c. Select `PO_Insert` and click Next.

Figure A-7 Publish Wizard—Select an Event page

2. Define Application View Page

a. Import Attributes

Import attributes from the common view by clicking Import and select Common View. The structure of the `PO_Insert` common view event displays. If the application view is different from the common view, use the database or an XML DTD to define the structure.

b. Create an Event Map

An event is received and converted into a common view to which any application can map. If the structure of one or more events is identical, then routing becomes an issue. An event map is used to distinguish the routing in this situation. The Action field in the application view contains an *I* for insert, a *U* for update, or a *D* for delete. Complete the following steps to create an event map:

- * Click Event Map, then click Add.
- * Select the Action field and enter *I*.
- * Click Add.

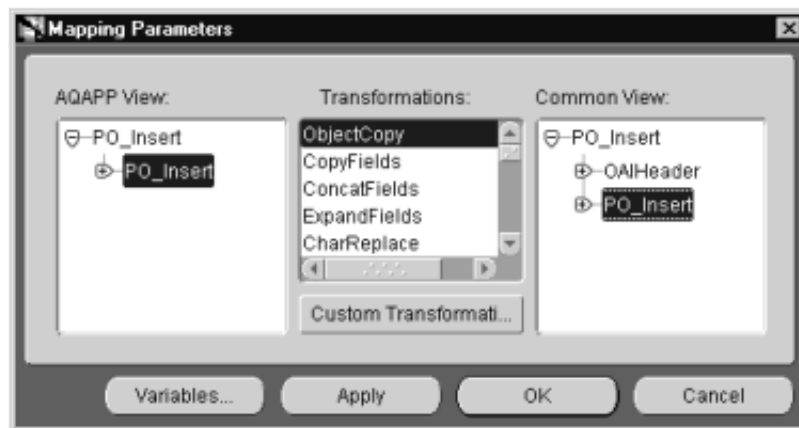
c. Click Next.

3. Define Mapping Page

Use the Define Mapping page to map fields from the AQAPP View to the common view using transformations. In this scenario the structure is identical, therefore, the `ObjectCopy` transformation is used to map all the fields at once. To define new mappings:

- a. Click New. The Mapping Parameters dialog displays.
- b. Expand the `PO_Insert` tree and select the `PO_Insert` node in the AQAPP View box.
- c. Select `ObjectCopy` in the Transformations box.
- d. Expand the `PO_Insert` tree and select the `PO_Insert` node in the Common View box.
- e. Click OK. The new mapping displays in the Summary box of the Define Mapping page.
- f. Click Finish.

Figure A-8 Publish Wizard—Mapping Parameters



To create the PO_Update and PO_Delete publish events, repeat the same steps, using the following values for steps 2 and 3.

- PO_Update
 - Use the PO_Update common view.
 - The event map value is U.
 - Use the ObjectCopy transformation and map to PO_Update.
- PO_Delete
 - Use the PO_Delete common view.
 - The event map value is D.
 - Use the ObjectCopy transformation and map to PO_Delete.

Subscribing to Events

The DBAPP application subscribes to the following three events:

- PO_Insert
- PO_Update
- PO_Delete

The AQAPP application subscribes only to the PO_Cancel event.

See Also: ["Subscribing to an Event"](#) on page 4-12

DBAPP Application Subscriptions

Step 1 Starting the Subscribe Wizard

1. In the Design Object navigator, expand the Application node.
2. Select and expand the Application node to display the Subscribed Events node.
3. Right-click Subscribed Events and select New. The Subscribe Wizard displays.

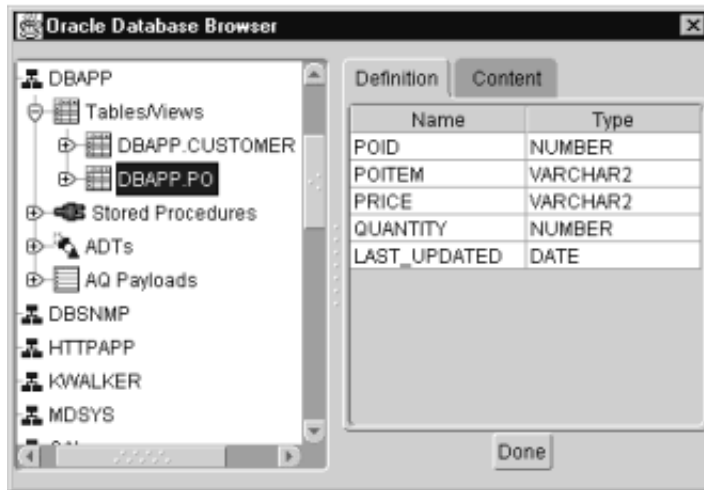
Step 2 Using the Subscribe Wizard to Subscribe to the PO_Insert Event

1. Select an Event Page
 - a. Enter information in the following fields:
 - * **Application**—Select DBAPP.
 - * **Message Type**—Select Database.
 - b. Expand the Business Objects node in the Select an Event box and drill down to PO_Insert.
 - c. Select PO_Insert and click Next.

Figure A-9 *Subscribe Wizard—Select an Event page*



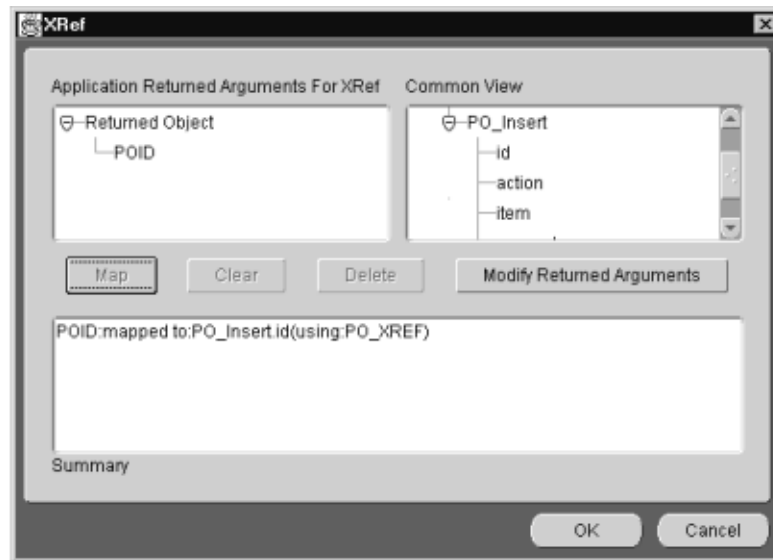
2. Define Application View Page
 - a. Import attributes from the database:
 - * Click Import and select Database. The Database Login dialog displays.
 - * Enter the correct information to login to the database and click Login. The Oracle Database Browser dialog displays.
 - * In the Browser dialog, expand the Tables/Views node and select DBAPP.PO.
 - * Click Done.

Figure A-10 *Subscribe Wizard—Oracle Database Browser*

b. Create a cross reference.

In ["Create a Cross Reference Table"](#) on page A-12, the PO_XREF cross reference table was created. This table synchronizes the primary keys on the source and target systems.

- * Click Cross Reference and select PO_XREF. The XRef dialog displays.
- * Select POID in the Application Returned Arguments For XRef box.
- * Select id in the Common View box.
- * Click Map.
- * Click OK.

Figure A-11 *Subscribe Wizard—Cross Reference*

c. Click Next.

3. Define Mapping Page

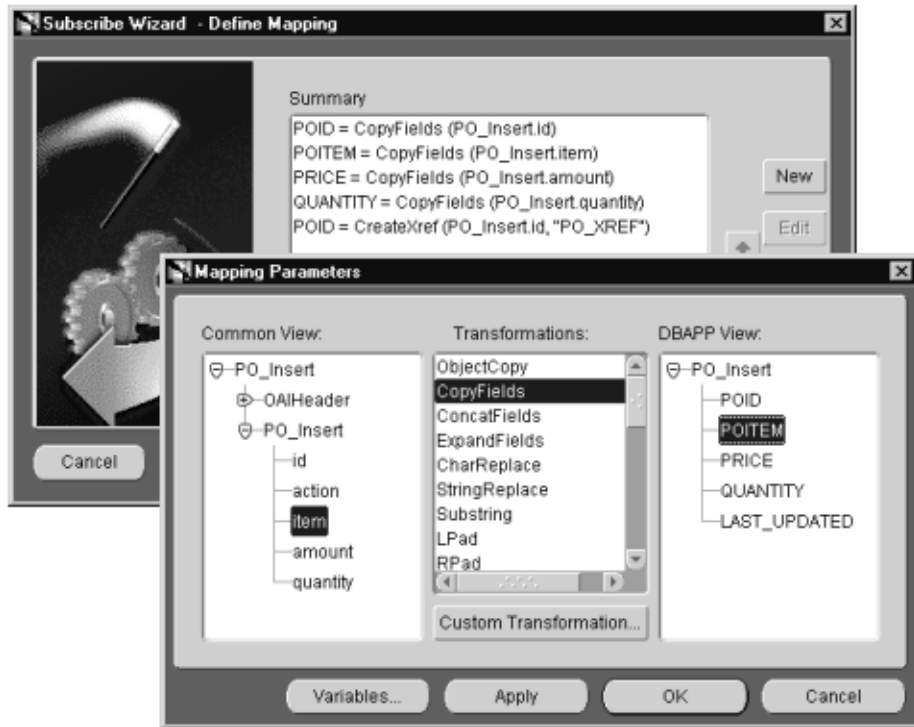
a. Define a new mapping:

- * Click New. The Mapping Parameters dialog displays.
- * Expand the `PO_Insert` tree and the `PO_Insert` node in the Common View box. Map the following:

Common View	Transformation	DBAPP View
item	CopyFields	POITEM
amount	CopyFields	PRICE
quantity	CopyFields	QUANTITY

- * Click OK.

Figure A-12 *Subscribe Wizard—Mapping Parameters*



- b. Click Next.
4. Define Stored Procedure Page
 - a. Select sub_PO_Insert_OAI_v1 from the SQL code drop down list. The SQL code displays in the box.

- b. Add the following code:

```
PROCEDURE sub_PO_Insert_OAI_V1( POID           IN OUT LONG,
                                POITEM        IN LONG,
                                PRICE         IN LONG,
                                QUANTITY      IN NUMBER,
                                LAST_UPDATED  IN DATE)

AS
  v_poid NUMBER;

BEGIN
  SELECT PO_SEQ.NEXTVAL INTO v_poid FROM dual;
  POID :=v_POID;

  INSERT INTO PO VALUES
    ( v_POID, POITEM, PRICE, QUANTITY, SYSDATE );
  COMMIT;
END sub_PO_Insert_OAI_V1;
```

- c. Click Finish.

Step 3 Create the Subscribed PO_Update Event

The wizard steps have been abbreviated:

1. Select an Event Page
Select the PO_Update event.
2. Define Application View Page
Import the Common View.

3. Define Mapping Page

- a. Map the same parameters as described in PO_Insert.
- b. In addition, map the following:
 - * Expand the PO_Update tree and node in the Common View box and select id.
 - * Select the LookupXref transformation.
 - * Expand the PO_Update tree and select POID in the Application View box.
 - * Click Apply. The Mapping dialog displays.
 - * Select the Req. checkbox for table listed in the Parameters column and click OK.
- c. Click Next.

4. Define Stored Procedure Page

- a. Select sub_PO_Update_OAI_V1 for the SQL code for field. The code displays in the box.
- b. Add the following code:

```

PROCEDURE sub_PO_Update_OAI_V1( POID           IN NUMBER,
                                POITEM        IN LONG,
                                PRICE         IN LONG,
                                QUANTITY     IN NUMBER,
                                LAST_UPDATED  IN DATE)

AS
  v_poid      NUMBER :=poid;
  v_poitem    LONG   :=poitem;
  v_price     LONG   :=price;
  v_quantity  NUMBER :=quantity;
BEGIN
  UPDATE PO SET  poitem = v_poitem, price = v_price
                quantity = v_quantity, last_updated = sysdate
  WHERE poid = v_poid;
  COMMIT;

EXCEPTION
  WHEN OTHER THENS NULL;

END sub_PO_Update_OAI_V1;
    
```

- c. Click Finish.

Step 4 Create the Subscribe PO_Delete Event

The wizard steps have been abbreviated:

1. Select an Event Page
 - Select the PO_Delete event.
2. Define Application View Page
 - Import the Common View.
3. Define Mapping Page
 - a. Map the same parameters as described in PO_Insert.
 - b. In addition, map the following:
 - * Expand the PO_Delete tree and node in the Common View box and select id.
 - * Select the DeleteXref transformation.
 - * Expand the PO_Delete tree and select POID.
 - * Click Apply. The Mapping dialog displays.
 - * Select PO_XREF from the values column and click OK.
 - c. Click Next.
4. Define Stored Procedure Page
 - a. Select sub_PO_Delete_OAI_V1 for the SQL code for field. The code displays in the box.
 - b. Add the following code:

```

PROCEDURE sub_PO_Delete_OAI_V1( POID           IN NUMBER,
                                POITEM        IN LONG,
                                PRICE         IN LONG,
                                QUANTITY      IN NUMBER,
                                LAST_UPDATED  IN DATE)

AS
  v_poid      NUMBER :=poid;
BEGIN
  DELETE FROM WHERE PO v_poid = poid;
  COMMIT;
EXCEPTION

```

```
WHEN OTHERS THEN NULL;  
  
END sub_PO_Update_OAI_V1;
```

- c. Click Finish.

AQAPP Application Subscriptions

The AQAPP subscribes to the PO_Cancel event.

1. Select an Event Page
 - a. Enter information in the following fields:
 - * **Application**—Select AQAPP.
 - * **Message Type**—Select AQAPP.
 - b. Select PO_Cancel and click Next.
2. Define Application View Page
 - a. Import attributes from the common view and click Next.
3. Define Mapping Page
 - a. Define a new mapping:
 - * Click New and map the following:
Id Copyfields Id
 - * Click OK.
 - b. Click Finish.

Create Content Based Routing

When an event is published, it is automatically routed to any subscriber to that event by default. If the routing of an event needs to be based on a value in the payload (message or message header) then Content Based Routing is required. In this scenario all changes to the purchased orders must be approved and therefore must be routed to Oracle Workflow to apply our business logic.

The logic to be applied for the Events PO_Insert, PO_Update & PO_Delete is:

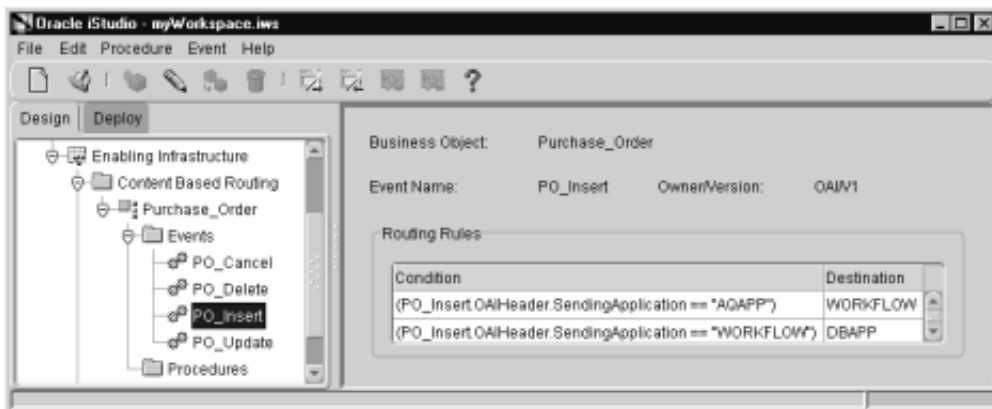
If the source application is AQAPP then route to destination application WORKFLOW. The Wizard steps are:

- Source Page: Select OAI_Header.SendingApplication
- Chose Operator Page: Select =
- Chose Value Page: Enter AQAPP
- Addition Condition Page: Select Radio Button Complete & press Finished
- Destination Page: Select WORKFLOW

If the source application is WORKFLOW, then route to destination application DBAPP. The Wizard steps are:

- Source Page: Select OAI_Header.SendingApplication
- Choose Operator Page: Select =
- Chose Value Page: Enter WORKFLOW
- Addition Condition Page: Select Radio Button Complete & press Finished
- Destination Page: Select DBAPP

Repeat for events: PO_Update and PO_Delete

Figure A-13 Completed Content Routing in iStudio

Create an Oracle Workflow Process Bundle

A process bundle enables related business processes to be grouped and transferred to the Oracle Workflow environment where user-defined business logic is applied.

Each business process enables related publish, subscribe, invoke, and implement activities to be grouped and placed in the Oracle Workflow Business Event System.

Step 1 Create a Process Bundle

The following steps describe creating the PO process bundle using iStudio:

1. From the project tree, expand the Workflow node and drill down to Process Bundle.
2. Right click on Business Processes and select New. The Create Process Bundle dialog displays.
3. Enter PO in the Process Bundle Name field and click OK.

Step 2 Create Business Process

The following steps describe creating the PO business process using iStudio:

1. Expand the Process Bundle node on the project tree and drill down to Business Processes.
2. Right click on Business Processes and select New. The Create Business Process dialog displays.

3. Enter PO in the Business Process Name field and click OK.

Step 3 Create the Subscribe and Publish Activities

The Oracle Workflow business process uses the common view therefore, transformation and mapping is not required. The following lists the types of activities:

- **Subscribe**—Oracle Workflow receives a message from OracleAS InterConnect.
- **Publish**—Oracle Workflow sends a message to OracleAS InterConnect.
- **Invoke**—Oracle Workflow sends a request message to OracleAS InterConnect and receives a reply.
- **Implement**—Oracle Workflow receives a request from OracleAS InterConnect and sends a reply.

In this scenario, the PO_Insert, PO_Update, and PO_Delete messages are routed to Oracle Workflow to apply business logic. Based on this logic, messages are either sent to the Order Fulfillment Application or the PO_Cancel message is sent to the Legacy Application. Oracle Workflow must:

- Subscribe to PO_Insert and publish PO_Insert.
- Subscribe to PO_Update and publish PO_Update.
- Subscribe to PO_Delete and publish PO_Delete.
- Publish PO_Cancel.

Step 4 Create Subscribe Activity: PO_Insert, PO_Update, and PO_Delete

The following steps describe creating the subscribe activity using iStudio:

1. From the project tree, expand the Workflow node and drill down to Business Processes.
2. Right click on PO business process and select New Subscribe Activity. A right click on any item displays a pop-up box.
3. Select Event PO_Insert and click OK.

Repeat these steps for the PO_Update and PO_Delete events, substituting the correct values where necessary.

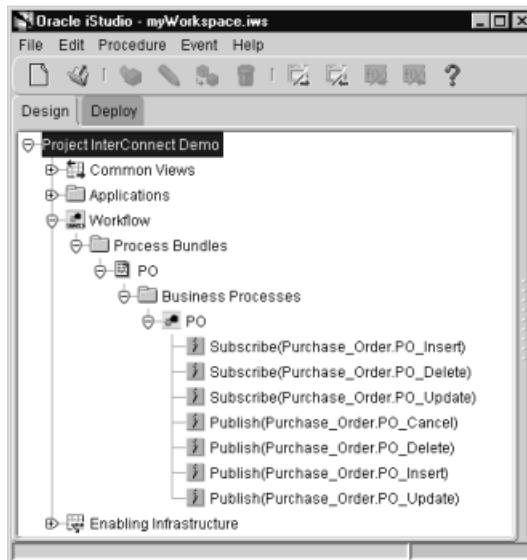
Step 5 Create Publish Activity: PO_Insert, PO_Update, PO_Delete, and PO_Cancel

The following steps describe creating the publish activity using iStudio:

1. From the project tree, expand the Workflow node and drill down to Business Processes.
2. Right click on PO business process and select New Publish Activity. A right click on any item displays a pop-up box.
3. Select Event PO_Insert and click OK.

Repeat these steps for the PO_Update, PO_Delete, and PO_Cancel events, substituting the correct values where necessary. The subscribe and publish events appear in the Design Object Navigator under the PO node as shown in [Figure A-14](#).

Figure A-14 *Subscribe and Publish Activities in iStudio*



Deploy the Process Bundle to Oracle Workflow

Deploying the Oracle Workflow process bundle accomplishes the following:

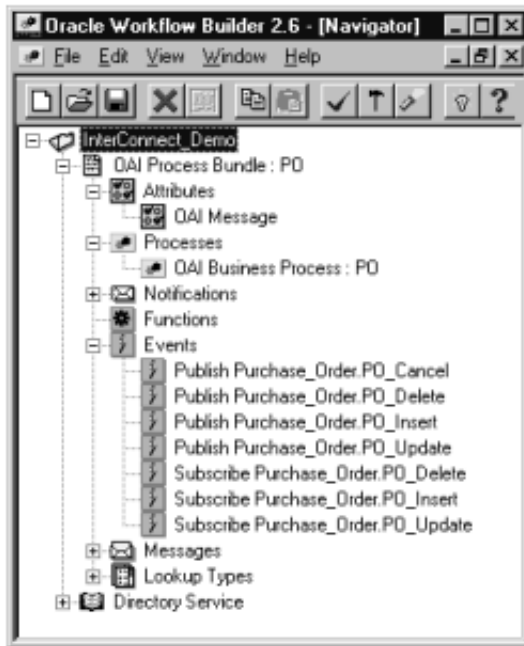
- Places the event definitions in the Oracle Workflow Business Event System.
- Creates a default Oracle Workflow file (.wft).
- Launches the Oracle Workflow Builder and Monitor.

See Also: [Chapter 7, "Using Oracle Workflow"](#)

The following steps describe deploying the process bundle to Oracle Workflow:

1. Right click the Workflow node on the Deploy tab in iStudio and select Deploy. The Deploy dialog displays.
2. Select Event Definitions to Workflow Business Event System, then Process Definitions for File in the Deploy to Workflow box and click OK. The Workflow BES Login dialog displays.
3. Log in to Oracle Workflow using the correct username, password, and URL. Click OK. The Deploy dialog displays.
4. Enter a file name for the Oracle Workflow file such as `InterConnect_Demo.wft` in the File Name field and click Open. Oracle Workflow is started with `InterConnect_Demo`.

Figure A-15 Completed Deployment in Oracle Workflow



Creating Objects in Oracle Workflow for Modeling

Lets' review the original requirement.

"An administrator must approve all changes such as insert, update, & delete before they are applied to the Order Fulfillment System. If a change is approved, it is sent to the Order Fulfillment System. If a change is rejected, then a cancellation notification is sent back the Legacy system."

This Business Logic can be implemented in Oracle Workflow. The Oracle Workflow components require are:

- An Item Type equivalent to a Project
- An Attribute An object to hold the message in the event
- A Process To model the Business Logic
- Events For the modeling in the process.
- A Notification To notify the administrator in the Oracle Workflow Monitor.

Components transferred from iStudio.

- Item Type: OAI Process Bundle: PO
- Attribute: OAI Message
- Process: OAI Business Process: PO
- Events:
 - Publish `Purchase_Order.PO_Cancel`
 - Publish `Purchase_Order.PO_Insert`
 - Publish `Purchase_Order.PO_Update`
 - Publish `Purchase_Order.PO_Delete`
 - Subscribe `Purchase_Order.PO_Insert`
 - Subscribe `Purchase_Order.PO_Update`
 - Subscribe `Purchase_Order.PO_Delete`

Oracle Workflow components are required to create a Notification.

Message

The message a notification activity will send.

Lookup Type

A static list of values that can be referenced various object. For example a message attribute can reference a lookup type as a means of providing a list of possible responses to the performer of a notification.

Notification

When the workflow engine reaches a notification activity, it issues a Send() API call to the Notification System to send the message to an assigned performer. When a performer responds to a notification activity, the Notification System processes the response and informs the workflow engine that the notification activity is complete.

What Oracle Workflow provides.

Oracle Workflow has a set of pre-defined item types with standard functionality. The Standard item type contains generic activities that can be copied in a users item type. In this scenario we will be using the Lookup Type Approval.

Copy Lookup Type (Approval)

As described, the user must create a Oracle Workflow Notification. The notification has two dependent objects, A lookup Type and a Message. The Lookup Type (Approval) can be copied from the standard item type.

Create an Oracle Workflow Message

The following steps describe creating a new Oracle Workflow message called Insert_Message

1. In the Object Navigator right click on the Message Node and select New to launch the property sheet. In each tab add the following entries:
2. Message Tab
 - Internal Name: Insert_Message
 - Display Name: Insert Message
 - Description: Insert Message
3. Body Tab:
 - Subject: Insert Message
 - Text Body: A record has been Inserted in the Purchase Order Table.

4. Result Tab:
 - Display Name: Insert_Message
 - Description: Insert_Message
 - Lookup Type: Approval (From Lookup Type)
5. Click OK

Using the default Copy and Paste functionality create the following messages using message Insert_Message as the template:

- Update_Message—Repeats the above steps and use the same setting, changing all references to insert to update.
- Delete_Message—Repeats the above steps and use the same setting, changing all references to insert to Delete.

Create an Oracle Workflow Notification

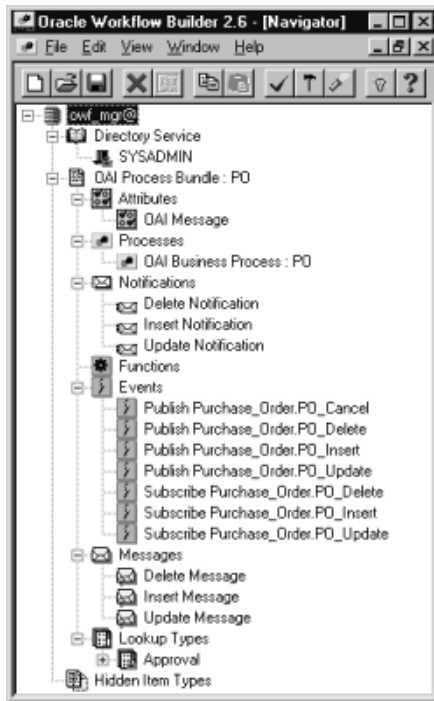
The following steps describe creating a new Oracle Workflow Notification.

1. In the Object Navigator right click on the Notification Node and select New to launch the property sheet. In each tab add the following entries:
2. Activity Tab:
 - Internal Name: Insert_Notification
 - Display Name: Insert_Notification
 - Description: Insert_Notification
 - Message: Insert_Message (Created previous step)
 - Result Type: Approval (From Lookup Type)
3. Click OK

Using the default Copy and Paste functionality create the following notifications using notification Insert_Notification as the template:

- Update_Notification—Repeats the above steps and use the same setting, changing all references to insert to update.
- Delete_Notification—Repeats the above steps and use the same setting, changing all references to insert to delete.

Figure A-16 Completed Oracle Workflow Notifications



Modeling Business Logic in Oracle Workflow

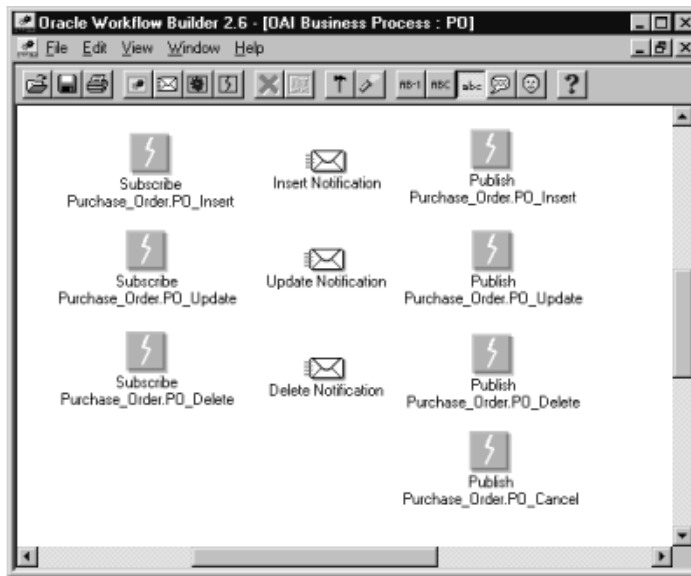
Now that all of the required objects have been created, the business logic can be modeled. The following steps describe this process.

1. In the Oracle Workflow Object Navigator, expand the OAI Process Bundle: PO item type.
2. Expand the Processes node.
3. Right click on OAI Business Process: PO and select Process Details.

Another way to display the process details is to double-click on OAI Business Process: PO.

4. Drag and drop the following from the Oracle Workflow Object Navigator to the Oracle Workflow Workspace:
 - Insert_Notification
 - Update_Notification
 - Delete_Notification
5. Rearrange the items as shown in Oracle Workflow Builder.

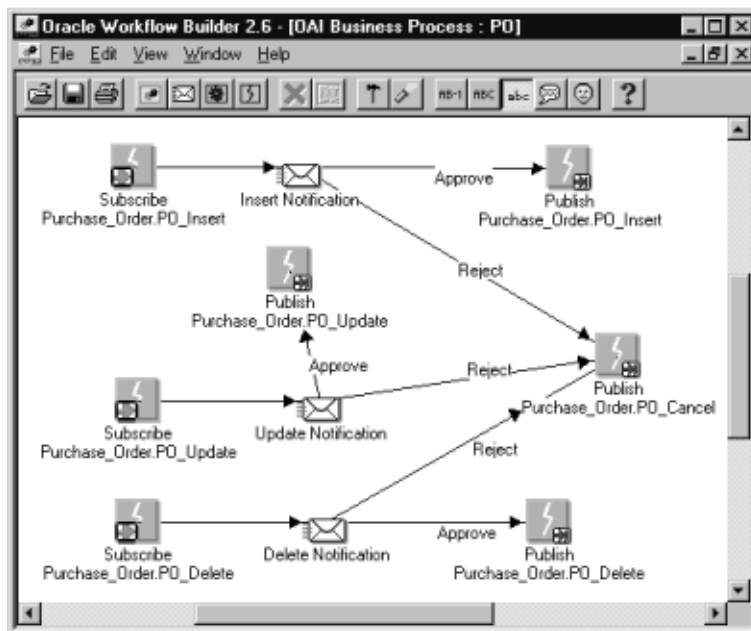
Figure A-17 Items Arranged in Oracle Workflow Builder



6. The subscribe events are the entry point into this process, therefore, the Start/End Property for each event must be edited and set to START. To launch the property sheet of each object by Double clicking on the object. The Start/End property is Under the Node tab.
 - Subscribe event `Purchase_Order.PO_Insert`
 - Subscribe event `Purchase_Order.PO_Update`
 - Subscribe event `Purchase_Order.PO_Delete`
7. The publish events are the exit point from this process, therefore, the Start/End Property for each event must be edited and set to END. To launch the property sheet of each object Double click on the object. The Start/End property is under the Node tab.
 - Publish Event `Purchase_Order.PO_Insert`
 - Publish Event `Purchase_Order.PO_Update`
 - Publish Event `Purchase_Order.PO_Delete`
 - Publish Event `Purchase_Order.PO_Cancel`

8. The notifications must be assignment to a performer in order for that person to receive the notification in the Oracle Workflow Monitor. The Performer value property should be set to SYSADMIN for each notification. To launch the property sheet of each object by Double clicking on the object. The Performer Value field is under the Node tab.
 - Notification: Insert_Notification
 - Notification: Update_Notification
 - Notification: Delete_Notification
9. Mapping lines need to be drawn between the object to define the process flow. Line are drawn by right click and drag fro one object to another
 - a. Draw a mapping line from Subscribe Purchase_Order.PO_Insert to Insert_Notification.
 - b. Draw a mapping line from Insert_Notification to Publish Purchase_Order.PO_Insert and select Approve from the pop-up that will automatically launch when the line is drawn.
 - c. Draw a mapping line from Insert_Notification to Publish Purchase_Order.PO_Cancel and select Reject from the pop-up that will automatically launch when the line is drawn.
 - d. Repeat steps for Update & Delete objects.
10. Save work to the database.

Figure A-18 Completed Business Process in Oracle Workflow Builder



Deployment

The Oracle Workflow item type OAI Process Bundle:PO is validated when saved to the database. The next step is to deploy the OracleAS InterConnect objects.

Setting Queues

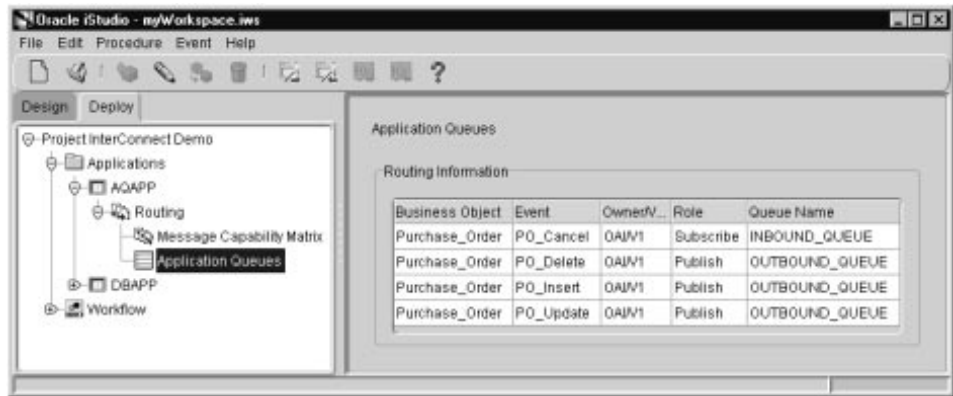
The AQAPP application in iStudio corresponds to the Advanced Queuing adapter that communicates with the legacy application. The legacy application, through a database trigger, places inserted, updated, and deleted records onto a queue using Oracle Advanced Queuing. To communicate to and from the OracleAS InterConnect environment, the adapter must be configured to send and receive on those external queues.

The following steps describe this task.

1. On the Deploy tab in iStudio, expand the Applications tree and drill down to AQAPP.
2. Expand the AQAPP node and drill down to the Routing node.
3. Expand the Routing node and select Application Queues. The Application Queues property sheet displays on the right side of the iStudio window.
4. Select Edit from the Edit menu on the menu bar. This will launch the Edit Application Queue dialog
5. Add the application Queue name to each event:

Queue Name	Event
INBOUND_QUEUE	PO_Cancel
OUTBOUND_QUEUE	PO_Insert, PO_Update, and PO_Delete

6. Click OK.

Figure A-19 Application Queues in iStudio

Pushing Metadata

Each adapter has different cache settings to minimize communication to the repository and to improve performance. Pushing metadata synchronizes the adapter and repository metadata. The following steps describe this task:

1. Select File from the menu bar, then Push Metadata. The Push Metadata dialog displays.
2. Select the applications to which to push metadata and click OK.

Exporting and Installing Code

Depending on the adapter type, there is code that must be exported to a file and installed in the target application database. The following steps describe exporting the code using the Export Application dialog in iStudio.

1. Select File from the menu bar, then select Export. The Export Application dialog displays.
2. Select the application(s) to export code.
3. Enter the file prefix in the File Prefix field and click OK.

The resulting text files are a SQL*Plus script that is executed on the target schema.

See Also: ["Exporting Stored Procedures"](#) on page 5-18

Example

The following example helps to explain the exporting and installing code task. This example is based on the following:

- **Adapter type**—Database Adapter
- **iStudio application**—DBAPP
- **Subscribe event**—PO_Delete

```
PROCEDURE sub_PO_Delete_OAI_V1 (   POID           IN NUMBER,
                                  POITEM          IN LONG,
                                  PRICE           IN LONG,
                                  QUANTITY        IN NUMBER,
                                  LAST_UPDATED     IN DATE,)
AS
    v_poid NUMBER :=poid;
BEGIN
    DELETE FROM PO WHERE v_poid = poid;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN NULL;

END sub_PO_Delete_OAI_V1;
```

Conclusion

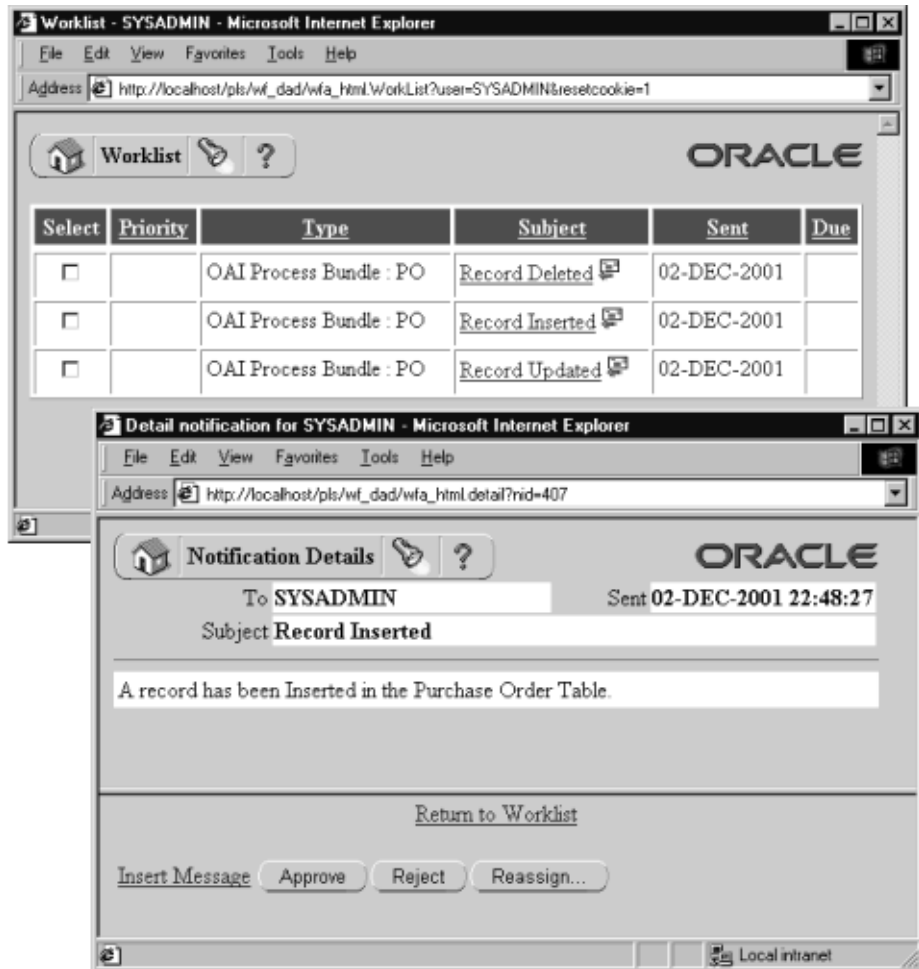
The final step in this scenario is to test the integration.

- A record should be inserted into the Legacy System.
- 1. The legacy system's database trigger enqueues the record onto its `OUTBOUND_QUEUE`.
- 2. OracleAS InterConnect received the message, performs transformations, converts data to a common view and routes the message to Oracle Workflow.
- 3. Oracle Workflow applies the business logic and issues a notification.
- 4. The `SYSADMIN`:
 - Logs on to Oracle Workflow Monitor
 - Receives the `Insert_notification`.
 - Approves the record.
- 5. OracleAS InterConnect received the message, performs transformations, cross references the primary keys, converts data to the application View, and routes the message to Order Fulfillment System
- 6. The deployed code receives the message and inserts the record into the Order Fulfillment system
- 7. Oracle Enterprise Manager
 - The inserted record can be examined.
 - Integration throughput can be monitored.

This process should be repeated for Update & Delete.

Figure A-20 Oracle Workflow Home Page

Figure A-21 Oracle Workflow Worklist and Notification Details



Using the Data Definition Description Language

This appendix describes how to use the data definition description language (D3L) in its native format message-to-application view and application view-to-native format message translations.

This appendix contains these topics:

- [About D3L](#)
- [Native Format Message and D3L File Example](#)
- [D3L File Structure](#)
- [D3L Integration with OracleAS InterConnect Adapters](#)
- [Installing D3L](#)
- [Configuring D3L](#)
- [D3L Use Case](#)
- [Additional D3L Sample Files and DTD](#)

About D3L

This section contains these topics:

- [What Is D3L?](#)
- [When Is D3L Used?](#)

What Is D3L?

D3L is an XML-based message description language that describes the structure that an application's native, non-XML format message (known also as its native view) must follow to communicate with Oracle Application Server InterConnect. Oracle Corporation provides several transport adapters (known as OracleAS InterConnect Adapters) that interact with the D3L message description language:

- FTP
- HTTP(S)
- MQ Series
- SMTP

OracleAS InterConnect Adapters perform the following tasks:

- Validate the D3L message description (XML) files during runtime initialization.
- Use the D3L translation engine (subcomponent of the bridge) to translate messages from:
 - Native format message to application view
 - Application view to native format message
- Transport message payload data between a participating application and Oracle Application Server InterConnect.

Note: Native format messages already in XML format are not translated by OracleAS InterConnect Adapters if the `ota.type` parameter is set to `XML` in the `adapter.ini` file.

See Also: "[D3L Integration with OracleAS InterConnect Adapters](#)" on page B-31 for detailed information on how D3L is integrated with OracleAS InterConnect Adapters

When Is D3L Used?

Not all applications use XML as their native message payload format. Applications also use other native formats, which are best described as structured records of bytes and/or characters. For these native formats to be successfully translated into a format understood by other applications, the content of their messages must follow a predefined, structured set of rules. This structured format can then be translated into an application view, transformed into a common view, and understood by other applications.

D3L provides both a predefined, structured set of rules and translation capabilities for native format messages. Specifically, D3L provides:

- An XML-based message description language that describes the contents of native format messages
- A translation engine that uses the instructions defined in the D3L file to translate the native format message contents into an application view, or vice versa

The D3L descriptions must comply with a syntax defined by the D3L document type definition (DTD). D3L enables you to describe the record layout of binary, string, structured, and sequence data. Use D3L only when the number of fields in the underlying native format message is fixed and known. D3L is not suitable for:

- Descriptions of arbitrarily-structured data (like regular XML)
- Name-value pair data
- Conditional data structures, which require token look-ahead to parse.

See Also:

- ["Native Format Message and D3L File Example"](#) on page B-4
- ["Supported D3L Data Types"](#) on page B-12
- ["D3L DTD"](#) on page B-75

Native Format Message and D3L File Example

This section provides an example of how the contents of a native format message are:

- Described in a D3L file
- Configured with the required D3L file

Satisfying both these requirements enables the native format message to be successfully translated. This section contains the following topics:

- [Native Format Message Contents Description in a D3L File](#)
- [Native Format Message Configuration with a D3L File](#)

Native Format Message Contents Description in a D3L File

This example shows an application's native format message (named `price`) that contains payload data for updating the price of personal computer model number 2468 to 199.99. The native message uses the following format to describe this payload data:

```
message ::= <action> <model> <price>
```

Where...	Is...
<action>	UPDATE_PRICE
<model>	2468
<price>	199.99

The payload data must strictly follow the structure defined in a D3L file (for this example, `price.xml`) for the D3L translation engine (subcomponent of the bridge) to successfully translate it into an application view. [Figure B-1](#) shows how a D3L file (`price.xml`) defines the structure that the native format message `price` must follow to successfully define the three preceding elements of payload data.

Figure B–1 Native Format Message Payload Data and D3L File Syntax

All three payload data elements are defined as strings with different delimiters for separating their data.

Native Format Message Configuration with a D3L File

When the D3L translation engine receives a native format message (for example, `price`), it must determine the exact D3L file to use to verify the native format message contents (for example, `price.xml`). This section describes the methods for configuring the correct D3L file with the native format message:

- [adapter.ini Parameter File Setting](#)
- [Message Header Attributes](#)

adapter.ini Parameter File Setting

The `ota.d3ls` parameter in the `%ORACLE_HOME%\oai\9.0.4\adapters\application\adapter.ini` file enables you to define the D3L file to use with the native format message. For example:

```
ota.d3ls=price.xml
```

This setting enables `price.xml` in [Figure B–1](#) to be configured with the native format message `price`. When the D3L translation engine receives the native format

message from the bridge, it retrieves the correct D3L file based on this parameter setting. Multiple D3L files can also be defined, for example:

```
ota.d3ls=price.xml,emp.xml,booking.xml
```

The D3L translation engine compares the data structure in the native format message to each D3L file until it finds the correct one to use for translation, unless one of the methods described in "[Message Header Attributes](#)" is used.

Message Header Attributes

The D3L file includes message header attributes that guide the D3L engine in choosing the correct D3L file for translating a native format message to an application view. The values for these message header attributes match with the same settings in the native format message.

Message header attribute values override the approach of comparing each D3L file defined with the `ota.d3ls` parameter in the `adapter.ini` file with a native format message.

Two message header attribute setting methods are available:

- [Name/Value Pair Message Header Attributes](#)
- [Magic Value Message Header Attribute](#)

Both methods enable the D3L translation engine to use the correct D3L file for translation after receiving the native format message.

Note: When the correct D3L file is selected (and a successful translation has taken place), the `<message>` element attributes `name` and `object` in the D3L file define the Oracle Application Server InterConnect event name and business object, respectively.

Name/Value Pair Message Header Attributes

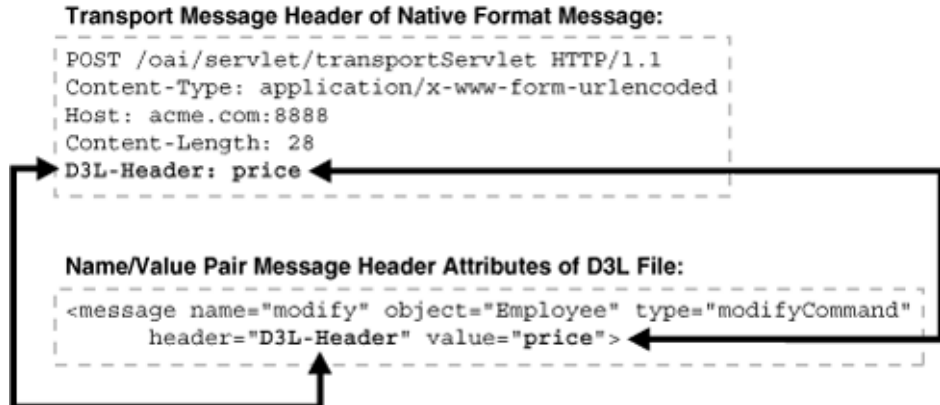
OracleAS InterConnect Adapters, such as the HTTP adapter, make their protocol level transport properties available to the D3L translation engine, including custom properties added by a sending application (for example, an HTTP client). The D3L file `<message>` element enables the user to specify two attributes, `header` and `value`, that match the protocol level headers in a received native format message.

For example, a third-party application uses the custom transport header `D3L-Header` to communicate to the D3L translation engine which D3L file to use to translate an incoming native format message. The following steps must be performed:

- Set the `D3L-Header` parameter in the transport message header to a value that matches the `value` attribute setting of the `<message>` element in the D3L file.
- Set the `header` attribute of the `<message>` element in the D3L file to `D3L-Header` to match the `D3L-Header` parameter name in the transport message header.

Figure B-2 provides an example using the HTTP adapter where `D3L-Header` and `price` are the header name and header value, respectively. Each are used to match a native format message with the correct D3L file. The D3L translation engine retrieves the correct D3L file based on these settings.

Figure B-2 Name/Value Pair Message Header Attributes



The D3L engine supports a rudimentary pattern matching capability in the `value` attribute of the D3L message element.

A D3L author can create a D3L definition, such as:

```
<message type="CrtCust" header="filename" value="cust_create%" ...
```

This example is relevant for the FTP adapter, which provides a header property called `filename` that holds the name of a received file.

The above D3L will be selected to parse incoming files which `filename` match the name pattern in the `value` attribute, such as,

```
cust_create01 <-- match!
cust_create02 <-- match!
po_int_ext01
cust_create03 <-- match!
po_int_ext02
```

The "fuzzy" character in the pattern ("%") can only appear at two places in the attribute string-value, as either the first or the last character, or both the first and last characters.

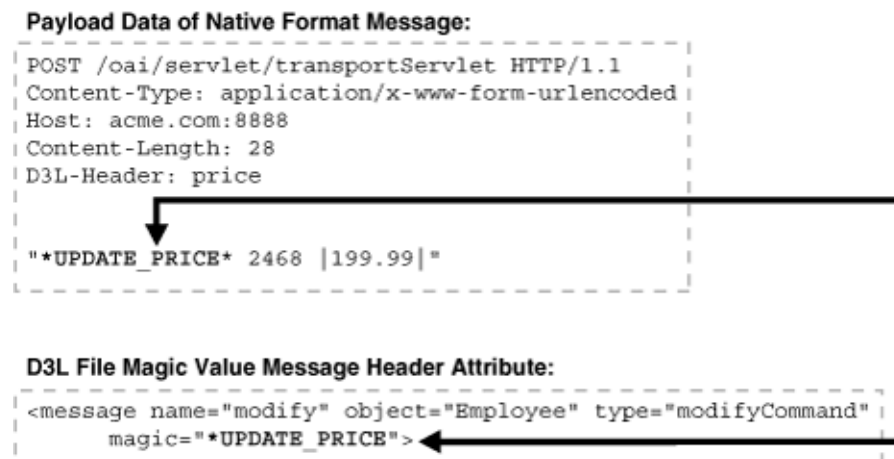
For example, the following patterns are acceptable:

- `%endstring` — such as `%.csv`
- `startstring%` — such as `po_int%`
- `%substring%` — such as `%create%`

Magic Value Message Header Attribute You can set the `magic` attribute of the `<message>` element in the D3L file to match the first *n* bytes of payload data in a native format message. This feature enables you to define the D3L file to use with the native format message. When a native format message is received by the D3L translation engine, the `magic` values of all D3L files are compared against the first *n* bytes of the native format message. The `magic` values must be long enough to be unique across all registered D3Ls for a given adapter instance.

Figure B-3 provides an example where `*UPDATE_PRICE` is the value that configures the native format message with the correct D3L file.

Figure B-3 Magic Value Message Header Attribute



The D3L translation engine retrieves the correct D3L file based on these settings.

The D3L attribute `startsat` of the `message` element enables the D3L author to specify the byte location that magic matching should start.

Having this attribute allows the D3L definition:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">

<message name="newBook" type="BookType" object="BookObj" magic="ISBN#"
  startsat="12">
```

This D3L definition will trigger if a native message contains the byte character sequence `ISBN#` in byte positions 12 to 16, counting from 0.

See Also:

- ["Native Format Message to Common View Incoming Message Translations"](#) on page B-32 for additional information on message header attributes
- ["Additional D3L Sample Files and DTD"](#) on page B-68 for additional D3L file examples

D3L File Structure

This section describes the contents of a sample D3L file named `book_reply.xml`.

```
1 <?xml version="1.0" encoding="US-ASCII"?>
2 <!DOCTYPE message SYSTEM "d3l.dtd">
3 <message name="replyFlight" type="BookingReplyType" object="Booking"
4 header="D3L-Header" value="replyOptions">
5   <unsigned4 id="u4" />
6   <unsigned2 id="u2" />
7   <struct id="DateTimeRecord">
8     <field name="DateInfo">
9       <date format="MMDDYY">
10        <pfxstring id="datstr" length="u4" />
11      </date>
12    </field>
13    <field name="TimeHour"><limstring delimiter="*" /></field>
14    <field name="TimeMinute"><limstring delimiter="*" /></field>
15  </struct>
16  <struct id="ItinRecord">
17    <field name="DepartureTime"><typeref type="DateTimeRecord" /></field>
```

```
18         <field name="ArrivalTime"><typeref type="DateTimeRecord" /></field>
19     </struct>
20     <pfxarray id="ItinArray" length="u2">
21         <typeref type="ItinRecord" />
22     </pfxarray>
23     <struct id="BookingReplyType">
24         <field name="AirportCodeFrom"><limstring delimiter="*" /></field>
25         <field name="AirportCodeTo"><limstring delimiter="*" /></field>
26         <field name="Itineraries"><typeref type="ItinArray" /></field>
27     </struct>
28 </message>
```

Lines 1 through 2

These lines define standard information, such as the Prolog and Document Type Declaration (DTD) that must always be these values (for example, specifying `d31.dtd` as the DTD).

Lines 3 through 4

These lines define the following:

- Message element attribute `name` (value `replyFlight`), which must correspond to the associated Oracle Application Server InterConnect application view event name defined in iStudio. The D3L file can also be imported in iStudio when defining the message attributes of an event (the name of which must match the name attribute of the D3L `<message>` element).
- Message element attribute `type` (value `BookingReplyType`) names a structure that is defined in subsequent lines of this D3L file.
- Message element attribute `object` (value `Booking`), which must match the Oracle Application Server InterConnect business object defined in iStudio.
- Message element attribute `header` (value `D3L-Header`), which is identified in the set of protocol level transport message headers associated with a native format message.
- Message element attribute `value` (value `replyOptions`), which must match the actual value of the corresponding protocol level transport message header (defined through the header attribute).

See Also:

- ["Creating Business Objects"](#) on page 3-3
- ["Creating Events"](#) on page 4-3
- ["Name/Value Pair Message Header Attributes"](#) on page B-6
- ["Task 6: Import a D3L File in iStudio"](#) on page B-40

Lines 5 through 6

These lines define an unsigned, four-byte integer and unsigned, two-byte integer. These data type declarations are named `u4` and `u2`, respectively, so they can be referred to later.

Lines 7 through 15

These lines define the fields of a structure named `DateTimeRecord`:

- Field `DateInfo` defines a date format of `MMDDYY` and length prefixed by an unsigned four-byte integer.
- Field `TimeHour` defines a string delimited by the character `*`.
- Field `TimeMinute` defines a string delimited by the character `*`.

Lines 16 through 19

These lines define the fields of the structure named `ItinRecord`:

- Fields `DepartureTime` and `ArrivalTime` both consist of the `DateTimeRecord` structure defined in ["Lines 7 through 15"](#) on page B-11.

Lines 20 through 22

These lines define a length-prefixed array named `InitArray`, where each array element is of type `ItinRecord`.

Lines 23 through 28

These lines define the fields of the message structure `BookingReplyType` (which satisfies the `BookingReplyType` type declaration in the message document element, as shown in ["Lines 3 through 4"](#)):

- Field `AirportCodeFrom` is a string delimited by the character `*`.
- Field `AirportCodeTo` is a string delimited by the character `*`.
- Field `Itineraries` is a field of type `ItinArray` (which is an array of `ItinRecord`).

Supported D3L Data Types

D3L supports use of the following data types and declarations in a D3L file:

- [Signed or Unsigned Integers](#)
- [Floating Point Numbers](#)
- [Strings](#)
- [Structures](#)
- [Sequences](#)

Signed or Unsigned Integers

D3L supports signed or unsigned integers that can be one, two, four, or eight octets in size, and in big or little endian octet ordering.

Example B-1 *Quantity Field*

```
<field name="quantity">  
  <unsigned4 endian="big" align="6"/>  
</field>
```

The field "quantity" defines a four byte unsigned binary integer, using big (default) endian, and at an alignment of 6 bytes. For example, D3L will ensure that the reading or writing of this integer will start at a position in the buffer, so that $\langle position \rangle \text{ modulus } \langle alignment \rangle = 0$.

Note: *Little Endian* means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. whereas *Big Endian* means that the high-order byte of the number is stored in memory at the lowest address, and the low-order byte at the highest address.

Data example*Byte addresses (hex):*00 01 02 03 04 05 06 07 08 09 0A 0B*Byte (hex):*00 00 00 00 00 00 **80 FF FF FF** 00 00*Parsed value (dec):*

$$\text{quantity} = 128 \times 256^3 + 255 \times 256^2 + 255 \times 256^1 + 255 \times 256^0 = 2164260863$$
Example B-2 Weight and Length Field

```
<field name="weight"> <unsigned2 align="3"/> </field>
<field name="length"> <unsigned2 align="3"/> </field>
```

The fields "weight" and "length" define two bytes unsigned binary integers, using big endian, and at an alignment of 3 bytes.

Data example*Byte addresses (hex):*

.. 07 08 09 0A 0B 0C 0D 0E ..

Byte (hex):.. 00 00 **EE 88** 00 **22 F0** 00 ..*Parsed value (dec):*

$$\begin{aligned} \text{weight} &= 238 \times 256^1 + 136 \times 256^0 = 61064 \\ \text{length} &= 34 \times 256^1 + 240 \times 256^0 = 8944 \end{aligned}$$
Example B-3 Temperature and Pressure Field

```
<field name="temperature"> <signed2 endian="little" /> </field>
<field name="pressure"> <unsigned4 endian="big" /> </field>
<field name="wind"> <unsigned2 endian="little" align="4" /> </field>
```

The field "temperature" defines a two byte signed binary integer, using little endian, and no alignment.

The field "pressure" defines a four byte unsigned binary integer, using big endian, and no alignment.

The field wind defines a two byte unsigned binary integer, using little endian, and a 4 byte alignment.

Data example

Byte addresses (hex):

.. 60 61 62 63 64 65 66 67 68 69 ..

Byte (hex):

.. **EF FE** 00 00 **04 0A** 00 00 **3C 00** ..
 ^ ^ ^
 little end big end alignment

Parsed value (dec):

temperature = $256 \times 256 - (239 \times 256^0 + 254 \times 256^1) = -273$
pressure = $4 \times 256^1 + 10 \times 256^0 = 1034$
wind = $60 \times 256^0 + 0 \times 256^1 = 60$

Floating Point Numbers

D3L supports single and double-precision, IEEE format, floating-point data. Single precision floating point numbers (known as floats) take up four bytes/octetets, whereas double precision floating point numbers (known as doubles) take up eight bytes/octetets.

Example B-4 Distance and Age Field

```
<field name="distance"> <double align="6"/> </field>  
<field name="age">      <float />              </field>
```

The field "distance" defines an eight byte double-float (floating-point value according to the IEEE 754 floating-point double precision bit layout), at an alignment of 6 bytes.

The field "age" defines a four byte single-float (floating-point value according to the IEEE 754 floating-point single precision bit layout).

Note: The IEEE 754 floating-point format is parsed and produced by the following Java class methods:

```

java.io.DataInput.readFloat()
java.io.DataInput.readDouble()
java.io.DataOutput.writeFloat()
java.io.DataOutput.writeDouble()

```

Data example

Byte addresses (hex):

```

..77 78 03 04 05 06 07 08 09 0A 0B xx xx ..

```

Byte (hex):

```

00 00 D2 47 D3 CE 16 2A B1 A1 5E 5D 6B 0B ..
          ^ double                ^ float

```

Parsed value (dec):

```

distance = 1 x 1038
age = 1 x 1018

```

Strings

D3L supports the following string types:

- Constant length strings (without delimiters, and with optional padding to fill out empty spaces)
- Delimited strings (can be delimited by an arbitrary delimiter character)
- Length-prefixed strings (where the length prefix is a numeric type). Numeric types are one of the binary integer types described in ["Signed or Unsigned Integers"](#) on page B-12 or are a number stored as a string.
- Strings terminated by a specified character.
- Strings terminated by a delimiter defined by an enclosing limarry structure.
- Four date formats: MMDDYY, DDDMMYY, MMDDYYYY, DDDMMYYYY, where the information is stored as a string in one of these formats with any separator character between month, date, and year (for example, 12!24=01).

- Numbers not defined as binary data, but as strings. Any one of the three string formats can define a number (either an integer or a floating-point entity). In iStudio, a D3L field of type number is handled as a double.

Example B-5 Constant length strings—padstring

```
<field name="CURRENCY_CODE">
  <padstring length="4" padchar=" " padstyle="tail"/>
</field>
<field name="COUNTRY_CODE">
  <padstring length="2" padchar="" padstyle="none"/>
</field>
<field name="TO_USD_RATE">
  <padstring length="12" padchar="0" padstyle="head"/>
</field>
```

The field "CURRENCY_CODE" defines a fixed length string of 4 characters. Any blank (" ") characters (pads) towards the end (padstyle="tail") of the string are not considered part of the data value.

The field "COUNTRY_CODE" defines a fixed length string of 2 characters. Since padstyle is "none", all characters in this field are part of the data value.

The field "TO_USD_RATE" defines a fixed length string of 12 characters. Any zero's ("0" pads) at the beginning (padstyle="head") of the string are not considered part of the data value.

Data example

Native byte(character) stream:

```
GBP UK000012550.00
```

Parsed values:

```
CURRENCY_CODE = 'GBP'
COUNTRY_CODE  = 'UK'
TO_USD_RATE   = '12550.00'
```

Example B-6 Delimited strings—limstring

```
<field name="State">    <limstring delimiter="." /> </field>
<field name="Region">  <limstring delimiter="." /> </field>
<field name="City">    <limstring delimiter="|" /> </field>
<field name="Landmark"><limstring delimiter="|" /> </field>
<field name="Street">  <limstring delimiter="+" /> </field>
```

The five fields "State", "Region", "City", "Landmark" and "Street" are delimited strings each respectively enclosed by ".", ":", "|", "|" and "+".

Data example

Native byte (character) stream:

```
.FL..Florida Keys.|Key West||Ernest Hemingway Museum|+Whitehead St.+
```

Parsed values:

```
State = 'FL'
Region = 'Florida Keys'
City = 'Key West'
Landmark = 'Ernest Hemingway Museum'
Street = 'Whitehead St.'
```

Example B-7 Length prefixed strings—pfxstring

```
<unsigned1 id="ubyte1" />
<unsigned2 id="ubyte2" endian="little" />
<struct>
  <field name="user">    <pfxstring length="ubyte1" /> </field>
  <field name="encr_user"> <pfxstring length="ubyte2" /> </field>
</struct>
```

The field "user" defines a string the length of which is defined by a one-byte binary integer preceding the string contents.

The field "encr_user" defines a string the length of which is defined by a two-byte binary integer preceding the string contents.

Data example

Byte addresses (hex):

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12
```

Characters:

```
03 j o e 0D 00 D U Z a c . 1 H K V m I Y
```

Parsed values:

```
user = 'joe'
encr_user = 'DUZac.1HKVmIY'
```

Example B-8 Terminated strings—termstring

```
<field name="product"> <termstring endchar=","/ > </field>
<field name="ordered"> <termstring endchar=","/ > </field>
<field name="inventory"> <termstring endchar=","/ > </field>
<field name="backlog"> <termstring endchar=","/ > </field>
<field name="listprice"> <termstring endchar="\n"/ > </field>
```

The first four fields will each be populated with input characters until the terminating/ending character (endchar) is encountered (",") whereas the last field be ended with a linefeed.

Data example

Native byte (character) stream:

```
1020,16,18,,1580.00<LF>
```

Parsed values:

```
product = '1020'
ordered = '16'
inventory = '18'
backlog = ''
listprice = '1580.00'
```

Note: The backlog field is empty.

Example B-9 Simple strings—simplestring

```
<limarray id="CSV_Type" contchar="," endchar="\n">
  <simplestring />
</limarray>
<struct>
  <field name="CSV"> <typeref type="CSV_Type" /> </field>
```

The field "CSV" references a type declaration "CSV_Type" which is a delimited array where array members are separated by commas (the continuation character contchar=",") and ended by the ending character linefeed (endchar="\n")

Data example

Native byte (character) stream:

```
5,18,2.5,255,78.75,9
```

Parsed values:

```
CSV[] = { '5', '18', '2.5', '255', '78.75', '9' }
```

Example B-10 Dates—date

```
<field name="StartDate">
  <date format="MMDDYY"> <termstring endchar="\n"/> </date>
</field>
<field name="EndDate">
  <date format="DDMMYY"> <termstring endchar="\n"/> </date>
</field>
<field name="Milestone">
  <date format="MDDYYYY"> <termstring endchar="\n"/> </date>
</field>
<field name="DueDate">
  <date format="DDMMYYYY"> <termstring endchar="\n"/> </date>
</field>
```

The four fields contain dates, representing the currently available 4 different date format.

Data example

Byte stream (characters):

```
11/16/02<LF>
24/11/02<LF>
11/20-2002<LF>
23*11*2002<LF>
```

Parsed values:

```
StartDate = Sat Nov 16 00:00:00 PST 2002
EndDate   = Sun Nov 24 00:00:00 PST 2002
Milestone = Wed Nov 20 00:00:00 PST 2002
DueDate   = Sat Nov 23 00:00:00 PST 2002
```

Note: The the D3L parser will accept any character between the DD, MM and YY (YY) characters in the native format, but will always produce the "/" separator (when translating from application message format to native message format). Currently hours, minutes and seconds are not parseable.

Example B-11 String based numbers—number

```
<unsigned1 id="u1" />
<pfxstring id="HueType" length="u1" />

<struct id="ColorDefinition">
  <field name="Red">
    <number> <padstring length="4" padstyle="head" padchar="0"/> </number>
  </field>
  <field name="Green">
    <number> <pfxstring length="u1" /> </number>
  </field>
  <field name="Blue">
    <number> <limstring delimiter="."/> </number>
  </field>
  <field name="Brightness">
    <number> <termstring endchar="|"/> </number>
  </field>
  <field name="Hue">
    <number> <typeref type="HueType"/> </number>
  </field>
</struct>
```

The declared type "u1" is an unsigned one-byte integer (0-255). The second type declaration "HueType" is a length prefixed string, where the string length will be defined in a one-byte binary integer preceding the string contents.

The field "Red" is a number defined as a fixed length string of 4 characters, which can be padded with '0's at the beginning.

The field "Green" is a number defined as a length prefixed string, where the string length will be defined in a one-byte binary integer preceding the string contents.

The field "Blue" is a number defined as a "." delimited string, the string beginning and end is demarcated by ".".

The field "Brightness" is a number defined as a string which is read from the current point until the ending character ("|") is encountered.

The field "Hue" is a number defined as a string of type "HueType" (defined above).

Data example

Byte addresses (hex) and Characters (hex values shown in italics):

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
  0  1  2  8 03  1  2  8  .  2  5  5  .  0  .  7
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E
  5  3  3  3  3  | 08 0  .  6  6  6  6  6  6

```

Parsed values:

```

Red = 128.0
Green = 128.0
Blue = 255.0
Brightness= 0.753333
Hue = 0.666666

```

Note: The parsed numbers always internally become double's.

Structures

D3L supports structured types, that is, ordered records containing other data types (predefined or user defined). Types can be nested to arbitrary depth. This means you can use structures of sequences of structures of sequences [...] to any finite depth. Recursive, self referencing, data structures, however, are not supported in D3L.

All data fields in a message format description must be named. These names are used as Oracle Application Server InterConnect message attribute names. All names within the same structure must be mutually unique.

Within a D3L file the first allowable element is <message>. The message element must refer to a <struct> (via the IDREF type attribute) which then becomes the top level data structure of the message.

Example B–12 ColorDefinition Field

```
<message type="ColorDefinition" name="myEV" object="myBO">
  <unsigned1 id="u1" />
  <pxstring id="HueType" length="u1" />
  <struct id="ColorDefinition">
    <field name="Red"> ...
  <field name="Green"> ...
```

Note: The top level structure can be placed anywhere in the D3L file (within the scope of the <message> element).

Sequences

D3L supports sequences (for example, arrays) of various types. These include:

- Delimited arrays (with arbitrary separator and terminator characters)
- Length-prefixed arrays (where the length is one of the numeric types)
- Fixed-length arrays
- Implicit-length arrays (This data type uses all remaining data in the native format message to the end of the buffer.)

The data being sequenced can be any other D3L type (predefined or user defined).

Example B–13 Delimited arrays—limarray

```
<field name="members">
  <limarray contchar=";" endchar=".">
    <limstring delimiter="." />
  </limarray>
</field>
```

The field "members" will become an array of data elements separated by semicolons (contchar=";"). The end of the array will be marked by a period (endchar="."). Each data element in the array will be a string delimited by a set of periods (delimiter=".").

Data example

Native byte (character) stream:

```
.John.;Steve.;Paul.;Todd..
```

Parsed values:

```
members[] = { 'John', 'Steve', 'Paul', 'Todd' }
```


Example B-14 Length prefixed arrays—pfxarray

```

<unsigned2 id="u2" endian="little" align="4" />
<struct>
  <field name="measurements">
    <pfxarray length="u2" > <signed1 /> </pfxarray>
  </field>

```

The field "measurements" will become an array of signed one-byte binary integers (signed1). The number of elements in the array will be determined by the unsigned two-byte binary integer at the beginning of the array.

Data example

Byte addresses (hex):

```
.. 08 09 0A 0B 0C 0D 0E 0F 10 ..
```

Bytes (hex):

```
.. 06 00 FF A2 6C 24 0E 77 ..
```

Values (dec):

```
measurements[] = { -1, -94, 108, 36, 14, 119 }
```

Example B-15 Fixed length arrays—fixarray

```

<field name="digits">
  <fixarray length="10">
    <number>
      <termstring endchar="-">
    </number>
  </fixarray>

```

The field "digits" will become an array of numbers (doubles). Each number element in the native byte format is represented as a string which is terminated by a dash (endchar="-"). The number of elements in the array will/must always be 10 (length="10").

Data example

Native byte (character) stream:

1-2-3-4-5-6-7-8-9-0-

Parsed values:

```
digits[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 0.0 }
```

Example B-16 Implicit length arrays—imparray

```
<message name="addOrders" object="Order" type="OrdersType">

  <number id="Number"> <termstring endchar="," /> </number>
  <number id="Price"> <termstring endchar=";" /> </number>
  <number id="Total"> <termstring endchar="\n"/> </number>

  <struct id="OrderLineType">
    <field name="LineNo"> <typeref type="Number" /> </field>
    <field name="ProductNo"> <typeref type="Number" /> </field>
    <field name="Quantity"> <typeref type="Number" /> </field>
    <field name="LinePrice"> <typeref type="Price " /> </field>
  </struct>

  <struct id="OrderType">
    <field name="OrderTotal"> <typeref type="Total" /> </field>
    <field name="OrderLines">
      <limarray contchar="\n" endchar="\n\n">
        <struct>
          <field name="OrderLine"> <typeref type="OrderLineType" /> </field>
        </struct>
      </limarray>
    </field>
  </struct>

  <number id="ID"> <termstring endchar="\n" /> </number>
```

```

<imparray id="OrdersArrayType">
  <struct>
    <field name="CustomerID"> <typeref type="ID" /> </field>
    <field name="CustomerName"> <termstring endchar="\n" /> </field>
    <field name="Order"> <typeref type="OrderType" /> </field>
  </struct>
</imparray>

<struct id="OrdersType">
  <field name="OrdersArray"> <typeref type="OrdersArrayType" /> </field>
</struct>
</message>

```

The "OrdersType" structure consists of a single field "OrdersArray" which is an implicit array of three fields—a "CustomerID", "CustomerName" and "Order" (of type "OrderType"). Since "OrdersArrayType" is an implicit array, it will consume all remaining bytes in the native byte input stream, i.e. the size of the array is first known when the input byte stream has been exhausted.

The field "OrderLines" is a (nested) array, where each array element is of type "OrderLineType". The "OrderLineType" is a structure of four fields—"LineNo", "ProductNo", "Quantity" and "LinePrice".

This abstract structure of the input stream would have to follow the below structure to be parseable:

```

CustomerID
CustomerName
Order:
  OrderTotal
OrderLines:
  LineNo, ProductNo, Quantity, LinePrice
  LineNo, ProductNo, Quantity, LinePrice
...

```

Data example

Native byte (character) stream:

```
1234
Boeing
1000
1,555,10,250.00;
2,666,10,750.00;
```

```
5678
Lockheed Martin
424
1,555,5,125.00;
2,777,1,100.00;
3,888,2,199.00;
```

Parsed values:

```
{OrdersArray=
  [ { CustomerName=Boeing, CustomerID=1234.0,
    Order=
      { OrderTotal=1000.0,
        OrderLines = [
          {OrderLine={LinePrice=250.0, ProductNo=555.0, Quantity=10.0,
            LineNo=1.0}},
          {OrderLine={LinePrice=750.0, ProductNo=666.0, Quantity=10.0,
            LineNo=2.0}}
        ]
      }
    },
  { CustomerName=Lockheed Martin, CustomerID=5678.0,
    Order=
      { OrderTotal=424.0,
        OrderLines = [
          { OrderLine={LinePrice=125.0, ProductNo=555.0, Quantity=5.0,
            LineNo=1.0}},
          { OrderLine={LinePrice=100.0, ProductNo=777.0, Quantity=1.0,
            LineNo=2.0}},
          { OrderLine={LinePrice=199.0, ProductNo=888.0, Quantity=2.0,
            LineNo=3.0}}
        ]
      }
    }
  ]
}
```

Data Padding

D3L supports data padding. Pads are unnamed gaps in a native format message that satisfy alignment constraints of the underlying native system. Pads are discarded in the Oracle Application Server InterConnect application view message.

The following D3L example defines a number as a left-aligned string, right padded with blanks to a field width of 10.

```
<field name="Quantity">
  <number>
    <padstring length="10" padchar=' ' padstyle="tail" />
</number>
```

The following native byte (character) stream satisfies this format:

```
9876.5____
```

Pads can also be explicitly defined between fields in a structure by using the `<pad>` element.

The following D3L example shows two fields, which are separated by a pad of size 5.

```
<struct id="PROD">
  <field name="PRODID"> <termstring endchar=";" /> </field>
  <pad length="10" />
  <field name="PRODESC"> <termstring endchar=";" /> </field>
</struct>
```

The following native byte (character) stream would satisfy this format:

```
48682HW;~~~~~WASHER AND DRYER;
[...]
```

Comma-Separated Values File Parsing with D3L

A comma-separated values (CSV) file consists of multiple lines. Each line contains values separated by commas that end when a new line is required:

```
a,b,c,d
1,2,3
```

Two string types, `termstring` and `simplestring`, have been added to make it easier to parse CSV files.

- `termstring`

String type `termstring` is a variation of `limstring`, but requires only a terminating delimiter and, not a beginning delimiter. For example:

```
<termstring endchar="," />
```

This parses any string contents until a comma is encountered.

- `simplestring`

String type `simplestring` is a special data type only used when the nearest parent structure defines a valid set of delimiters, which for the current data definition description language (D3L) library is limited to `limarray`. For example:

```
<limarray contchar="," endchar="\n">  
  <simplestring />  
</limarray>
```

These new string types provide two ways for parsing CSV files. The examples provided in the following sections use `imparray` so that input can be any number of elements, lines, or both.

- [CSVs are Assigned to Named Fields](#)
- [All CSVs are Read into an Array](#)

CSVs are Assigned to Named Fields

With this method, all CSVs on each line are assigned to named fields (fixed number of fields per line). [Example B-17](#) provides an example.

Example B-17 CSVs Assigned to Named Fields

```
<message name="createPhone" object="Phone" type="phoneRecord">  
  <imparray id="lines">  
    <struct>  
      <field name="rectype"> <termstring endchar="," /> </field>  
      <field name="quantity"><termstring endchar="," /> </field>  
      <field name="endHour"> <termstring endchar="," /> </field>  
      <field name="endMin"> <termstring endchar="," /> </field>  
      <field name="cost"> <termstring endchar="\n"/> </field>  
    </struct>  
  </imparray>
```

```

    <struct id="phoneRecord">
      <field name="csv"> <typeref type="lines" /> </field>
    </struct>
  </message>

```

The native format message payload for [Example B-17](#) is as follows:

```

4,,9,22,2324.29
''''
55,2342,11,46,728372339.57

```

All CSVs are Read into an Array

With this method, all CSVs on each line are read into an array (variable number of fields per line). [Example B-18](#) provides an example.

Example B-18 All CSVs are Read into an Array

```

<message name="createPhone" object="Phone" type="phoneRecord">
  <limarray id="linearr" contchar="#44" endchar="\n">
    <simplestring />
  </limarray>
  <impparray id="myArray">
    <struct>
      <field name="line"> <typeref type="linearr" /> </field>
    </struct>
  </impparray>
  <struct id="phoneRecord">
    <field name="csv"> <typeref type="myArray" /> </field>
  </struct>
</message>

```

The native format message payload for [Example B-18](#) is as follows:

```

4,,9,22,2324.29
55,2342,11,46,728372339.57
55,2342,11,46,728372339.57,4,,9,22,2324.29
1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0

```

Delimiter Encoding Styles

The delimiters for `limstring`, `termstring`, and `limarray` are enhanced to allow multiple characters, as well as additional encoding styles. The associated ASCII table codes are shown in parentheses:

1. Escaping using "\"— This works for "\r" (13), "\n" (10), "\t" (9), and "\f" (12).
where:
 - (13) is the ASCII code for a carriage return (CR)
 - (10) is the ASCII code for a line feed (LF)
 - (9) is the ASCII code for a horizontal tab (HT)
 - (12) is the ASCII code for a form feed (FF)
2. Escaping ASCII code using "#"— for example, "#13".
3. Escaping ASCII hexadecimal code using "#x"— for example, "#x0D".
4. End-of-file delimiter "\eof" which maps to a virtual end-of-file character—This delimiter can only be used once. No other fields can follow once it has been used.

[Example B-19](#) provides several examples of delimiter encoding styles.

Example B-19 *Delimiter Encoding Styles*

```
<termstring endchar="#x2C"/>
<termstring endchar="\n"/>
<limarray id="linearr" contchar="," endchar="\r\n">
  <simplestring/>
</limarray>

<termstring id="FileContents" endchar="\eof"/>
```

The "\r\n" on line 3 of [Example B-19](#) represents a DOS style line break.

D3L Integration with OracleAS InterConnect Adapters

This section provides information on how the D3L files and D3L translation engine are integrated in runtime events and message translations with the OracleAS InterConnect Adapter agent and bridge subcomponents. This section contains these topics:

- [Runtime Initialization](#)
- [Native Format Message to Common View Incoming Message Translations](#)
- [Common View to Native Format Message Outgoing Messages Translations](#)

Runtime Initialization

The OracleAS InterConnect Adapter agent reads `.ini` files (such as `adapter.ini`) at runtime to access each OracleAS InterConnect Adapter's configuration information. The OracleAS InterConnect Adapter bridge initializes itself and the common transport layer with configuration information provided by the OracleAS InterConnect Adapter agent. At the completion of a successful initialization, the OracleAS InterConnect Adapter bridge knows:

- The Oracle Application Server InterConnect application name and its default endpoint (message destination)
- The various Oracle Application Server InterConnect events to be handled by the OracleAS InterConnect Adapter bridge
- The D3L files that describe each of these events
- The D3L files that are accessible and valid. If a file is invalid, the OracleAS InterConnect Adapter cannot start.

See Also:

- [Chapter 1, "Getting Started with OracleAS InterConnect"](#) for additional information on bridges and agents
- [Chapter 2, "Using iStudio"](#) for additional information about events

Native Format Message to Common View Incoming Message Translations

When the OracleAS InterConnect Adapter common transport layer detects an incoming message from an application, it receives the message in its native format. The common transport layer passes it to the OracleAS InterConnect Adapter bridge. The bridge:

- Uses the D3L translation engine to translate the native format message into an application view (an Oracle Application Server InterConnect message object) using the D3L file with which it is configured.
- Raises an application view event

The agent transforms the application view event into a common view event and passes it on for further routing and processing. [Table B-1](#) describes the data flow sequence if D3L message header attributes are used.

Table B-1 Message Header Attributes

If The...	Then...
Name/value pair message header attributes are used	<p>If the incoming native event:</p> <ul style="list-style-type: none"> ■ Contains transport message headers/properties (made available to the bridge by the transport layer) ■ Has a transport message header parameter name (for example, <code>D3L-Header</code>) that matches the <code>header</code> attribute of the <code><message></code> element in the D3L file (<code>header="D3L-Header"</code>) ■ Has a transport message header value (for example, <code>D3L-Header:price</code>) that matches the <code>value</code> attribute of the <code><message></code> element in the D3L file (<code>value="price"</code>) <p>the bridge assumes the matching D3L describes the incoming native event. Any conflicting header and value settings are detected and rejected by the bridge at initialization time. OracleAS InterConnect Adapter operations are logged by Oracle Application Server InterConnect logging and tracing APIs for debugging, performance analysis, and business intelligence functions.</p> <p>See Also: Figure B-2 on page B-7 for complete syntax examples</p>

Table B-1 Message Header Attributes

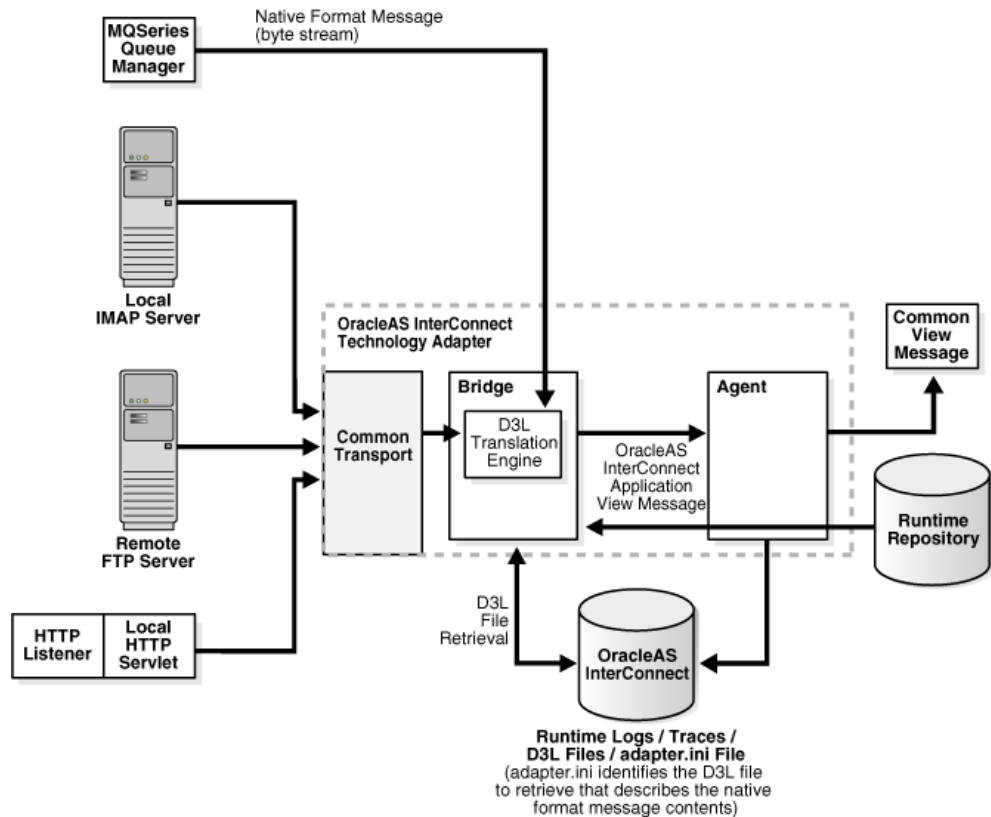
If The...	Then...
Magic value message header attribute is used	<p data-bbox="511 302 711 322">If a magic value is:</p> <ul style="list-style-type: none"> <li data-bbox="511 343 1011 364">■ Specified for the D3L file (length = <i>n</i> bytes) <li data-bbox="511 385 1310 482">■ The first <i>n</i> bytes of payload data in an incoming native event (for example, *UPDATE_PRICE) match the magic attribute of the <message> element in the D3L file (for example, magic="*UPDATE_PRICE"> <p data-bbox="511 503 1310 631">the bridge assumes the native event must be processed using the matching D3L. If multiple D3Ls specify magic values that may match the same native event, the bridge randomly picks a D3L; this can lead to undesirable bridge behavior (the resulting application view event raised may not be the correct one).</p> <p data-bbox="511 652 1182 673">See Also: Figure B-3 on page B-8 for complete syntax examples</p>

Figure B-4 shows the data flow sequence.

See Also:

- ["Message Header Attributes"](#) on page B-6
- ["Task 4: Configure a Native Format Message with a D3L File"](#) on page B-38

Figure B-4 Native Format Message to Common View Incoming Messages



Common View to Native Format Message Outgoing Messages Translations

When a common view event is raised, the OracleAS InterConnect Adapter agent subscribing to the event:

- Receives the associated message
- Transforms it to an Oracle Application Server InterConnect message object
- Hands it to the OracleAS InterConnect Adapter bridge (as an application view event)

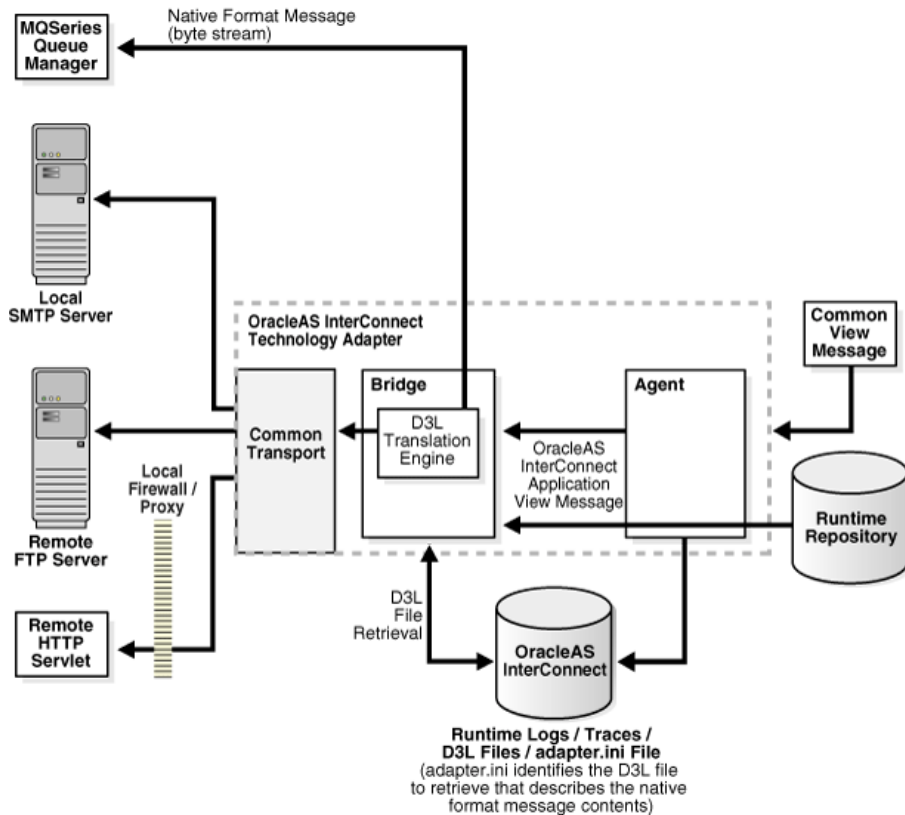
The OracleAS InterConnect Adapter bridge queries the metadata associated with the event to determine:

- The D3L file for the D3L translation engine to use to translate the application view event into a native format message
- The application to which to send the native format message event. There are two levels of rules to determine the application endpoint (the destination) of an Oracle Application Server InterConnect event:
 - If the event contains metadata that specifies an endpoint (note that here, the metadata itself names the endpoint; the content of the event is not searched), the bridge uses this endpoint for this event. With the exception of the MQ Series adapter, all OracleAS InterConnect Adapters follow this rule.
 - If the message metadata did not specify an endpoint, the bridge uses its default endpoint (that was specified at installation time, added to the `adapter.ini` file, and made available to the bridge during initialization).

All OracleAS InterConnect Adapter operations are logged using the Oracle Application Server InterConnect logging and tracing APIs for debugging, performance analysis, and other business intelligence functions.

Figure B-5 shows the data flow sequence.

Figure B-5 Common View to Native Format Message Outgoing Messages



Installing D3L

D3L is automatically installed with Oracle Application Server InterConnect. See *Oracle Application Server InterConnect Installation Guide* for information on installing Oracle Application Server InterConnect.

See Also: Your OracleAS InterConnect Adapter documentation for instructions on installing and configuring the appropriate adapter to use with D3L

Configuring D3L

After installation, perform the following tasks to configure D3L:

- [Task 1: Configure D3L with iStudio](#)
- [Task 2: Create a Native Format Message](#)
- [Task 3: Create a D3L File Describing the Native Format Message](#)
- [Task 4: Configure a Native Format Message with a D3L File](#)
- [Task 5: Configure D3L with OracleAS InterConnect Adapters](#)
- [Task 6: Import a D3L File in iStudio](#)
- [Task 7: Define Metadata Properties with Each Event \(Optional\)](#)

Task 1: Configure D3L with iStudio

You must define D3L in the `browsers.init` file. This enables you to import D3L files as attributes and select D3L as the message type in iStudio.

To integrate D3L with iStudio:

1. Use a text editor to open the `ORACLE_HOME\oai\9.0.4\iStudio\browsers.init` file.
2. Add the following information:
`D3L;oracle.oai.agent.adapter.technology.D3LBrowser;`
3. Save your changes and exit the file.

See Also: ["Task 6: Import a D3L File in iStudio"](#) on page B-40 for the locations of D3L functionality in iStudio

Task 2: Create a Native Format Message

1. Create a native format message. The native format is typically predefined by your third-party application. For example, this native format message updates the salary of employee number 33201 to 55000:

```
*UPDATE_EMPLOYEE_SALARY* 33201 |55000|
```

Where...	Is...
<action>	UPDATE_EMPLOYEE_SALARY
<EmployeeID>	33201
<newSalary>	55000

Task 3: Create a D3L File Describing the Native Format Message

1. Use a text editor to create a D3L file (for example, named `updemp.xml`) that describes the format of the native message. For example, the following D3L file describes the contents of the native format message created in "[Task 2: Create a Native Format Message](#)".

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message name="modify" object="Employee" type="modifyCommand"
  header="D3L-Header" value="employee">
  <struct id="modifyCommand">
    <field name="action"><limstring delimiter="*"/></field>
    <field name="EmployeeID"><limstring delimiter=" "/></field>
    <field name="newSalary"><limstring delimiter="|"/></field>
  </struct>
</message>
```

2. Store the D3L file (`updemp.xml`) in the `ORACLE_HOME\oai\9.0.4\adapters\application` directory (for direct access at deployment time).

See Also: The following sections for additional examples of D3L files:

- [Figure B-1](#) on page B-5
- [Example B-20](#) on page B-45
- ["Additional D3L Sample Files"](#) on page B-68

Task 4: Configure a Native Format Message with a D3L File

Configure a native format message with the correct D3L file. This enables the D3L translation engine to use the correct D3L file to verify native format message contents. For example, the D3L file created in "[Task 3: Create a D3L File Describing the Native Format Message](#)" on page B-38 includes settings for name/value pair message header attributes:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message name="modify" object="Employee" type="modifyCommand"
  header="D3L-Header" value="employee">
```

These settings can match with the transport message header `D3L-Header` parameter name and `employee` value of a native format message:


```
POST /oai/servlet/transportServlet HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: acme.com:8888
Content-Length: 38
D3L-Header: employee
```

See Also:

- [Figure B-2](#) on page B-7
- [Table B-1](#) on page B-32

Task 5: Configure D3L with OracleAS InterConnect Adapters

Define D3L in the following places in the `adapter.ini` file. The `adapter.ini` file is read by the appropriate OracleAS InterConnect Adapter at startup.

1. Use a text editor to open the `ORACLE_HOME\oai\9.0.4\adapters\application\adapter.ini` file.

where *application* is the name of your application and the value of the `application` parameter in the `adapter.ini` file.

2. Ensure that parameter `ota.type` is set to the following value:

```
ota.type=D3L
```

This defines D3L as the message type for the OracleAS InterConnect Adapter to handle for both incoming and outgoing messages.

3. Add the following line to define the D3L files for the bridge and D3L translation engine to use:

```
ota.d3ls=updemp.xml
```

where `updemp.xml` is an example of the D3L file created in "[Task 3: Create a D3L File Describing the Native Format Message](#)" on page B-38. Each event handled by the bridge must have its own D3L file. Whenever a new D3L file is imported in iStudio for use by an application, this parameter must be updated and the OracleAS InterConnect Adapter restarted.

4. Save your changes and exit the file.

Task 6: Import a D3L File in iStudio

iStudio enables you to import a D3L file for use with the following Oracle Application Server InterConnect features:

- Common data types
- Application data types
- Published/subscribed events
- Invoked/implemented procedures
- Business object events and procedures

When a D3L file is associated with Oracle Application Server InterConnect common data types, application data types, events, or procedures, an iStudio OracleAS InterConnect Adapter browser plug-in verifies that the file conforms to the syntax and semantics of D3L. [Table B-2](#) identifies the tasks and locations in iStudio where you can import a D3L file as an attribute and select D3L as a message type. Documentation references that describe how to perform these tasks are also provided.

Table B-2 D3L Functionality in iStudio

For this D3L Functionality...	Do This...
Common Data Type Tasks:	
<ul style="list-style-type: none"> ■ Create a common data type that imports a D3L file as an attribute 	See " Creating Common Data Types " on page 3-3
Application Data Types Tasks:	
<ul style="list-style-type: none"> ■ Creating an application data type that imports a D3L file as an attribute 	Select File > New > Application Data Type from the iStudio menu
Event Tasks:	
<ul style="list-style-type: none"> ■ Create an event that imports a D3L file as an attribute 	See " Creating Events " on page 4-3
<ul style="list-style-type: none"> ■ Publish an event that uses D3L as the message type and imports a D3L file as an attribute 	See " Publishing an Event " on page 4-5
<ul style="list-style-type: none"> ■ Subscribe to an event that uses D3L as the message type and imports a D3L file as an attribute 	See " Subscribing to an Event " on page 4-12
Procedure Tasks:	
<ul style="list-style-type: none"> ■ Create a procedure that imports a D3L file as an attribute 	See " Creating a Procedure " on page 5-3

Table B–2 D3L Functionality in iStudio

For this D3L Functionality...	Do This...
<ul style="list-style-type: none"> Invoke a procedure that uses D3L as the message type and imports a D3L file as an attribute 	See " Invoking a Procedure " on page 5-5
<ul style="list-style-type: none"> Implement a procedure that uses D3L as the message type and imports a D3L file as an attribute 	See " Implementing a Procedure " on page 5-12

Note: D3L functionality with procedures in iStudio is only available with the MQ Series adapter.

Task 7: Define Metadata Properties with Each Event (Optional)

You can associate metadata with each event in iStudio by selecting the Modify Fields buttons on the Subscribe Wizard - Define Application View dialog. The Modify Fields button appears after you select D3L as the Message Type on the preceding Subscribe Wizard - Select an Event dialog. Such metadata is used for content-based routing, for example, of events at runtime.

The following application view event metadata is used by the OracleAS InterConnect Adapters. The property name is prefixed by `ota` to minimize namespace conflicts with user-defined metadata on application view events. The property name is considered a keyword/reserved name, and is used by both iStudio and the bridge (and must be kept consistent between these two components).

Property Name	Property Value Type	Explanation
<code>ota.d3lPath</code>	The D3L filename (string). This is automatically set. Do not modify this property.	The path name (relative or absolute) of the file that contains the D3L guidelines for this event.
<code>ota.isD3L</code>	This value is always true (boolean) and automatically set. Do not modify this property.	A flag indicating that this event is based on D3L.

Property Name	Property Value Type	Explanation
<code>ota.send.endpoint</code>	The endpoint URL (string). This is mandatory. For example: <code>http://foo.com/servlet/test</code>	The actual endpoint to which this message is sent. This setting must match the type of OracleAS InterConnect Adapter that subscribes to the event.
<code>http.sender.*</code> <code>file.sender.*</code>	See Chapter 2 of the appropriate OracleAS InterConnect Adapter documentation for the adapter being defined in the <code>ota.send.endpoint</code> parameter URL ¹). This is optional. For example: <code>http.sender.timeout=5000</code>	The properties define the transport layer configuration.

¹ The SMTP adapter does not define any `smtp.sender` properties. The MQ Series adapter does not support multiple sending endpoints in this release.

See Also:

- ["Subscribing to an Event"](#) on page 4-12
 - ["D3L Use Case"](#) on page B-42
-
-

D3L Use Case

This section contains these topics:

- [D3L Use Case Overview](#)
- [Creating Data Type Definitions for Application Views](#)
- [Configuring the `aqapp_pub` and `fileapp_sub` Applications in iStudio](#)
- [Installing the Advanced Queuing and FTP Adapters](#)
- [Running the D3L Use Case](#)
- [Using Other Adapters and XML Mode](#)

D3L Use Case Overview

This use case provides an example of a minimal Oracle Application Server InterConnect configuration and setup that uses D3L. This use case involves two applications using OracleAS InterConnect Adapters:

- `aqapp_pub`, which is based on the Advanced Queuing adapter
- `fileapp_sub`, which is based on the FTP adapter running in D3L mode

These applications use a business object called `Employee`, which has one defined event called `newEmployee`.

`aqapp_pub` publishes the `newEmployee` event, while `fileapp_sub` subscribes to it. [Table B-3](#) describes the attributes (message structure) of the `newEmployee` event:

Table B-3 Common View Attributes

Attribute Name	Attribute Type
<code>EmpName</code>	String
<code>EmpDept</code>	Integer
<code>EmpHiredate</code>	Date
<code>EmpSalary</code>	Double

All these attributes are scalar (that is, there are no arrays). This message structure represents the common view of the `newEmployee` event. For simplicity, the application views for the two applications have the exact same structure as the common view.

In "[Creating Data Type Definitions for Application Views](#)", a DTD file and a D3L file are created that match the common view attributes shown in [Table B-3](#) on page B-43. These files are used when the application views for the two applications are defined.

Creating Data Type Definitions for Application Views

You must create data type definitions for the two application views.

This section contains these topics:

- [Task 1: Create a DTD File for the Advanced Queuing Adapter](#)
- [Task 2: Create a D3L File for the FTP Adapter](#)

Note: This use case assumes that you have already installed and configured Oracle Application Server InterConnect and iStudio.

Task 1: Create a DTD File for the Advanced Queuing Adapter

The application view for the Advanced Queuing adapter must be defined through a DTD. The DTD enables the Advanced Queuing adapter to translate a received XML (text) document into a runtime application view (Java) object. The agent component of the Advanced Queuing adapter can then transform it to a common view object before routing it to any application subscribers. A DTD is registered with (imported to) the application while defining, for example, a publication in iStudio.

1. Create a DTD file that matches the common view message structure shown in [Table B-3](#) on page B-43:

```
<!ELEMENT NewEmpRec (EmpName, EmpDept, EmpHiredate, EmpSalary)>
<!ELEMENT EmpName    (#PCDATA)>
<!ELEMENT EmpDept    (#PCDATA)>
<!ELEMENT EmpHiredate (#PCDATA)>
<!ELEMENT EmpSalary  (#PCDATA)>
```

2. Save this DTD in a text file named `newemp.dtd`. This file can be saved to any location.

Task 2: Create a D3L File for the FTP Adapter

When running in D3L mode, the FTP adapter must have its application view defined by a D3L (XML) file. The D3L file enables a bidirectional translation between the internal runtime application view (Java) object representation and an external binary/native format message representation. The D3L file is registered with (imported to) the application while defining, for example, a subscription in iStudio.

Assume the external binary native format message of the `newEmployee` event is as follows:

```
message ::= <empname> <empdept> <emphiredate> <empsalary>
empname ::= char[20] // left adjusted string, 20 chars wide, right padded with spaces
empdept ::= byte[2]  // unsigned 2-byte integer, little endian
emphiredate ::= '|' + <month> + <anysep> + <day> + <anysep> + <year> + '|'
empsalary ::= '$' <number> '$'
```

Where...	Is...
<month>, <day>, and <year>	The date format elements MM, DD, and YYYY (all digits)
<anysep>	Any single character
<number>	Any decimal number using the character "." as a decimal separator

1. Create a D3L file that describes the structure that the native format message must follow to communicate with Oracle Application Server InterConnect. The native format message can be expressed/mapped in the D3L XML definition as shown in [Example B-20](#):

Example B-20 D3L Sample File

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message type="NewEmpRec" name="newEmployee" object="Employee">
  <!-- TYPE DECLARATIONS -->
  <!-- string field 20 chars wide with trailing spaces -->
  <padstring id="str20" padchar=" " padstyle="tail" length="20" />
  <!-- unsigned 2-byte integer -->
  <unsigned2 id="uword" endian="little" />
  <!-- date format using pattern MM-DD-YYYY enclosed by '|' -->
  <date id="date" format="MMDDYYYY"><limstring delimiter="|" />
  </date>
  <!-- decimal number format enclosed by '$' -->
  <number id="number"><limstring delimiter="$" /></number>
  <!-- MESSAGE STRUCTURE -->
  <struct id="NewEmpRec">
    <field name="EmpName"> <typeref type="str20" /> </field>
    <field name="EmpDept"> <typeref type="uword" /> </field>
    <field name="EmpHiredate"> <typeref type="date" /> </field>
    <field name="EmpSalary"> <typeref type="number" /> </field>
  </struct>
</message>
```

2. Save the D3L definition in [Example B-20](#) to a file called newemp.xml.
3. Include a copy of this file on both the host computer where Oracle Application Server InterConnect is installed and on the Windows computer where iStudio is installed.

Note: `newemp.xml` is also copied to the FTP adapter application directory in "[Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory](#)" on page B-61.

The following example shows a native format message that can be translated by the `newemp.xml` D3L file (The ? character means nonprintable):

Pos	Bytes (in hexadecimal)	Characters
0000000	4a6f 686e 2044 6f65 2020 2020 2020 2020	John Doe
0000020	2020 2020 4000 7c31 322f 3134 2f32 3030	@? 12/14/200
0000040	317c 2435 3432 3230 2e37 3524	1 \$54220.75\$

Where...	Is...
EmpName	John Doe
EmpDept	64 (hex: 0x40)
EmpHiredate	12/14/2001
EmpSalary	54220.75

In "[Configuring the aqapp_pub and fileapp_sub Applications in iStudio](#)" on page B-46, you complete all the steps necessary in iStudio, including defining the common view, defining the application creation, and so on.

Configuring the aqapp_pub and fileapp_sub Applications in iStudio

This section describes the tasks to complete in iStudio.

This section contains these topics:

- [Task 1: Create a New Workspace and New Project](#)
- [Task 2: Create the Employee Business Object](#)
- [Task 3: Create the newEmployee Event](#)
- [Task 4: Create the aqapp_pub Application](#)
- [Task 5: Enable the aqapp_pub Application to Publish the newEmployee Event](#)
- [Task 6: Define the Application Queue for the aqapp_pub Application](#)
- [Task 7: Create the fileapp_sub Application](#)
- [Task 8: Enable the fileapp_sub Application to Subscribe to the newEmployee Event](#)

Task 1: Create a New Workspace and New Project

1. Start iStudio from the Start menu.

When iStudio starts, the last used workspace is automatically loaded. For this use case, define a new workspace and new Project.

2. Select File > New Workspace.
3. Enter `d3l_tests` for the Workspace Name and click OK.
4. Select File > New Project.
5. Enter `d3l_test_ftp` for the Project Name and click OK.
6. Enter the following values in the Hub Information dialog:

For...	Enter...
Hub database username	<code>oaihub</code>
Hub database password	<code>oaihub</code> (the default)
Hub database URL	<code>hubDB-host:hubDB-port:hubDB-SID</code> For example: <code>dlsun10:1521:V904</code>

Task 2: Create the Employee Business Object

1. Select File > New > Business Object.
2. Enter `Employee` for the Business Object name and click OK.

Note: The `Employee` Business Object name matches with the value for the object attribute of the `<message>` element in the D3L file created in "[Task 2: Create a D3L File for the FTP Adapter](#)" on page B-44.

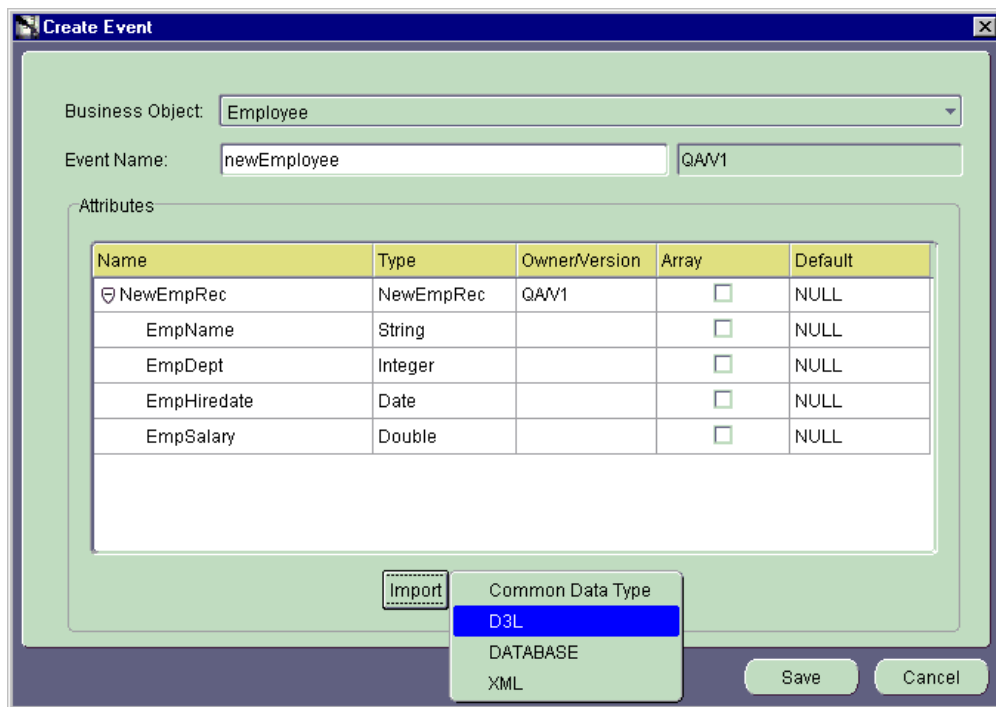
Task 3: Create the newEmployee Event

Define the newEmployee event as described in "D3L Use Case Overview" on page B-43. Define the (common view) attributes of the event by importing the newemp.xml D3L file defined in "Task 2: Create a D3L File for the FTP Adapter" on page B-44. This D3L file defines the same data types as used by the common view. (See Table B-3 on page B-43.)

1. Select File > New > Event.
2. Select Employee in the Business Object drop down list.
3. Enter newEmployee in the Event Name field.
4. Click Import.
5. Select D3L from the list that appears.
6. Locate and select the newemp.xml D3L file created in "Task 2: Create a D3L File for the FTP Adapter" on page B-44. The contents of newemp.xml display in the Attributes fields of the Create Event dialog. If you receive an error while importing, check if the contents of the newemp.xml file on your iStudio computer are identical to the text shown in Example B-20 on page B-45.

Note: The newEmployee Event Name matches with the value for the name attribute of the <message> element in the D3L file created in "Task 2: Create a D3L File for the FTP Adapter" on page B-44.

The Create Event dialog looks as follows:



7. Click Save.

See Also: ["Creating Events"](#) on page 4-3

Task 4: Create the aqapp_pub Application

Now create the aqapp_pub application, which publishes the defined event Employee.newEmployee.

1. Select File > New > Application.
2. Enter aqapp_pub for the Application Name and click OK.

Task 5: Enable the aqapp_pub Application to Publish the newEmployee Event

Use the Publish Wizard to publish the newEmployee event.

This section contains these topics:

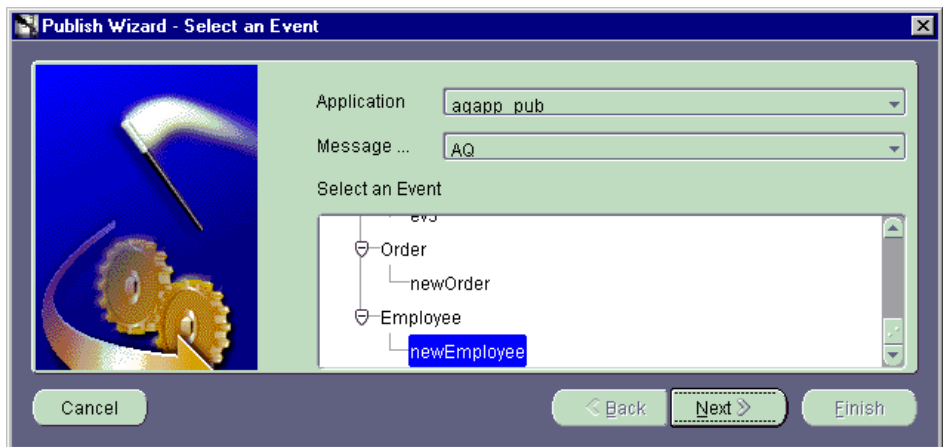
- [Select the Event to Publish](#)
- [Define the Application View](#)
- [Define the Application View to Common View Mapping](#)

Select the Event to Publish Select the event to publish with the Publish Wizard.

1. Select Event > Publish Event.

The Publish Wizard - Select an Event dialog appears.

2. Select aqapp_pub from the Application drop down list.
3. Select AQ from the Message Type drop down list. This choice means that the aqapp_pub application is based on the Advanced Queuing adapter.
4. Click the newEmployee event in the Select an Event tree, which is a child of the Employee business object.

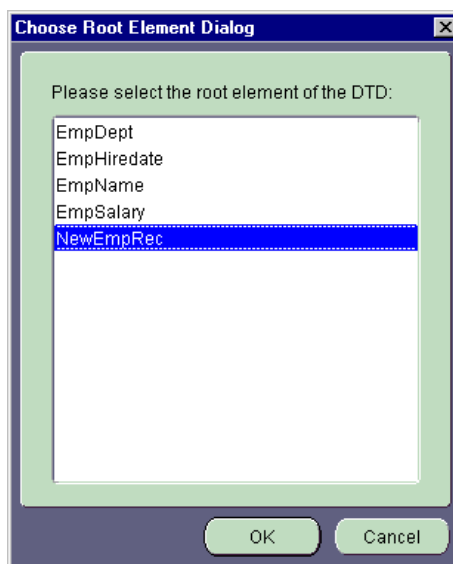


5. Click Next.

The Publish Wizard - Define Application View dialog appears.

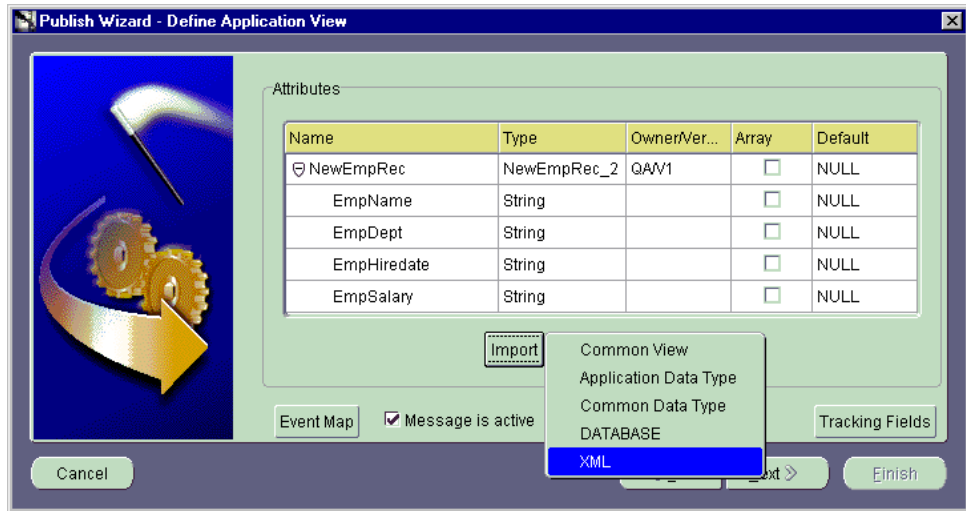
Define the Application View Define the application view for the Advanced Queuing adapter-based application `aqapp_pub` in this dialog. This view was defined in "[Task 1: Create a DTD File for the Advanced Queuing Adapter](#)" on page B-44 as an XML DTD, which is a requirement of the Advanced Queuing adapter. Import this DTD to define the application view.

1. Click the Import button.
2. Select XML from the list that appears.
3. Locate and select the `newemp.dtd` file, which you created in "[Task 1: Create a DTD File for the Advanced Queuing Adapter](#)" on page B-44.
4. Select `NewEmpRec` in the Choose Root Element dialog.



5. Click OK.

The Publish Wizard - Define Application View dialog looks as follows:



6. Click Next.

The Publish Wizard - Define Mapping dialog appears.

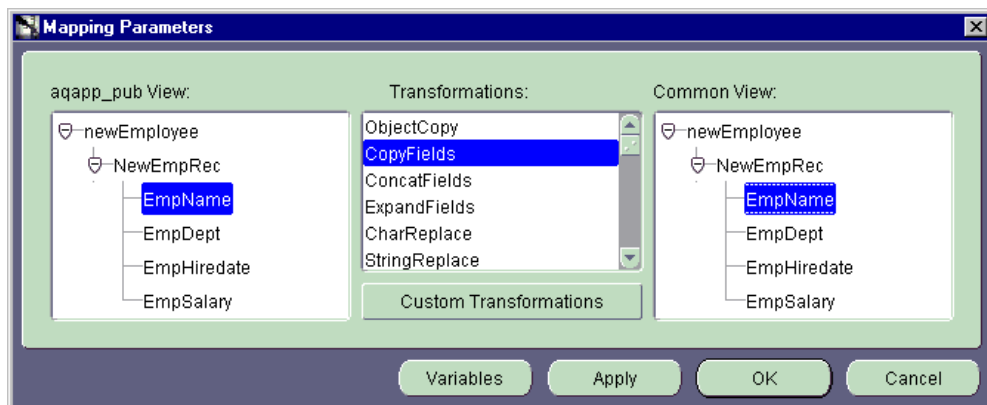
Define the Application View to Common View Mapping Define the application view to common view mapping on this dialog.

1. Click the New button.

The Mapping Parameters dialog appears.

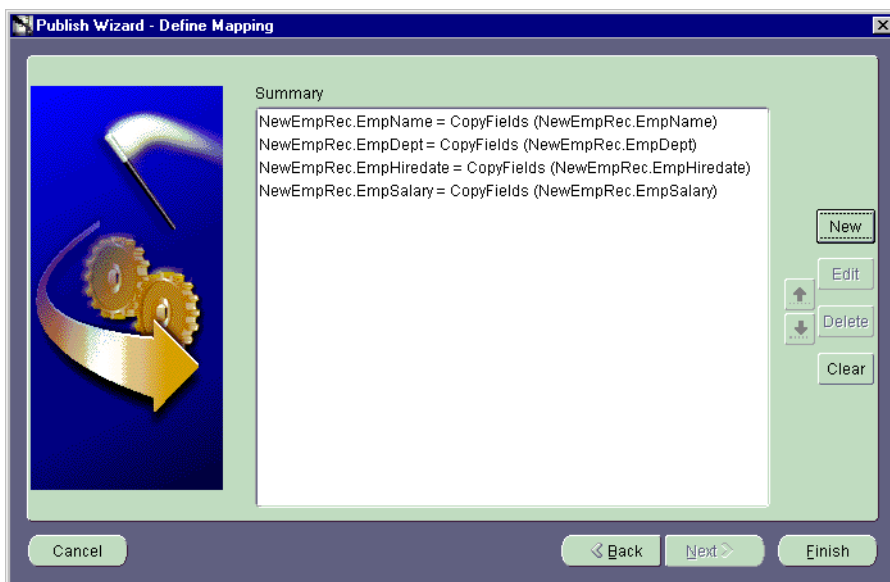
2. Expand newEmployee and NewEmpRec (clicking the '+') in the aqapp_pub View pane.
3. Expand newEmployee and NewEmpRec (clicking the '+') in the Common View pane.
4. Click the EmpName attribute in both panes.
5. Select CopyFields in the Transformations list.

The Mapping Parameters dialog appears as follows:



6. Click OK.
7. Repeat Steps 4 through 6 for the remaining attributes EmpDept, EmpHiredate, and EmpSalary.

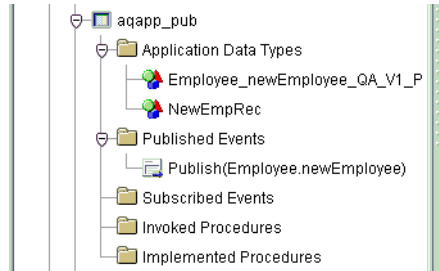
When complete, the Publish Wizard - Define Mapping dialog appears as follows:



There is one line for each attribute.

8. Click Finish.

The Publication for application `aqapp_pub` is complete. The navigation tree pane on the left hand side of iStudio shows the following structure for the `aqapp_pub` application:



Task 6: Define the Application Queue for the `aqapp_pub` Application

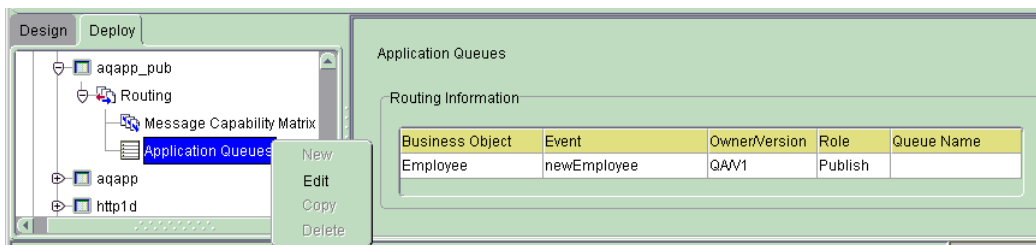
Since the `aqapp_pub` application publishes the `newEmployee` event and is based on the Advanced Queuing adapter, you must define the (Oracle Advanced Queuing) queue from which the Advanced Queuing adapter reads the event. When an XML message, which complies with the DTD defined in "[Task 1: Create a DTD File for the Advanced Queuing Adapter](#)" on page B-44, is enqueued onto the outbound queue, the Advanced Queuing adapter:

- Picks the message up
- Translates the message to an application view event
- Passes the message to the adapter agent for further transformation to the common view representation

The following steps describe how to choose the queue name. The queue does not have to exist physically at this point, as you create it in a later step. (See section "[Task 2: Create the Application Queue AQAPP_NEWEMP](#)" on page B-59.)

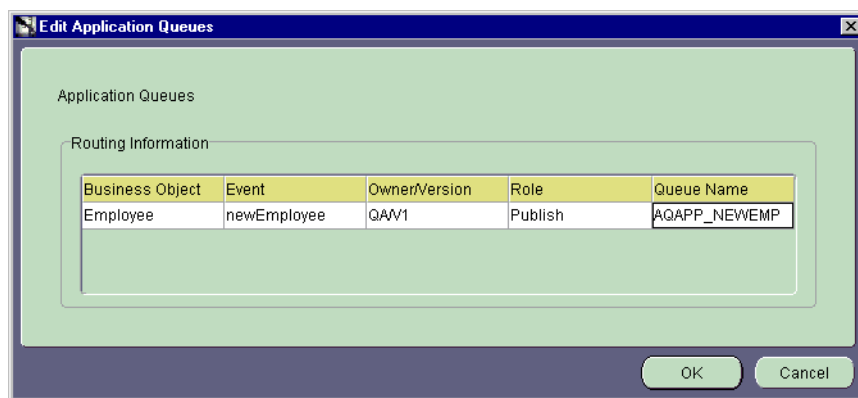
1. Click the Deploy navigation tab on top of the iStudio navigation tree.
2. Expand the Applications node.
3. Expand the `aqapp_pub` node.
4. Expand the Routing node.
5. Right click the Application Queues node.

6. Select the Edit option from the list that appears.



The Edit Application Queues dialog appears.

7. Click in the empty field under the Queue Name column header, and enter the chosen queue name, for example, AQAPP_NEWEMP:



8. Click OK.

Task 7: Create the fileapp_sub Application

Create the fileapp_sub application to subscribe to the defined event Employee.newEmployee (which is published by aqapp_pub).

1. Select File > New > Application.
2. Enter fileapp_sub for the Application Name and click OK.

Task 8: Enable the fileapp_sub Application to Subscribe to the newEmployee Event

Use the Subscribe Wizard to subscribe to the newEmployee event.

This section contains these topics:

- [Select the Event to which to Subscribe](#)
- [Define the Application View](#)
- [Define the Application View to Common View Mapping](#)

Select the Event to which to Subscribe Select the event to which to subscribe with the Subscribe Wizard.

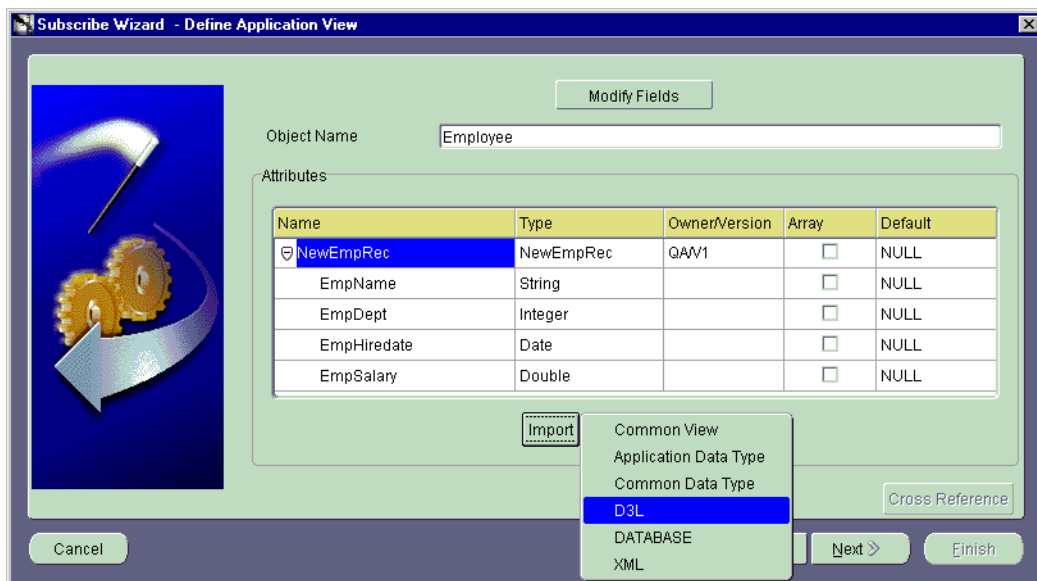
1. Select Event > Subscribe Event.
The Subscribe Wizard - Select an Event dialog appears.
2. Select fileapp_sub from the Application drop down list.
3. Select D3L from the Message Type drop down list.
4. Click newEmployee (under Employee) in the Select an Event tree.
5. Click Next.

The Subscribe Wizard - Define Application View dialog appears.

Define the Application View Define the application view for the FTP adapter-based application fileapp_sub in this dialog. This view was defined in "[Task 2: Create a D3L File for the FTP Adapter](#)" on page B-44 as a D3L file. This is a requirement of any OracleAS InterConnect Adapter operating in D3L mode. Import this D3L file to define the application view.

1. Enter Employee as the business object name in the Object Name input field.
2. Click the Import button.
3. Select D3L from the list that appears.
4. Locate and select the newemp.xml file, which you saved in "[Task 2: Create a D3L File for the FTP Adapter](#)" on page B-44.

The contents of `newemp.xml` display in the Attributes fields:



5. Click Next.

The Publish Wizard - Define Mapping dialog appears.

Define the Application View to Common View Mapping Define the application view to common view mapping in this dialog.

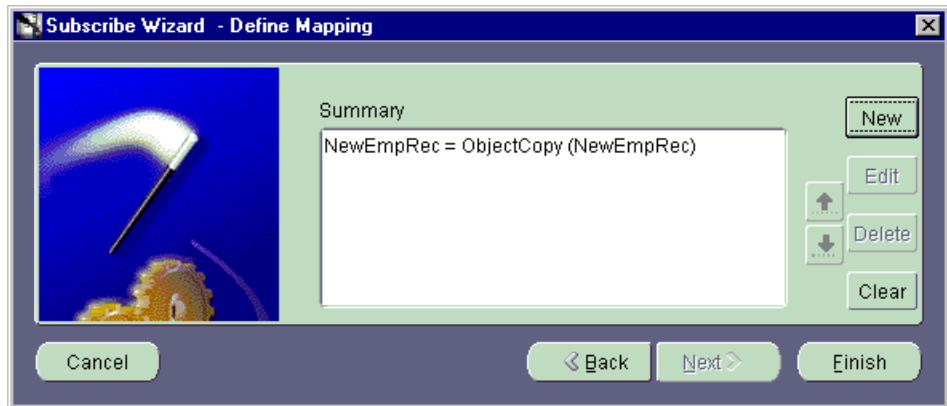
1. Click the New button.

The Mapping Parameters dialog appears.

2. Expand `newEmployee` (clicking the '+') in the Common View pane.
3. Expand `newEmployee` (clicking the '+') in the `fileapp_sub` View pane.
4. Click the `NewEmpRec` node in both panes.
5. Select `ObjectCopy` in the Transformations list and click OK.

Note: You can choose `ObjectCopy` here because the common view and application view are based on the same D3L file.

The Subscribe Wizard - Define Mapping dialog appears as follows:



6. Click Finish.

This completes the necessary setup steps in iStudio.

Installing the Advanced Queuing and FTP Adapters

Now that iStudio setup is complete, you must install one instance of each of the two adapter types. This section contains these topics:

- [Task 1: Install the Advanced Queuing Adapter for Application aqapp_pub](#)
- [Task 2: Create the Application Queue AQAPP_NEWEMP](#)
- [Task 3: Install the FTP Adapter for Application fileapp_sub](#)
- [Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory](#)
- [Task 5: Set the D3L file and Payload Type in the adapter.ini Adapter Initialization File](#)

Task 1: Install the Advanced Queuing Adapter for Application aqapp_pub

1. See "Advanced Queuing Adapter Installation" in Chapter 2 of the *Oracle Application Server InterConnect Adapter for AQ Installation and User's Guide* for installation instructions. During installation, enter the following specific values when prompted:
 - a. Enter aqapp_pub in the Application Name field of the Oracle Application Server InterConnect AQ Adapter Configuration dialog.
 - b. Enter the database connection information to connect to the database instance on the Application Spoke Database page. The AQAPP_NEWEMP application queue defined in "[Task 6: Define the Application Queue for the aqapp_pub Application](#)" on page B-54 is created here.
 - c. Enter the database username and password of the account and schema on the Spoke Application Database AQ Username dialog, which owns the Application Queue (AQAPP_NEWEMP). Select the schema name aqapp and the password aqapp. Leave the Consumer Name field blank, as you are creating the AQAPP_NEWEMP queue as a single consumer queue.
2. Complete adapter installation by providing appropriate responses when prompted.

When installation is complete, the new adapter instance is located in the following directory:

Platform	Directory
Windows	%ORACLE_HOME%\oai\9.0.4\adapters\aqapp_pub
UNIX	\$ORACLE_HOME/oai/9.0.4/adapters/aqapp_pub

Task 2: Create the Application Queue AQAPP_NEWEMP

To create the Advanced Queuing AQAPP_NEWEMP application queue, you must first create the queue table, create the queue, and start the queue.

1. Ensure that the database user issuing the commands in this section has been granted these roles:


```
RESOURCE, CONNECT, AQ_ADMINISTRATOR_ROLE
```
2. Use SQL*Plus to log in to the database account specified in Steps 1b and 1c of "[Task 1: Install the Advanced Queuing Adapter for Application aqapp_pub](#)" on page B-59.

3. Create the queue table using the same name as the application queue:

```
SQL> EXECUTE dbms_aqadm.create_queue_table('AQAPP_NEWEMP', 'RAW');
```

4. Create the queue:

```
SQL> EXECUTE dbms_aqadm.create_queue('AQAPP_NEWEMP', 'AQAPP_NEWEMP');
```

5. Start the queue:

```
SQL> EXECUTE dbms_aqadm.start_queue('AQAPP_NEWEMP');
```

Task 3: Install the FTP Adapter for Application `fileapp_sub`

1. See "FTP Adapter Installation" in Chapter 2 of the *Oracle Application Server InterConnect Adapter for FTP Installation and User's Guide* for installation instructions. During installation, enter the following specific values when prompted:

- a. Enter `fileapp_sub` in the Application Name field of the Oracle Application Server InterConnect FTP Adapter Configuration dialog.
- b. Enter the following value in the URL field of the Oracle Application Server InterConnect FTP Adapter Configuration Configure receiving endpoint information dialog:

```
ftp://localhost/tmp/fileapp_sub/read
```

- c. Enter the following value in the URL field of the Oracle Application Server InterConnect FTP Adapter Configuration Configure sending endpoint information dialog:

```
ftp://localhost/tmp/fileapp_sub/write
```

This places every `newEmployee` message received by the `fileapp_sub` application (by way of its configured subscription created in "[Task 8: Enable the fileapp_sub Application to Subscribe to the newEmployee Event](#)" on page B-56) in the `/tmp/fileapp_sub/write` directory of the computer where the FTP adapter is installed. Ensure that you create these directories with global read and write permissions before starting the `fileapp_sub` application (based on the FTP adapter), for example:

```
$ umask 0
$ mkdir -p /tmp/fileapp_sub/read
$ mkdir -p /tmp/fileapp_sub/write
```

2. Complete adapter installation by providing appropriate responses when prompted.

When installation is complete, the new adapter instance is located in the following directory:

Platform	Directory
Windows	%ORACLE_HOME%\oai\9.0.4\adapters\fileapp_sub
UNIX	\$ORACLE_HOME/oai/9.0.4/adapters/fileapp_sub

Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory

1. Copy the newemp.xml D3L file defined in "[Task 2: Create a D3L File for the FTP Adapter](#)" on page B-44 to the platform-specific directory mentioned in the preceding Step 2 on page B-61.

Task 5: Set the D3L file and Payload Type in the adapter.ini Adapter Initialization File

Set the ota.d3ls and ota.type parameters in the adapter.ini adapter initialization file for the FTP adapter. The adapter.ini file is located in the platform-specific directory mentioned in the preceding Step 2 on page B-61.

1. Use a text editor to set the ota.d3ls parameter to newemp.xml in adapter.ini:

```
ota.d3ls=newemp.xml
```

If the ota.d3ls parameter line already exists in adapter.ini, replace it with this version.

2. Use a text editor to set the ota.type parameter to D3L in adapter.ini:

```
ota.type=D3L
```

Running the D3L Use Case

Now that both the Advanced Queuing adapter instance `aqapp_pub` and the FTP adapter instance `fileapp_sub` have been installed, use both to run the D3L use case.

This section contains these topics:

- [Task 1: Start the Adapters](#)
- [Task 2: Create PL/SQL Code to Trigger the Native `newEmployee` Event](#)
- [Task 3: Trigger the `newEmployee` Event](#)
- [Task 4: Verify Receipt of `newEmployee` Event](#)

Task 1: Start the Adapters

To Start the Adapters on UNIX: Follow these steps to start the adapters on UNIX:

To start the `aqapp_pub` (Advanced Queuing) adapter:

1. Change directories to where the `aqapp_pub` adapter is installed:

```
$ cd $ORACLE_HOME/oai/9.0.4/adapters/aqapp_pub
```

2. Start the adapter as a background process:

```
$ start &
```

To start the `fileapp_sub` (FTP) adapter:

1. Change directories to where the `fileapp_sub` adapter is installed:

```
$ cd $ORACLE_HOME/oai/9.0.4/adapters/fileapp_sub
```

2. Start the adapter as a background process:

```
$ start &
```

To Start the Adapters on Windows: Follow these steps to start the adapters on Windows:

To start the `aqapp_pub` (Advanced Queuing) adapter:

1. Change directories to where the `aqapp_pub` adapter is installed:

```
cd %ORACLE_HOME%\oai\9.0.4\adapters\aqapp_pub
```


2. Start the adapter:

```
start
```

To start the `fileapp_sub` (FTP) adapter:

1. Change directories to where the `fileapp_sub` adapter is installed:

```
cd %ORACLE_HOME%\oai\9.0.4\adapters\fileapp_sub
```

2. Start the adapter:

```
start
```

Note: You can also start adapters from the Windows Control Panel. See your OracleAS InterConnect Adapter documentation for instructions.

Task 2: Create PL/SQL Code to Trigger the Native `newEmployee` Event

The next task generates the native event (that is, triggers the `newEmployee` event). As configured in iStudio, the `aqapp_pub` application publishes the `newEmployee` event. It does so when it sees a new (XML) message on the `AQAPP_NEWEMP` queue that conforms to the DTD defined in "[Task 1: Create a DTD File for the Advanced Queuing Adapter](#)" on page B-44.

To generate the native event, you must enqueue a message on the application queue (`AQAPP_NEWEMP`) for the application `aqapp_pub`. You do this through an anonymous PL/SQL block.

1. Change directories to where the `aqapp_pub` application (of the Advanced Queuing adapter) is installed, for example:

On...	Go To...
UNIX	<code>\$ cd \$ORACLE_HOME/oai/9.0.4/adapters/aqapp_pub</code>
Windows	<code>cd %ORACLE_HOME%\oai\9.0.4\adapters\aqapp_pub</code>

2. Create a file (named `newemp.sql` in this example) with the contents shown in [Example B-21](#):

Example B-21 File newemp.sql

```
DECLARE
  enqueue_options    dbms_aq.enqueue_options_t;
  message_properties dbms_aq.message_properties_t;
  msgid              RAW(16);
  raw_payload        RAW(32767);
  payload            varchar2(2000);
BEGIN
  payload :=
    '<?xml version="1.0" standalone="no"?>
    <NewEmpRec>
      <EmpName>Scott Tiger</EmpName>
      <EmpDept>257</EmpDept>
      <EmpHiredate>05/01/2001</EmpHiredate>
      <EmpSalary>52308.75</EmpSalary>
    </NewEmpRec>';

  raw_payload := utl_raw.cast_to_raw(payload);
  dbms_aq.enqueue(queue_name      => 'AQAPP_NEWEMP',
                  enqueue_options => enqueue_options,
                  message_properties => message_properties,
                  payload          => raw_payload,
                  msgid            => msgid);

  commit;
END;
/
```

Note: The payload variable is being assigned a string value, which contains a valid XML document that conforms to the DTD newemp.dtd defined in ["Task 1: Create a DTD File for the Advanced Queuing Adapter"](#) on page B-44.

Task 3: Trigger the newEmployee Event

Everything is now defined, created, and started. You must now trigger the newEmployee event, which was prepared in "[Task 2: Create PL/SQL Code to Trigger the Native newEmployee Event](#)" on page B-63.

As mentioned earlier, the event is triggered when you place an XML message on the AQAPP_NEWEMP queue, which is what the newemp.sql script does.

Run the PL/SQL script to generate the event.

1. Log in to the database account aqapp where the AQAPP_NEWEMP queue was defined. (See "[Task 2: Create the Application Queue AQAPP_NEWEMP](#)" on page B-59.) For example, assuming no connect string is necessary:

```
sqlplus aqapp/aqapp
```

2. Execute the newemp.sql script:

```
SQL> START newemp.sql
```

The following message appears:

```
PL/SQL procedure successfully completed.
```

3. Exit SQL*Plus:

```
SQL> EXIT
```

Task 4: Verify Receipt of newEmployee Event

After some time (maybe several minutes depending on overall system performance), a file appears in the /tmp/fileapp_sub/write directory, which represents the sending endpoint for the FTP adapter. The file is named after the pattern:

app-name-timestamp

1. Verify that the newEmployee event has been published and received by the fileapp_sub application. On UNIX, for example, perform the following commands:

```
$ cd /tmp/fileapp_sub/write
```

```
$ ls -l
```

```
total 2
```

```
-rw-rw-r-- 1 bstern svrtech 44 Dec 18 15:29 FILEAPP_SUB-1008718194783
```

The contents of the file can be displayed in different formats:

```
$ od -c FILEAPP_SUB-1008718194783
0000000  S  c  o  t  t      T  i  g  e  r
0000020          001 001  |  0  5  /  0  1  /  2  0  0
0000040  1  |  $  5  2  3  0  8  .  7  5  $
```

or

```
$ od -x FILEAPP_SUB-1008718194783
0000000 5363 6f74 7420 5469 6765 7220 2020 2020
0000020 2020 2020 0101 7c30 352f 3031 2f32 3030
0000040 317c 2435 3233 3038 2e37 3524
```

2. Verify that this output corresponds to the D3L definition shown in "[Task 2: Create a D3L File for the FTP Adapter](#)" on page B-44 and the data enqueued by `newemp.sql`.
3. Repeat Step 2 on page B-65 to trigger and generate another event (file). The second time you trigger the event, the new file in the `/write` directory appears much faster (in approximately 3-4 seconds). This is because the adapter allocated and initialized all connections and data structures after processing the first message.
4. You have completed the use case.

Using Other Adapters and XML Mode

This section briefly describes how to use adapters other than the FTP adapter, and how to run them in XML mode instead of D3L mode.

This section contains these topics:

- [Using the HTTP, SMTP, or MQ Series Adapters in D3L Mode](#)
- [Using XML Mode](#)

Using the HTTP, SMTP, or MQ Series Adapters in D3L Mode

Perform the following steps to use the D3L use case with a different OracleAS InterConnect Adapter.

1. Enter another application name that indicates which adapter you are using in "[Task 7: Create the fileapp_sub Application](#)" on page B-55 (for example, `smtppapp_sub`).
2. Specify the parameters needed for the particular adapter in Steps 1b and 1c on page B-60. See the installation documentation for the appropriate OracleAS InterConnect Adapter.
3. In "[Task 4: Verify Receipt of newEmployee Event](#)" on page B-65, the verification process depends entirely on the adapter type, or more specifically, the exact sending endpoint defined.
4. Replace the `fileapp_sub` application name where ever it appears with the new application name.

The remaining steps are the same.

Using XML Mode

Perform the following steps to use XML as the operational mode of the OracleAS InterConnect Adapters.

1. Skip "[Task 2: Create a D3L File for the FTP Adapter](#)" on page B-44.
2. Define the following common view event attributes in Step 4 and Step 5 of "[Task 3: Create the newEmployee Event](#)" on page B-48:
 - a. Manually create a common data type (right click + New) named `NewEmpRec` that has the same attributes as shown in the Create Event dialog on page B-49.
 - b. Import the common data type defined in Step 2a instead of importing a D3L file.
3. Select XML instead of D3L in Step 3 of "[Select the Event to which to Subscribe](#)" on page B-56.
4. Select to import XML and choose the file `newemp.dtd` in Step 4 of "[Define the Application View](#)" on page B-56.
5. Perform Steps 2 through 4 in "[Define the Application View to Common View Mapping](#)" on page B-57 like you did Steps 2 through 7 in "[Define the Application View to Common View Mapping](#)" on page B-52.

6. Skip ["Task 4: Copy the newemp.xml D3L File to the fileapp_sub Adapter Directory"](#) on page B-61 and ["Task 5: Set the D3L file and Payload Type in the adapter.ini Adapter Initialization File"](#) on page B-61.

Note: Replacement steps Steps 2a and 2b assume that you do not have the D3L file. However, as a shortcut, you can still define the common view event attributes as they were performed in Step 6 of ["Task 3: Create the newEmployee Event"](#) on page B-48.

Additional D3L Sample Files and DTD

This section contains these topics:

- [Additional D3L Sample Files](#)
- [D3L DTD](#)

Additional D3L Sample Files

This section provides several D3L sample files. These example files describe how to use the D3L language to define the content of native format messages.

- [Sample File with Structure VehicleRegistration](#)
- [Sample File with Structure Hierarchy PersonRecord](#)
- [Sample File with Structure ProductRecord](#)

Sample File with Structure VehicleRegistration

Sample file `msg-1.xml` represents a structure named `VehicleRegistration`. [Table B-4](#) describes the file fields and [Example B-22](#) shows `msg-1.xml` file contents.

Table B-4 *msg-1.xml File Fields*

Field	Description
SizeWeight	A fixed-length array of four signed, one-byte, little-endian integers, each aligned on two-byte boundaries (implying a one-byte padding between elements of the array)
ProductCode	An unsigned, two-byte, big-endian integer aligned on two-byte boundaries
VIN	An unsigned, eight-byte, big-endian integer aligned on two-byte boundaries
PreviousOwners	A length-prefixed array of dates in the MMDDYYYY format (the length of the array is a signed, one-byte, little-endian integer with a two-byte alignment)
Miles	An unsigned, two-byte, big-endian integer with a two-byte alignment
DateProduced	A single date in the MMDDYYYY format

Example B-22 *Sample File msg-1.xml with Structure EmployeeRegistration*

```
<?xml version="1.0" encoding="US-ASCII"?>
<message type="VehicleRegistration" name="Register" object="Vehicle">
  <date format="MMDDYYYY" id="Date_T">
    <padstring id="FixString10_T" length="10" padchar=' ' padstyle="none" />
  </date>
  <struct id="VehicleRegistration">
    <!-- Width x Length x Height x Weight (inch/lb) -->
    <field name="SizeWeight"><typeref type="ShortArray4_T" /></field>
    <field name="ProductCode"><unsigned2 align="2" endian="big" /></field>
    <field name="VIN"><unsigned8 align="2" endian="big" /></field>
    <field name="PreviousOwners"><typeref type="StringArray_T" /></field>
    <field name="Miles"><unsigned2 align="2" endian="big" /></field>
    <field name="DateProduced"><typeref type="Date_T" /></field>
  </struct>
  <fixarray id="ShortArray4_T" length="4">
    <unsigned2 align="2" endian="little" id="" />
  </fixarray>
  <unsigned1 align="2" endian="little" id="Short_T" />
  <pfxarray id="StringArray_T" length="Short_T">
    <typeref type="FixString10_T" />
  </pfxarray>
</message>
```

The following native format message examples show a hexadecimal and character representation of the same message, which can be parsed by the `msg-1.xml` D3L file:

- Hexadecimal format:

```
0000000 4500 b200 3400 8a0b 30d9 0000 0000 0072
0000020 55ff 0200 4a6f 6e65 732c 502e 2020 536d
0000040 6974 682c 522e 2020 5208 3131 2532 3225
0000060 3139 3939
```

- Character format:

```
0000000 E \0 262 \0 4 \0 212 013 0 331 \0 \0 \0 \0 \0 r
0000020 U 377 002 \0 J o n e s , P . S m
0000040 i t h , R . R \b 1 1 % 2 2 %
0000060 1 9 9 9
```


Sample File with Structure Hierarchy PersonRecord

Sample file `msg-2.xml` demonstrates a structure hierarchy named `PersonRecord`. [Table B-5](#) describes the file fields and [Example B-23](#) shows `msg-2.xml` file contents.

Table B-5 *msg-2.xml File Fields*

Field	Description
Name	A string delimited by a comma
Age	An unsigned, one-byte integer
DOB	A date in MMDDYYYY format, length prefixed by a signed, four-byte integer
Phone	An unsigned, four-byte integer
City	A structure named <code>CityRecord</code> that consists of the following fields: <ul style="list-style-type: none"> ▪ Name A string delimited by * ▪ State A string delimited by * ▪ Country A string delimited by * ▪ Population An unsigned, four-byte integer
State	A structure named <code>StateRecord</code> that consists of the following fields: <ul style="list-style-type: none"> ▪ Name A string delimited by a space ▪ Capital A string delimited by a space ▪ Population An unsigned, four-byte integer

Example B-23 Sample File msg-2.xml with Structure PersonRecord

```

<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message type="PersonRecord">
  <signed4 id="s4" />
  <struct id="CityRecord">
    <field name="Name"><limstring delimiter="*" /></field>
    <field name="State"><limstring delimiter="*" /></field>
    <field name="Country"><limstring delimiter="," /></field>
    <field name="Population"><unsigned4 /></field>
  </struct>

  <struct id="StateRecord">
    <field name="Name"><limstring delimiter=" " /></field>
    <field name="Capital"><limstring delimiter=" " /></field>
    <field name="Population"><unsigned4 /></field>
  </struct>

  <struct id="PersonRecord">
    <field name="Name"><limstring delimiter="," /></field>
    <field name="Age"><unsigned1 /></field>
    <field name="DOB">
      <date format="MMDDYYYY">
        <pfxstring id="dobstr" length="s4" />
      </date>
    </field>
    <field name="Phone"><unsigned4 /></field>
    <field name="City"><typeref type="CityRecord" /></field>
    <field name="State"><typeref type="StateRecord" /></field>
  </struct>
</message>

```

The following is a combined hexadecimal and character representation of a native message, which can be parsed by msg-2.xml:

```

000 2c4a 6f68 6e20 446f 652c 1e00 0000 000a ,John Doe,_. ....
020 3131 2f32 352f 3139 3635 0000 002c a155 11/25/1965... .U
040 2a50 6f72 746c 616e 642a 2a4f 522a 2c55 *Portland**OR*,U
060 5341 2c00 000f 4240 204f 7265 676f 6e20 SA,...B@_Oregon_
100 2053 616c 656d 2000 003d 0900      _Salem_...=..

```

Sample File with Structure ProductRecord

Sample file `msg-3.xml` defines a structure named `ProductRecord`. [Table B-6](#) describes the file fields and [Example B-24](#) shows `msg-3.xml` file contents.

Table B-6 *msg-3.xml File Fields*

Field	Description
Manufacturer	A string delimited by a space
Weight	A single-precision, floating-point number
Widgets	A length-prefixed array of <code>WidgetRecord</code> structures. A <code>WidgetRecord</code> consists of: <ul style="list-style-type: none">■ Name A string delimited by a space■ Color A string delimited by a space■ Weight A single-precision, floating point number

Example B-24 Sample File msg-3.xml with Structure ProductRecord

```
<?xml version="1.0" encoding="US-ASCII"?>

<!DOCTYPE message SYSTEM "d3l.dtd">

<message type="ProductRecord">
  <unsigned1 id="u1" />
  <unsigned2 id="u2" />
  <number id="pfxnum">
    <padstring length="8" padchar=" " padstyle="none" />
  </number>
  <pfxarray id="Unsigned1Tab" length="u1">
    <unsigned1 />
  </pfxarray>
  <pfxarray id="Signed4Tab" length="pfxnum">
    <unsigned4 />
  </pfxarray>
  <pfxarray id="StrTab" length="u1">
    <limstring delimiter=" " />
  </pfxarray>
  <struct id="WidgetRecord">
    <field name="Name"><limstring delimiter=" " /></field>
    <field name="Color"><limstring delimiter=" " /></field>
    <field name="Weight"><float /></field>
  </struct>
  <pfxarray id="WidgetTab" length="u2">
    <typeref type="WidgetRecord" />
  </pfxarray>
  <struct id="ProductRecord">
    <field name="Manufacturer"><limstring delimiter=" " /></field>
    <field name="Weight"><float /></field>
    <field name="Widgets"><typeref type="WidgetTab" /></field>
  </struct>
</message>
```

D3L DTD

Example B-25 shows the DTD to which D3L (XML) files must conform.

Example B-25 D3L DTD

```

<!ENTITY % Name      "CDATA"      >
  <!ENTITY % Number   "NMTOKEN"    >
  <!ENTITY % Comment   "CDATA"      >
  <!ENTITY % DelimiterChar
    "CDATA"
  >
  <!ENTITY % QuotationCharAttribute
    "
    quote %DelimiterChar; #IMPLIED
    "
  >
  <!-- ===== -->
  <!ENTITY % GenericAttributes
    "
    name %Name;          #IMPLIED
    comment %Comment;    #IMPLIED
    id ID                #IMPLIED
    "
  >
  <!ENTITY % FieldAttributes
    "
    name %Name;          #REQUIRED
    comment %Comment;    #IMPLIED
    id ID                #IMPLIED
    "
  >
  <!ENTITY % NonTypeAttributes
    "
    name %Name;          #IMPLIED
    comment %Comment;    #IMPLIED
    "
  >
  <!-- ===== -->
  <!ENTITY % StructAttributes
    "
    %GenericAttributes;
    %QuotationCharAttribute;
    "
  >
  <!-- ===== -->

```

```
<!ENTITY % Align
    "%Number;"
>
<!ENTITY % IntegerSize
    "( 1 | 2 | 4 | 8 )"
>
<!ENTITY % Endian
    "( big | little )"
>
<!ENTITY % IntegerAttributes
    "
    %GenericAttributes;
    endian %Endian;          'big'
    "
>
<!ENTITY % IntegerTypes
    " signed1 | unsigned1
    | signed2 | unsigned2
    | signed4 | unsigned4
    | signed8 | unsigned8
    "
>
<!ENTITY % FloatAttributes
    "
    %GenericAttributes;
    "
>
<!ENTITY % FloatTypes
    " float | double
    "
>
<!-- ===== -->
<!ENTITY % PadStyle
    "( head | tail | none )"
>
<!ENTITY % PadChar
    "CDATA"
>
<!ENTITY % StringAttributes
    "
    %GenericAttributes;
    "
>
<!ENTITY % SimpleStringAttributes
    "
```

```

        %StringAttributes;
    "
>
<!ENTITY % TerminatedStringAttributes
    "
        %StringAttributes;
        endchar      %DelimiterChar; #REQUIRED
    "
>
<!ENTITY % QuotedTerminatedStringAttributes
    "
        %StringAttributes;
        %QuotationCharAttribute;
        endchar      %DelimiterChar; #REQUIRED
    "
>
<!ENTITY % PaddedStringAttributes
    "
        %StringAttributes;
        length       %Number;          #REQUIRED
        padchar      %PadChar;         #REQUIRED
        padstyle     %PadStyle;        #REQUIRED
    "
>
<!ENTITY % PrefixedStringAttributes
    "
        %StringAttributes;
        length       IDREF              #REQUIRED
    "
>
<!ENTITY % DelimitedStringAttributes
    "
        %StringAttributes;
        delimiter    %DelimiterChar; #REQUIRED
    "
>
<!ENTITY % StringTypes
    "padstring | pfxstring | limstring | qtdtermstring | termstring |
simplestring "
>
<!-- ===== -->
<!ENTITY % DateFormat
    "( DDMYY | DDMYYYY | MDDYY | MDDYYYY )"
>
<!ENTITY % DateAttributes

```

```

"
  %GenericAttributes;
  format %DateFormat; #REQUIRED
"
>
<!-- ===== -->
<!ENTITY % NumberAttributes
"
  %GenericAttributes;
"
>
<!-- ===== -->
<!ENTITY % ArrayAttributes
"
  %GenericAttributes;
"
>
<!ENTITY % FixedArrayAttributes
"
  %ArrayAttributes;
  length %Number; #REQUIRED
"
>
<!ENTITY % PrefixedArrayAttributes
"
  %ArrayAttributes;
  length IDREF #REQUIRED
"
>
<!ENTITY % DelimitedArrayAttributes
"
  %ArrayAttributes;
  contchar %DelimiterChar; #REQUIRED
  endchar %DelimiterChar; #REQUIRED
"
>
<!ENTITY % ImplicitArrayAttributes
"
  %ArrayAttributes;
"
>
<!-- ===== -->
<!ENTITY % ScalarElements
" signed1 | unsigned1
| signed2 | unsigned2

```



```

    | signed4 | unsigned4
    | signed8 | unsigned8
    | float   | double
    | date    | number
    | padstring
    | pfxstring
    | limstring
    | termstring
    | qtdtermstring
    | simplestring
    "
>
<!ENTITY % TypeElements
    "%ScalarElements;
    | struct
    | fixarray
    | pfxarray
    | limarray
    | imparray
    "
>
<!-- ===== -->
<!ENTITY % FieldElements
    "%TypeElements;"
>
<!ENTITY % MessageElements
    "%TypeElements;"
>
<!ENTITY % StructElements
    "field | pad"
>
<!ENTITY % ArrayElements
    "%ScalarElements; | struct"
>
<!ENTITY % ImplicitArrayElements
    "%ArrayElements; | limarray"
>
<!-- ===== -->
<!ELEMENT message ( %MessageElements; )* >
  <!ATTLIST message
    name      %Name;          #REQUIRED
    object    CDATA           #REQUIRED
    type      IDREF           #REQUIRED
    comment   %Comment;      #IMPLIED
    id        ID              #IMPLIED

```

```

        header      CDATA          #IMPLIED
        value       CDATA          #IMPLIED
        magic       CDATA          #IMPLIED
        startsat    %Number;       #IMPLIED
        reply       (Y|N)          "N"
        %QuotationCharAttribute;
    >
<!-- ===== -->
<!ELEMENT struct ( %StructElements; )* >
    <!ATTLIST struct
        %StructAttributes;
    >
<!-- ===== -->
<!ELEMENT field ( typeref | %FieldElements; ) >
    <!ATTLIST field
        %FieldAttributes;
    >
<!-- ===== -->
<!ELEMENT signed1 EMPTY >
    <!ATTLIST signed1
        %IntegerAttributes;
        size %IntegerSize; #FIXED "1"
        align %Align;      "1"
    >
<!ELEMENT unsigned1 EMPTY >
    <!ATTLIST unsigned1
        %IntegerAttributes;
        size %IntegerSize; #FIXED "1"
        align %Align;      "1"
    >
<!ELEMENT signed2 EMPTY >
    <!ATTLIST signed2
        %IntegerAttributes;
        size %IntegerSize; #FIXED "2"
        align %Align;      "2"
    >
<!ELEMENT unsigned2 EMPTY >
    <!ATTLIST unsigned2
        %IntegerAttributes;
        size %IntegerSize; #FIXED "2"
        align %Align;      "2"
    >
<!ELEMENT signed4 EMPTY >
    <!ATTLIST signed4
        %IntegerAttributes;

```

```

        size    %IntegerSize;    #FIXED    "4"
        align   %Align;          "4"
    >
<!ELEMENT unsigned4 EMPTY >
    <!ATTLIST unsigned4
        %IntegerAttributes;
        size    %IntegerSize;    #FIXED    "4"
        align   %Align;          "4"
    >
<!ELEMENT signed8 EMPTY >
    <!ATTLIST signed8
        %IntegerAttributes;
        size    %IntegerSize;    #FIXED    "8"
        align   %Align;          "8"
    >
<!ELEMENT unsigned8 EMPTY >
    <!ATTLIST unsigned8
        %IntegerAttributes;
        size    %IntegerSize;    #FIXED    "8"
        align   %Align;          "8"
    >
<!-- ===== -->
<!ELEMENT float EMPTY >
    <!ATTLIST float
        %FloatAttributes;
        align   %Align;          "4"
    >
<!ELEMENT double EMPTY >
    <!ATTLIST double
        %FloatAttributes;
        align   %Align;          "8"
    >
<!-- ===== -->
<!ELEMENT simplestring EMPTY >
    <!ATTLIST simplestring
        %SimpleStringAttributes;
    >
<!ELEMENT qtdtermstring EMPTY >
    <!ATTLIST qtdtermstring
        %QuotedTerminatedStringAttributes;
    >
<!ELEMENT termstring EMPTY >
    <!ATTLIST termstring
        %TerminatedStringAttributes;
    >

```

```

<!ELEMENT padstring EMPTY >
  <!ATTLIST padstring
    %PaddedStringAttributes;
  >
<!ELEMENT pfxstring EMPTY >
  <!ATTLIST pfxstring
    %PrefixedStringAttributes;
  >
<!ELEMENT limstring EMPTY >
  <!ATTLIST limstring
    %DelimitedStringAttributes;
  >
<!-- ===== -->
<!ELEMENT fixarray ( typeref | %ArrayElements; ) >
  <!ATTLIST fixarray
    %FixedArrayAttributes;
  >
<!ELEMENT pfxarray ( typeref | %ArrayElements; ) >
  <!ATTLIST pfxarray
    %PrefixedArrayAttributes;
  >
<!ELEMENT limarray ( typeref | %ArrayElements; ) >
  <!ATTLIST limarray
    %DelimitedArrayAttributes;
  >
<!ELEMENT imparray ( typeref | %ImplicitArrayElements; ) >
  <!ATTLIST imparray
    %ImplicitArrayAttributes;
  >
<!-- ===== -->
<!ELEMENT date ( typeref | %StringTypes; ) >
  <!ATTLIST date
    %DateAttributes;
  >
<!-- ===== -->
<!ELEMENT number ( typeref | %StringTypes; ) >
  <!ATTLIST number
    %NumberAttributes;
  >
<!-- ===== -->
<!ELEMENT typeref EMPTY >
  <!ATTLIST typeref
    %NonTypeAttributes;
    type IDREF #REQUIRED
  >

```

```
<!-- ===== -->
<!ELEMENT pad EMPTY >
  <!ATTLIST pad
    %NonTypeAttributes;
    length %Number;      #REQUIRED
  >
<!-- ===== -->
```


C

Transformations

This appendix provides a list of the OracleAS InterConnect transformations.

Copy Fields

Copy the source field(s) into the destination field(s).

Parameters:

None

Copy Object

Copy the source object into the destination object.

Parameters:

None

Concat Fields

Concatenate the source field(s) and copy into the destination fields.

Parameters:

<code>prefix</code>	An optional prefix to the concatenated string.
<code>separator</code>	The separator, a string of characters, that separate source fields in the concatenated string.
<code>suffix</code>	An optional suffix to the concatenated string.

Expand Fields

Expand the source field into the destination fields.

Parameters:

<code>delimiter</code>	The delimiter or string of characters around which the source field should be broken up.
------------------------	--

Set Constant

Copy a constant into the destination fields.

Parameters:

`constant` The constant to be copied.

True Conditional Lookup XRef

Finds the source field in a cross reference table. If condition is satisfied, copy it into the destination field.

Parameters:

`condition` The condition for this parameter.

`table` The cross reference table.

`pass through` When there is no corresponding cross reference, and this parameter is `true`, the destination field is set to the source field. If this parameter is `false`, the destination field is set to null.

True Conditional Lookup DVM

Finds the source field in a domain value map table. If condition is satisfied, copy it into the destination field.

Parameters:

`condition` The condition for this parameter.

`table` The domain value map table.

`pass through` When there is no corresponding domain value map and this parameter is set to `true`, the destination field is set to the source field. If this parameter is set to `false`, the destinations field is set to null.

Conditional Copy

Copy the source field(s) into the destination field(s) if expression is satisfied.

Parameters:

<code>expression</code>	The expression.
<code>only copy on true</code>	If this parameter is set to <code>true</code> and the expression evaluates to <code>false</code> , nothing is copied. If this parameter is set to <code>false</code> and the expression evaluates to <code>false</code> , the second input object is copied.

True Conditional Copy

Copy the source field(s) into the destination field(s) if condition is satisfied.

Parameters:

<code>condition</code>	The condition for this parameter.
------------------------	-----------------------------------

True Conditional Concat

Concatenate the source field(s) into the destination field if the condition is satisfied.

Parameters:

<code>condition</code>	The condition for this parameter.
<code>prefix</code>	An optional prefix to the concatenated string.
<code>separator</code>	The separator, a string of characters, that separate source fields in the concatenated string.
<code>suffix</code>	An optional suffix to the concatenated string.

True Conditional To Number

Convert the sign, value, and precision source fields into a number and copy it into the destination field if condition is satisfied.

Parameters:

<code>condition</code>	The condition for this parameter.
<code>int length</code>	The number of digits before the decimal point excluding the sign.
<code>dec length</code>	The number of digits after the decimal point.
<code>character</code>	The padding character.
<code>DVM</code>	An optional domain value map to lookup decimal point character.

False Conditional Copy

Copy the source field(s) into the destination field(s) if condition is NOT satisfied.

Parameters:

<code>condition</code>	The condition for this parameter.
------------------------	-----------------------------------

False Conditional Concat

Concatenate the source field(s) into the destination field if the condition is NOT satisfied.

Parameters:

<code>condition</code>	The condition for this parameter.
<code>prefix</code>	An optional prefix to the concatenated string.
<code>separator</code>	The separator, a string of characters, that separate source fields in the concatenated string.
<code>suffix</code>	An optional suffix to the concatenated string.

False Conditional To Number

Convert the sign, value, and precision source fields into a number and copy it into the destination field if condition is NOT satisfied.

Parameters:

<code>condition</code>	The condition for this parameter.
<code>int length</code>	The number of digits before the decimal point excluding the sign.
<code>dec length</code>	The number of digits after the decimal point.
<code>character</code>	The padding character.
<code>DVM</code>	An optional domain value map to look up a decimal point character.

To Number

Convert the sign, value, and precision source fields into a number and copy it into the destination field.

Parameters:

<code>int length</code>	The number of digits before the decimal point excluding the sign.
<code>dec length</code>	The number of digits after the decimal point.
<code>character</code>	The padding character.
<code>DVM</code>	An optional domain value map to look up a decimal point character.

Substring

Copy a substring of the source field into the destination field.

Parameters:

<code>begin index</code>	The index at which the substring begins.
<code>length</code>	An optional length of the substring.

Char Replace

Replace characters in the source field and copy it into the destination field.

Parameters:

<code>targets</code>	The string of characters to replace.
<code>replacements</code>	The string of replacement characters.

String Replace

Replace each occurrence of a string in the source field and copy it into the destination field.

Parameters:

<code>targets</code>	The string of characters to replace.
<code>replacements</code>	The string of replacement characters.

LTrim

Delete source field characters starting from the left until a character from the set is found and copy the remaining string into the destination field.

Parameters:

<code>characters</code>	The string of characters to seek that stop the deletion.
-------------------------	--

RTrim

Delete source field characters starting from the right until a character from the set is found and copy the remaining string into the destination field.

Parameters:

`characters` The string of characters to seek that stop the deletion.

LPad

Pad source field starting from the left for a given length and copy it into the destination field.

Parameters:

`length` The padding length.

`character` An optional character to pad with, default is `<space>`.

RPad

Pad source field starting from the right for a given length and copy it into the destination field.

Parameters:

`length` The padding length.

`character` An optional character to pad with, default is `<space>`.

Lookup XRef

Lookup the source field in a cross reference table and copy it into the destination field.

Parameters:

<code>table</code>	The cross reference table.
<code>pass through</code>	When there is no corresponding cross reference and this parameter is set to <code>true</code> , the destination field is set to the source field. If this parameter is set to <code>false</code> , the destination field is set to <code>null</code> .

Delete XRef

Delete the source field from a cross reference table.

Parameters:

<code>table</code>	The cross reference table.
--------------------	----------------------------

Lookup DVM

Look up the source field in a domain value map table and copy it into the destination field.

Parameters:

<code>table</code>	The cross reference table.
<code>pass through</code>	When there is no corresponding domain value map and this parameter is set to <code>true</code> , the destination field is set to the source field. If this parameter is set to <code>false</code> , the destination field is set to <code>null</code> .

Truncate

Truncate source field starting from the right for a given length and copy it into the destination field.

Parameters:

length The length to truncate.

Increment.

Increment a counter and copy the incremented value into the destination field.

Parameters:

start value The initial counter value.

counter Give this counter a name to distinguish it from other counters that may be at different values at a given time and may have a different step size.

step size The increment size.

DatabaseOperation

Apply some SQL or PL/SQL operations to the source field(s) and copy the result to the destination field(s). This transformation can be processed on any database, not necessarily the hub or database adapter instance. Connect to the database given by the connectivity parameters and bind the input variables to the corresponding bind variables of the SQL or PL/SQL given by the `operation` parameters.

The statement is then executed. Upon successful execution, the results are copied to the destination field of the transformation. The connection to the database is then closed and the result of the transformation is returned.

Parameters:

db user The database user name.

db password The password of the database user.

db host The hostname of the database.

db port The listener port of the database.

db sid	The SID of the database.
jdbc driver	The JDBC driver to use (thin or oci).
retry count	The number of retries.
retry delay	The delay before each retry in seconds.
operation	The SQL or PL/SQL statement of the database operation.

Note: For the PL/SQL type of transformation, the following syntax of the PL/SQL statement is assumed:

- IN parameters are specified with a **?I**
- OUT parameters are specified with a **?O:<T>**
- IN/OUT parameters are specified with a **?IO:<T>**

where <T> is a single character type specifier denoting the type of the variable. Valid variable values are:

- S-String
 - I-Integer
 - F-Float
 - D-Double
 - T-Date
 - B-Binary
-

Index

A

- activity
 - populating a business process with, 7-11
- adapter.ini file
 - configuring for D3L, B-39
 - setting the ota.d3ls parameter, B-5, B-39, B-61
 - setting the ota.type parameter, B-39, B-61
 - specifying the default message endpoint, B-34
- adapters, 1-3
 - agents, bridges, 8-7
 - common features, 9-4
 - D3L responsibilities, B-2
 - features, 9-5
 - integration, 1-12
 - sdk, 1-4
- agents, 8-7
 - use with D3L, B-31
- application data types, 3-2
 - importing a D3L file, B-40
- application view, 3-2
- applications, 2-3
 - application view, 3-2
 - creating, 3-2
 - overview, 3-2
- attributes
 - adding to common data types, 3-5
 - deleting and clearing from common data types, 3-8
 - importing for common data types, 3-6
 - modifying mappings, 6-11
 - removing mappings, 6-11

B

- bridges, 8-7
 - use with D3L, B-31
- browsers.init file
 - configuring for D3L, B-37
- business objects, 2-3
 - creating, 3-3
 - defining in a D3L file, B-10
 - overview, 3-3
- business process, 7-9
 - creating, 7-11
 - populating with activities, 7-11

C

- common data types
 - adding attributes, 3-5
 - creating, 3-3
 - deleting and clearing attributes, 3-8
 - importing a D3L file, B-40
 - importing attributes, 3-6
- common view, 2-3
 - defining, 3-3
 - overview, 3-3
- components, 1-3
 - adapters, 1-3
 - oracle workflow, 1-4
 - OracleAS interconnect hub, 1-3
 - repository, 8-9
 - sdk, 1-4
- console monitors, 9-2
- content-based routing, 2-6
 - working with, 6-2

- context menu, 2-13
- cross reference tables, 2-6
 - adding applications, 6-7
 - populating, 6-8
 - removing applications, 6-7
 - working with, 6-7
- custom transformations
 - adding, 6-11
 - deleting, 6-12

D

D3L. *See* Data Definition Description Language (D3L)

Data Definition Description Language (D3L)

- common view, B-43
- common view to native format message outgoing messages translations, B-34
- configuration
 - configuring a native format message with a D3L file, B-5, B-38
 - configuring the browsers.init file with iStudio, B-37
 - configuring with OracleAS InterConnect Adapters, B-39
 - creating a D3L file describing native format messages, B-38
 - creating a native format message, B-37
 - defining metadata properties with each event, B-41
 - importing a D3L file in iStudio, B-40
- D3L DTD, B-75
- D3L file examples, B-5, B-9, B-38, B-45, B-68, B-71, B-73
- defining business objects, B-10
- defining events, B-10
- definition, B-2
- file structure example, B-9
- installing, B-36
- iStudio values in the D3L file, B-10
- magic value message header attributes, B-8, B-33
- message header attributes, B-6
- name/value pair message header attributes, B-6, B-32, B-38

- native format message examples, B-4, B-37, B-44, B-70, B-72
- native format message to common view
 - incoming message translations, B-32
- runtime initialization, B-31
- setting the ota.d3ls parameter in the adapter.ini file, B-5
- supported data types, B-12
- unsuitable D3L formats, B-3
- use case, B-42
 - configuring the aqapp_pub and fileapp_sub applications in iStudio, B-46
 - creating a business object Employee, B-47
 - creating a D3L file for the FTP adapter, B-44
 - creating a DTD file for the Advanced Queuing Adapter, B-44
 - creating a new workspace and new project, B-47
 - creating the aqapp_pub application, B-49
 - creating the fileapp_sub application, B-55
 - creating the newEmployee Event, B-48
 - defining the application queue for the aqapp_pub application, B-54
 - defining the application view, B-51, B-56
 - defining the application view to common view mapping, B-52, B-57
 - enabling the aqapp_pub application to publish the newEmployee event, B-50
 - installing the Advanced Queuing and FTP adapters, B-58
 - making the fileapp_sub application subscribe to the newEmployee event, B-56
 - overview of aqapp_pub and fileapp_sub applications, B-43
 - running the use case, B-62
 - selecting the event to publish, B-50
 - selecting the event to which to subscribe, B-56
 - using other adapters, B-66
 - using XML mode, B-66
 - when to use, B-3
 - XML mode, B-2
- data types
 - supported by D3L, B-12
- deploy navigation tree, 2-12

- deployment
 - to oracle workflow, 7-13
- design navigation tree, 2-12
- design time, 1-7
- design time tools, 7-4
- detail view, 2-13
- development kit, 1-3
- domain value mapping tables
 - deleting, 6-10
- domain value mappings, 2-7
 - adding applications to, 6-9
 - creating, 6-9
 - deleting, 6-10
 - modifying data in, 6-10
 - removing, 6-9
 - working with, 6-9

E

- edit menu, 2-10
- enabling infrastructure, 6-2
 - content-based routing, 6-2
 - cross reference tables, 6-7
 - domain value mappings, 6-9
- error management
 - resubmission, 8-4
 - tracing, 8-4
 - tracking, 8-4
- event maps, 4-2
- event menu, 2-10
- events, 2-3
 - creating, 4-3
 - defining in a D3L file, B-10
 - importing a D3L file, B-40
 - publishing, 4-4, 4-5
 - subscribing, 4-12

F

- features
 - adapter, 9-5
 - error management, 8-4
 - integration lifecycle management, 1-11
 - integration logic, platform functionality, 1-8
 - integration methodology, 1-9

- message delivery, 8-3
- message retention, 8-3
- messaging paradigms, 8-2
- repository, 9-5
- routing support, 8-3
- scalability and load balancing, 8-4
- standard messaging, 1-5

H

- help menu, 2-10
- hub and spoke
 - how it works, 1-9

I

- infrastructure
 - enabling, 6-2
- installation
 - oracle workflow components, 7-8
- integration
 - create a cross reference table, A-12
 - create a project, A-7
 - create an oracle workflow process bundle, A-26
 - create applications, A-11
 - create business object events, A-8
 - create common view business object, A-8
 - create content based routing, A-25
 - create publish events, A-13
 - creating objects in oracle workflow for modeling, A-31
 - deploy the process bundle to oracle workflow, A-29
 - deployment, A-39
 - dtd code, A-9
 - exporting and installing code, A-40
 - implementing the scenario, A-6
 - legacy system, A-2
 - legacy system database trigger, A-6
 - modeling business logic in oracle workflow, A-35
 - modeling the integration, A-4
 - new centralized system, A-2
 - overview, A-2
 - pushing metadata, A-40

- setting queues, A-39
- subscribing to events, A-16
- the integration scenario, A-3
- using adapters, 1-12
- using istudio for modeling, A-5
- integration architecture, 8-2
- integration logic, 1-8
- integration methodology, 1-9
- integration process
 - design time, 1-7
 - overview, 1-7
 - runtime, 1-7
- istudio, 1-3
 - activities, 7-9
 - adding applications to cross reference tables, 6-7
 - adding applications to domain value mappings, 6-9
 - adding custom transformations, 6-11
 - adding mapping variables, 6-12
 - application data types, 3-2
 - applications, 2-3
 - common views and business objects, 2-3
 - concepts, 2-2
 - content-based routing, 2-6, 6-2
 - context menu, 2-13
 - creating a business object, 3-3
 - creating a business process, 7-11
 - creating a new project, 2-15
 - creating a new workspace, 2-13
 - creating a procedure, 5-3
 - creating a process bundle, 7-11
 - creating an application, 3-2
 - creating an event, 4-3
 - creating common data types, 3-3
 - creating domain value mappings table, 6-9
 - cross reference tables, 2-6, 6-7
 - deleting custom transformations, 6-12
 - deleting domain value mapping tables, 6-10
 - deleting domain value mappings, 6-10
 - deleting mapping variables, 6-13
 - deploy navigation tree, 2-12
 - design navigation tree, 2-12
 - detail view, 2-13
 - domain value mapping, 2-7

- domain value mappings, 6-9
- event maps, 4-2
- events, 2-3
- exporting stored procedures, 5-18
- implementing a procedure, 5-12
- invoking a procedure, 5-5
- invoking and implementing a procedure, 5-4
- launching oracle workflow builder, 7-17
- launching oracle workflow tools, 7-16
- launching the oracle workflow home page, 7-16
- mapping, transformations, 2-4
- menu bar, 2-9
- metadata versioning, 2-5
- modifying attribute mappings, 6-11
- modifying data in domain value mappings, 6-10
- opening a project, 2-16
- opening a workspace, 2-14
- overview, 2-2
- parts of the window, 2-8
- populating cross reference tables, 6-8
- procedures, 2-4, 5-2
- projects, 2-14
- publishing an event, 4-4, 4-5
- removing applications from cross reference tables, 6-7
- removing applications from domain value mappings, 6-9
- removing attribute mappings, 6-11
- routing, message capability matrix, 2-7
- sdk, 1-4
- starting, 2-7
- subscribing to an event, 4-12
- toolbar, 2-11
- tracking fields, 2-6
- workspaces, 2-13

M

- mapping, 2-4
- mapping and transformations, 2-4
- mapping variables
 - adding, 6-12
 - deleting, 6-13
- menu bar, 2-9

- edit menu, 2-10
- event menu, 2-10
- file menu, 2-9
- help menu, 2-10
- procedure menu, 2-10
- message capability matrix, 2-7
- messaging
 - standard, 1-5
 - supported paradigms, 1-6
- metadata
 - defining for D3L, B-41
- metadata versioning, 2-5

N

- native format message
 - examples, B-4, B-37, B-44, B-70, B-72

O

- oracle enterprise manager central console
 - starting, 9-2
- oracle workflow, 1-4, 7-2
 - apply business logic, 7-8
 - composite services, 7-3
 - deploy business process for runtime, 7-8
 - deployment, 7-13
 - design business process, 7-8
 - design time tools, 7-4
 - error management, 7-2
 - installing components, 7-8
 - integration with oracle applications
 - interconnect, 7-4
 - launching oracle workflow builder, 7-17
 - launching the home page, 7-16
 - launching tools, 7-16
 - message junctions, 7-3
 - modify existing processes, 7-18
 - overview, 7-2
 - runtime, 7-7
 - solves business problems, 7-2
 - stateful routing, 7-3
- OracleAS InterConnect, C-1
- OracleAS interconnect
 - components, 1-3

- overview, 1-2
- sdk, 1-4
 - using oracle workflow, 7-8
- OracleAS interconnect components
 - development kit, 1-3
- OracleAS interconnect hub, 1-3
- ota.d3ls parameter
 - setting in the adapter.ini file, B-5, B-39, B-61
- ota.type parameter
 - setting in the adapter.ini file, B-2, B-39, B-61

P

- platform functionality, 1-8
- procedure menu, 2-10
- procedures, 2-4
 - creating, 5-3
 - exporting stored procedures, 5-18
 - implementing, 5-12
 - importing a D3L file, B-40
 - invoking, 5-5
 - invoking and implementing, 5-4
 - using, 5-2
- process bundle, 7-9
 - creating, 7-11
- projects
 - creating, 2-15
 - opening, 2-16
 - using, 2-14

R

- RAC, 8-10
 - configuration, 8-11
 - introduction, 8-10
- Real Application Clusters, 8-10
- repository
 - common features, 9-4
 - features, 9-5
- routing, 2-7
- routing support
 - content-based routing, 8-3
- runtime, 1-7, 7-7
 - components, 8-7
 - features, 8-2

- management features, 9-4
- runtime components
 - adapters, 8-7
 - advanced queues, 8-10
 - workflow, 8-10

S

- standard messaging, 1-5
- stored procedures, 5-18

T

- toolbar, 2-11
- tracking fields, 2-6
- transformations, 2-4
 - char replace, C-7
 - concat fields, C-2
 - conditional copy, C-4
 - copy fields, C-2
 - copy object, C-2
 - delete xref, C-9
 - expand fields, C-2
 - false conditional concat, C-5
 - false conditional copy, C-5
 - false conditional to number, C-6
 - increment, C-10
 - l trim, C-7
 - lookup dvm, C-9
 - lookup xref, C-9
 - lpad, C-8
 - r trim, C-8
 - rpadd, C-8
 - set constant, C-3
 - string replace, C-7
 - to number, C-6
 - true conditional concat, C-4
 - true conditional copy, C-4
 - true conditional lookup dvm, C-3
 - true conditional lookup xref, C-3
 - true conditional to number, C-5
 - truncate, C-10
 - x substring, C-7

W

- workspaces
 - creating, 2-13
 - opening, 2-14
 - using, 2-13