Oracle® Message Broker

Administration Guide

Release 2.0.1.0 for SPARC Solaris and Windows NT

September 2000

Part No. A65435-01

ORACLE®

Oracle Message Broker Administration Guide, Release 2.0.1.0 for SPARC Solaris and Windows NT

Part No. A65435-01

Primary Author: Tom Van Raalte

Contributors: Kirk Bittler, Geoff Brown, Mark Callaghan, Pascal Felber, Kathryn Gruenefeldt, Charles Hall, Doug Hood, Anish Karmarkar, John Lang, John Leinaweaver, Brian Lough, Vivek Maganty, Nachiketa Sharma, Shengsong Ni, Joan Silverman, Sanjay Singh, Jean Marie Sulmont, Neal Wyse.

Contents

2 Quick Start

3 JMS Programming

4 Administration

5 Oracle Message Broker Features

6 Oracle Message Broker Extensions

7 Message Servers and Drivers

8 Oracle Message Broker Propagation

12 Security

13 Lightweight Configuration

14 Asynchronous Component Invocation

A Oracle AQ Driver ADTs

Index

Send Us Your Comments

Administration Guide, Release 2.0.1.0 for SPARC Solaris and Windows NT

Part No. A65435-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: infodev@oracle.com
- FAX - (503) 525-8000. Attn: Oracle Message Broker
- Postal service:
  Oracle Corporation
  Oracle Message Broker
  1211 SW 5th Avenue, Suite 900
  Portland, Oregon 97204
  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

The *Oracle Message Broker Administration Guide* provides instructions for configuring and using the Oracle Message Broker. The Oracle Message Broker implements a version of the Java Message Service (JMS) API. This guide shows the components and commands that allow you to work with the Oracle Message Broker.

Intended Audience

This guide is intended for anyone who needs to work with the Oracle Message Broker. It assumes that you are familiar with the common commands on a Windows NT system, or on a Unix system.

Structure

This guide contains fourteen chapters and one appendix:

Chapter 1	Introduction, provides an introduction to the Oracle Message Broker and its features.
Chapter 2	Quick Start, provides instructions and samples that show you how to quickly set up and use the Oracle Message Broker.
Chapter 3	JMS Programming, describes the procedures for basic JMS programming using the JMS API with the Oracle Message Broker.
Chapter 4	Administration, describes the organization of the Oracle Message Broker's directory entries, and the Oracle Message Broker administrative commands and utilities.

Chapter 5	Oracle Message Broker Features, includes advanced programming features and Oracle Message Broker JMS provider specific information.
Chapter 6	Oracle Message Broker Extensions, covers Oracle Message Broker features that are not part of the JMS Specification.
Chapter 7	Message Servers and Drivers, describes the Oracle Message Broker features that allow programmers to store JMS messages and use existing Message Oriented Middleware (MOM) products.
Chapter 8	Oracle Message Broker Propagation, describes the Oracle Message Broker components that control the transfer of messages between Oracle Message Brokers using either IIOP or HTTP protocols.
Chapter 9	Oracle Message Broker C++ API, contains instructions for using the Oracle Message Broker C++ API. The C++ API allows clients to write C++ code that uses the Oracle Message Broker.
Chapter 10	Logging and Troubleshooting, contains information on the Oracle Message Broker logging file.
Chapter 11	Administration GUI, contains information on the Oracle Message Broker Manager that is used for performing administration tasks.
Chapter 12	Security, contains information on the Oracle Message Broker security features.
Chapter 13	Lightweight Configuration, contains information on Oracle Message Broker configuration using Java properties and files, instead of the LDAP Directory.
Chapter 14	Asynchronous Component Invocation, describes the ACI feature. The ACI feature links JMS, as provided by the Oracle Message Broker, to applications running in the Oracle Database Server.
Appendix A	Oracle AQ Driver ADTs, describes the JMS ADTs that the Oracle Message Broker supports with the AQ Driver.

Related Documents

For details on JMS, refer to the Java Message Service documentation and the Java Message Service specification available from Javasoft at the following site,

`http://www.javasoft.com/products/jms`

Conventions

The following conventions are used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
`courier`	Text to be entered exactly as it appears.

1 Introduction

The following sections introduce the Oracle Message Broker and its features:

- What is the Oracle Message Broker?
- What is JMS?
- Oracle Message Broker Components
- Oracle Message Broker Features
- Administration and Monitoring Utilities
- Oracle Message Broker Deployment Options

What is the Oracle Message Broker?

Oracle Message Broker plays a major role in Oracle's solution for Enterprise Application Integration (EAI). It is a scalable and open platform ideal for integrating strategic applications, e-commerce, or legacy systems.

The Oracle Message Broker provides an open, asynchronous, system-independent, message-based communication mechanism. The foundation of the Oracle Message Broker is an implementation of the Java Message Service API. Oracle Message Broker allows different applications or systems to interact in a near real-time, robust, reliable, and scalable manner to complete end-to-end cross-functional business processes. A message selection engine supports message routing based on the contents of a message header, and transactions are supported across application boundaries.

The Oracle Message Broker integrates applications on the business process level that work across the enterprise. While integrating applications, the Oracle Message Broker shields you from the complicated underlying messaging technology. Because of its interoperability with a complete line of Oracle products, and with the leading commercial messaging products, the Oracle Message Broker seamlessly integrates applications and leverages existing technology.

What is JMS?

The Oracle Message Broker implements an important and powerful standard called the Java Message Service (JMS)[1]. The Oracle Message Broker implementation of JMS supports many of the existing Message Oriented Middleware (MOM) solutions. The JMS API is a standard developed by Sun Microsystems, Inc. to support enterprise messaging. JMS provides a standards based API for writing distributed applications. Enterprise messaging using JMS provides a reliable, flexible service for the asynchronous exchange of critical business data and events throughout an enterprise.

Refer to the Java Message Service documentation and the Java Message Service specification available from Javasoft at the following site for more information on JMS:

`http://www.javasoft.com/products/jms`

[1] The Oracle Message Broker implements the JMS specification with some restrictions, and with limitations based on the selected driver. The *Oracle Message Broker Release Notes*, and Chapter 7, "Message Servers and Drivers" describe JMS features, limitations, and restrictions. When we refer to the Oracle Message Broker implementation of JMS or refer to the Oracle Message Broker as a JMS provider throughout this guide, we acknowledge these limitations.

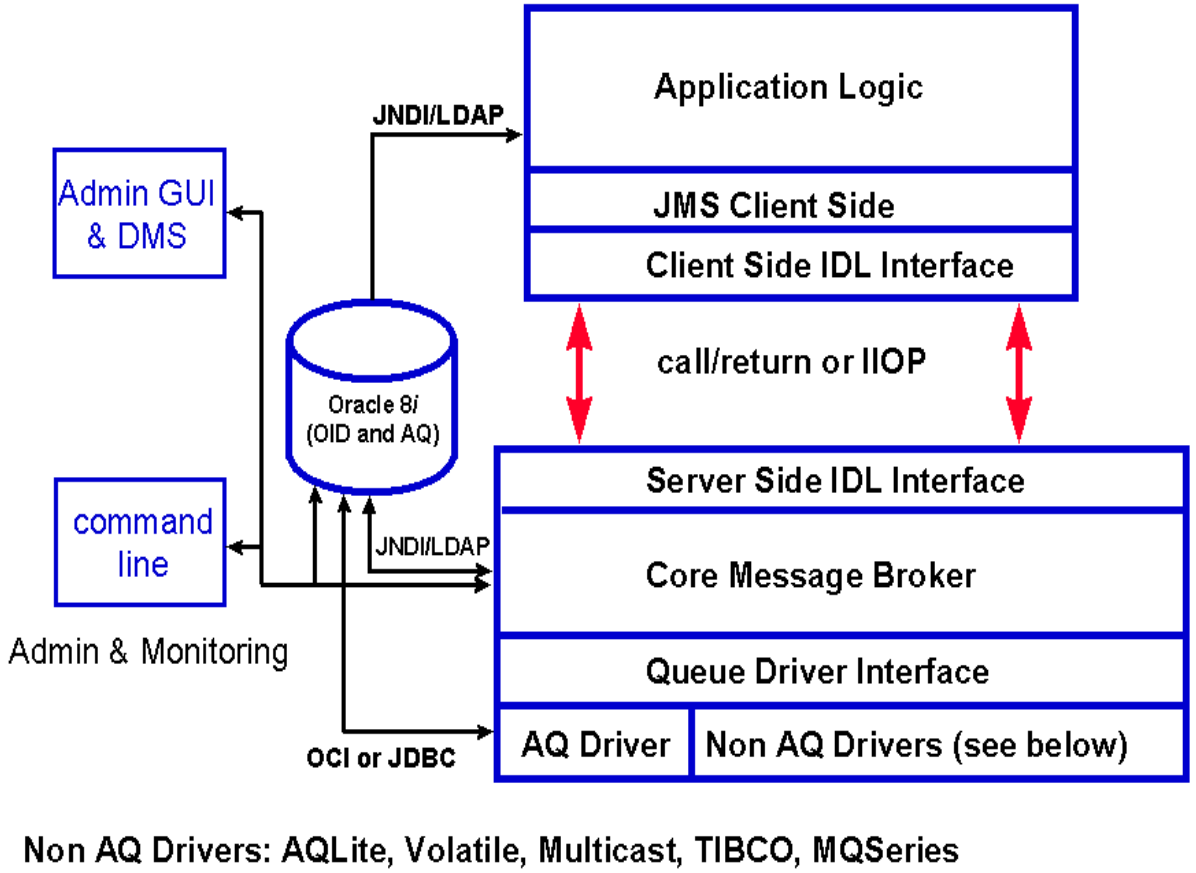
Oracle Message Broker Components

The Oracle Message Broker consists of the following components:

- Oracle Message Broker Core
- Drivers and Message Servers
- Administrative Components
- Client Programming Interface

Figure 1–1 shows the Oracle Message Broker's important components.

Figure 1–1 Oracle Message Broker Architecture



Oracle Message Broker Core

The Oracle Message Broker Core is a JMS provider[1]. A JMS provider implements the JMS API that allows you to use JMS for messaging in the enterprise, with one or more Oracle Message Brokers communicating and coordinating message transmission. The Oracle Message Broker Core provides:

- Support for pushing messages to clients
- Polling of message servers that do not support notification
- Handling of certain administrative tasks
- Multiplexing JMS clients onto the underlying message servers

The Oracle Message Broker Core also communicates with message servers using a standard driver interface that is transparent to client applications.

Drivers and Message Servers

Oracle Message Broker drivers provide access to proprietary message servers. **Message servers** manage and store messages. Drivers also coordinate message translation to native storage formats. **Native storage formats** are the formats that messages are stored in on a message server, when the message server does not directly support JMS.

Table 1–1 shows the drivers that support message servers in Oracle Message Broker.

Table 1–1 Oracle Message Broker Drivers

Driver	Underlying Message Server
Oracle Advanced Queuing (AQ) Driver	Supports persistent delivery of JMS messages using the Oracle 8*i* Database with the Advanced Queuing (AQ) messaging infrastructure.
Oracle AQ Lite Driver	Supports persistent delivery of JMS messages using Oracle 8*i* Lite and the AQ Lite messaging infrastructure.
Oracle Volatile Driver	Provides very fast delivery of JMS messages using lightweight, in-memory, communication facilities. The Volatile Driver is useful for high throughput of messages when the messaging system does not require persistent message storage.
MQSeries Driver	Supports commercial messaging system based on IBM MQSeries V5.1.

[1] The Oracle Message Broker implements the JMS specification with general restrictions, and with restrictions based on the selected driver. The *Oracle Message Broker Release Notes*, and Chapter 7, "Message Servers and Drivers" describe JMS features, limitations, and restrictions.

Table 1–1 (Cont.) Oracle Message Broker Drivers

Driver	Underlying Message Server
Oracle Multicast Driver	Supports very fast delivery of JMS messages using lightweight, multicast communication facilities. The Multicast Driver uses the Oracle Application Server Multicast Communication libraries.
TIBCO Driver	Provides very fast delivery of transient messages based on lightweight multicast communication facilities.The TIB/Rendezvous (TIBCO) Driver is based on TIB/Rendezvous Release 5.x, or TIB/Rendezvous Pro Release 5.x.

Administrative Components

The Oracle Message Broker supports two options for specifying configuration information for administration:

- LDAP Directory Configuration
- Lightweight Configuration

See "Configuration Options" on page 1-10 for a summary of the features of the configuration options.

The configuration information for an LDAP Directory is modified using the administrative utilities. See "Administration and Monitoring Utilities" on page 1-8 for information on the Oracle Message Broker utilities.

What is an OMB Instance

An **OMB Instance** defines the configuration information used by an active process running the Oracle Message Broker (an instance of the Oracle Message Broker). The configuration information is stored, either in the form of LDAP Directory entries or using the Lightweight configuration facilities. The information that defines an OMB Instance includes parameters used to configure the Oracle Message Broker process, the connection factories that JMS clients use to establish a connection with the Oracle Message Broker, and the configuration information that the Oracle Message Broker requires to connect to a queue or topic.

Using an LDAP Directory for configuration, the entries that define an OMB instance are unique for that instance and are not shared by other OMB instances. However, these entries may contain references to a physical entity that is commonly used by more than one OMB Instance. For example, from an administrative standpoint, OMB destinations, including queues and topics, are aliases for an object that is managed by a message server that stores messages. This allows different OMB destination objects to refer to the same queue or topic on the message store.

Client Programming Interface

The Oracle Message Broker client programming interface lets client programs communicate with the Oracle Message Broker. The Oracle Message Broker supports the following client programming interfaces:

- The JMS API for Java program access (see Chapter 3, "JMS Programming" for more information on this client programming interface)
- A C++ API that is similar to the JMS API (see Chapter 9, "Oracle Message Broker C++ API" for more information on this client programming interface)
- PL/SQL Operational Interface

PL/SQL Operational Interface

The Oracle Message Broker provides an AQ Driver specific PL/SQL package. The PL/SQL package is an easy to use mechanism for enqueueing and dequeueing JMS messages directly to and from Oracle AQ queues. The Oracle AQ queue used for this purpose needs to be created using the Oracle Message Broker administrative utilities. The PL/SQL package eliminates the need for PL/SQL applications running within the Database Server to use the JMS Java or C++ client interfaces to pass messages to and from Oracle AQ queues.

A message placed on an Oracle AQ queue using the PL/SQL package can be propagated between Oracle Message Brokers, or to any of the supported Oracle Message Broker drivers, including the following drivers: Oracle Volatile, IBM MQSeries, TIB/Rendezvous, or Oracle Multicast.

Legacy applications written in PL/SQL can easily be integrated using the Oracle Message Broker and the PL/SQL package. This deployment option supports the following environments:

- Developers using PL/SQL
- A hub and spoke architecture where the hub is based on AQ and PL/SQL

The following list includes some of the advantages of using the PL/SQL package:

- Only two PL/SQL subroutines are necessary to either enqueue or dequeue a message
- Efficiency – enqueues and dequeues are performed within the context of the Database Server using PL/SQL
- Support for advanced features – including propagation and message selection, are available through the Oracle Message Broker

Refer to "Using the PL/SQL Operational Interface" on page 5-7 for more information on the PL/SQL operational interface.

Oracle Message Broker Features

The Oracle Message Broker provides a complete set of messaging features. Table 1–2 lists some of the Oracle Message Broker features.

Table 1–2 Oracle Message Broker Features

Feature	Description
JMS publish/subscribe model	The publish/subscribe model is a development and deployment model well suited for integrating loosely-coupled applications. Using the publish/subscribe model, subscribers specify their interest in messages using content-based topics.
JMS point-to-point (PTP) model	The Oracle Message Broker supports the point-to-point messaging model. This model is useful for message delivery to a single destination.
Persistent JMS Delivery Mode	The Oracle AQ Driver, the AQ Lite Driver, and the IBM MQSeries Driver support persistent message delivery. Using persistent message delivery, the Oracle Message Broker leverages Oracle AQ, Oracle AQ Lite, or IBM MQSeries to log messages to stable storage as part of the message-send operation. This insures that messages are not lost during message propagation, or in the event of system failures.
Non-persistent Delivery Mode	Non-persistent delivery mode is supported for Volatile, Multicast, TIBCO/Rendezvous, and MQSeries drivers. A Non-persistent message is handled with an at-most-once delivery guarantee.
Volatile Queues	Volatile queues offer the following benefits: ■ High speed transaction processing ■ A Transient state with a short lifetime – messages are kept for the duration of an active Oracle Message Broker ■ Low overhead – the maintenance infrastructure for recoverability and persistence is not required ■ Flexibility – can be used for both transaction oriented processing and inter-process communication
Scalability	The service oriented approach combined with excellent thread management provides an inherently scalable architecture. The Oracle Message Broker can push messages to clients with a connection required only for the duration of the delivery of the message. Supporting push based clients, a single Oracle Message Broker can support a large number of clients.

Table 1–2 (Cont.) Oracle Message Broker Features

Feature	Description
Oracle Message Broker Interoperability	Provides a rich messaging mechanism, in terms of both message type and models, over widely adopted message queuing systems. This feature protects corporate investments by leveraging existing messaging systems. Users can write their API code once, and deploy it over several messaging systems. The Oracle Message Broker is designed to provide a functionally rich messaging model over a number of industry queuing systems, including: ■ Oracle Advanced Queuing ■ Oracle AQ Lite ■ IBM MQSeries ■ TIBCO Rendezvous Integrated drivers for these queuing systems ensure a base level of transparency over any given queuing system.
Platform Independence	Allows communication to a wide variety of operating system platforms. The Oracle Message Broker can talk to any of its supported messaging systems on several supported heterogeneous platforms.
Industry Standard Programming Interface	Adherence to the JMS specification provides a standards based approach, with minimal complexity.

Administration and Monitoring Utilities

This section introduces the utilities for managing the Oracle Message Broker. These utilities include:

- Command Line Tools
- Graphical User Interface
- Performance Monitoring Service

Command Line Tools

The Oracle Message Broker includes the following administration and monitoring utilities:

- A command line tool, `AdminUtil` for Oracle Message Broker administration This connects to an LDAP Directory and allows an administrator to create, delete, and manage Oracle Message Broker configuration information. The command operates in either interactive mode or batch mode. In interactive mode an administrator enters commands at an input prompt. In batch mode the utility processes files containing stored commands
- A command line tool, `AdminDirCheck`, for checking and validating Oracle Message Broker directory entries. This performs basic validation for Oracle Message Broker configuration data. It checks and validates Oracle Message Broker entries stored in the LDAP Directory.
- A command line tool, `Migrate10To20`, for migrating Oracle Message Broker directory entries from older versions.

Graphical User Interface

The Oracle Message Broker provides an easy to use Graphical User Interface (GUI) that allows an administrator to modify administrative entries stored in the LDAP Directory. The GUI helps an administrator set up an Oracle Message Broker and provides a view of an active Oracle Message Broker's configuration. Using the GUI, an administrator can easily view, create, delete, and manage Oracle Message Broker configuration information.

Performance Monitoring Service

Oracle Message Broker optionally can collect performance metrics so that an administrator can dynamically monitor system performance. The Oracle Message Broker performance metrics are available using the Dynamic Monitoring Service (DMS). The Oracle Message Broker includes a command line tool to save DMS metrics. Refer to Chapter 6, "Oracle Message Broker Extensions" for more information on working with the performance monitoring information.

Oracle Message Broker Deployment Options

This section summarizes the deployment options available for running the Oracle Message Broker. The available configuration options are:

- Using LDAP Directory Configuration
- Using Lightweight Configuration

In addition, there are two operation modes for the Oracle Message Broker:

- Local Mode
- Remote Mode

There are four possible deployment options for an Oracle Message Broker application:

1. Using the LDAP Directory in Local Mode.
2. Using the LDAP Directory in Remote Mode.
3. Using Lightweight Configuration in Local Mode.
4. Using Lightweight Configuration in Remote Mode.

Choosing from these four available options for running the Oracle Message Broker, it is up to you to select the option that best meets your needs.

Configuration Options

Selecting a configuration option for the Oracle Message Broker allows you to choose between the following:

- Using LDAP Directory Configuration
- Using Lightweight Configuration

Using LDAP Directory Configuration

Using an LDAP Directory for administrative information allows for an open, standards based access protocol for system management and configuration. LDAP Directory configuration requires an LDAP Directory installation and the directory must be configured for operation with Oracle Message Broker. This configuration

option provides location independence for Oracle Message Broker configuration information. LDAP Directory configuration also provides the following:

- Configuration files do not need to be stored locally
- A scripting language is available to modify configuration data
- A GUI is provided to modify configuration data
- Dynamic updates to configuration data are supported for Oracle Message Broker executing in Remote Mode
- Provides security services, including access control
- Support for Oracle Message Broker Propagation
- Provides location independence for Oracle Message Broker clients to access remote Oracle Message Brokers
- Supports durable storage of durable subscribers and message selectors for durable subscribers.

For more information on configuration using an LDAP Directory, refer to Chapter 4, "Administration".

Using Lightweight Configuration

With lightweight configuration, the Oracle Message Broker reads configuration information from a file or from Java properties when it begins its execution. The Oracle Message Broker configuration information specifies the names and configuration options for all administrative objects, such as the names and types of JMS destinations (topics or queues). Lightweight configuration provides the following:

- Simple configuration using either Java properties or a stored information from a configuration file.
- Lightweight configuration does not require an LDAP Directory.
- Methods for starting the AQ Driver using minimal configuration options, by providing only a since a service name, a username, and a password.
- Better availability (a running LDAP server available is not required).

For more information on configuration using an Lightweight Configuration, refer to Chapter 13, "Lightweight Configuration".

Operation Modes

The Oracle Message Broker allows you to choose between the following operation modes:

- Local Mode
- Remote Mode

Local Mode

A Local Mode Oracle Message Broker runs within the same process as the Oracle Message Broker client. When running in local operation mode, the Oracle Message Broker uses local procedure calls to interact with the Oracle Message Broker client (the Local Mode Oracle Message Broker does not start an ORB). Thus, in Local Mode, Oracle Message Broker clients can use ORBs for their own purposes.

Operation in Local Mode has the following characteristics:

- Performance is usually better when running in Local Mode. In Local Mode the Oracle Message Broker client requests do not have to pass through a separate process to access drivers, including Oracle AQ, MQSeries, or another driver.
- Availability is easier to manage in Local Mode. There is no need to keep a separate Oracle Message Broker process available.
- The Volatile Driver is of limited use in Local Mode. The Volatile Driver is more useful in Remote Mode when different clients can connect to the Oracle Message Broker and use the Volatile Driver as a destination separate from the client (in Local Mode, clients can only access the Oracle Message Broker within a single process, and all clients must run within the same process).
- Oracle Message Broker Propagation is not supported in Local Mode.
- If the security service is used within a Local Mode Oracle Message Broker, an Oracle Message Broker developer can write code that changes any decisions made by the security service.

For more information on Local Mode, refer to "Running in Local Mode" on page 5-8.

Remote Mode

A Remote Mode (Non-Local Mode) Oracle Message Broker communicates with Oracle Message Broker clients using IIOP. A Remote Mode Oracle Message Broker runs in a separate process from Oracle Message Broker clients. However, it is possible to run a Remote Mode Oracle Message Broker in the same process and communicate using IIOP.

Operation in Remote Mode has the following characteristics:

- Remote Mode Oracle Message Brokers support Oracle Message Broker Propagation.
- Remote Mode Oracle Message Brokers support the security service for access control.
- A Remote Oracle Message Broker can be configured to start the AQ Driver with a Database Server password that the Oracle Message Broker administrator does not want to expose to Oracle Message Broker clients.
- If Oracle Message Broker clients use a Remote Mode Oracle Message Broker to access Oracle AQ, then they will only be able to access Oracle AQ in a manner that is authenticated and authorized by the security service. That is, these clients will only not be able to use the Database Server username/password for general purpose Database Server access.
- Oracle Message Broker clients that use the Volatile Driver are most useful when running the Oracle Message Broker in Remote Mode.

For more information on Remote Mode, refer to "Running in Remote Mode" on page 5-14.

2

Quick Start

Oracle Message Broker provides sample administration scripts and sample programs that allow you to quickly set up and use the system. By editing the sample scripts you can customize the system for your needs.

> **Note:** This chapter provides a quick start guide for the steps required to operate the Oracle Message Broker in Remote Mode (Non-Local Mode). For information on using the Oracle Message Broker in Local Mode, refer to "Running in Local Mode" on page 5-8.

This chapter assumes that you have installed the Oracle Message Broker and that you have access to an LDAP Directory that has been updated to support the Oracle Message Broker. Refer to the *Oracle Message Broker Installation Guide* for information on installing these components.

This chapter covers the following:

Working with the Administration Utilities

This section shows you how to use the sample administration scripts to create Oracle Message Broker administrative objects. Oracle Message Broker stores administrative objects as entries in an LDAP Directory. See Chapter 4 for detailed information on the organization of the LDAP Directory.

> **Note:** The Oracle Message Broker also supports lightweight configuration for administration without using an LDAP Directory Refer to Chapter 13 for information on lightweight configuration.

The Oracle Message Broker retrieves configuration information from the LDAP Directory and uses the directory to locate and set parameters for destinations, message servers, and for other administrative tasks. Client programs also use the directory to locate the Oracle Message Broker and to find destinations for messages.

If you are setting up the Oracle Message Broker, you need to create an OMB Instance and create entries for your destinations and message servers. To perform these administration tasks you can use `AdminUtil` with the sample administration scripts shown in this chapter, or you can use the Oracle Message Broker Graphical User Interface. The Oracle Message Broker Graphical User Interface provides wizards for creating OMB Instances and other required entries. For information on using `ombadmin`, see Chapter 11, "Administration GUI".

Overview of the Sample Administration Scripts

The Oracle Message Broker sample administration scripts allow you to quickly create required administrative entries in the directory and set up Oracle Message Broker queues and topics. You can use these scripts as they are or customize them for your needs. Before working with the sample scripts, determine which drivers you need to use for your queues or topics (see Chapter 7, for more information on drivers). The Oracle Message Broker includes the following drivers:

- Oracle Advanced Queuing (AQ) Driver
- Oracle AQ Lite Driver
- IBM MQSeries Driver
- Oracle Volatile Driver
- Oracle Multicast Driver
- TIBCO/Rendezvous Driver

Table 2–1 lists the sample scripts (these are in $OMB_HOME/samples/admin or %OMB_HOME%\samples\admin on Windows NT systems.)

The sample scripts include comments that explain the entries and attributes. If you need to modify a script, copy it and then modify the copy. You can create directory entries for Oracle Message Broker by running `AdminUtil` using the sample scripts.

Table 2–1 Sample Administration Scripts

Script	Description
SetupOMB	Creates all the basic entries needed to set up an OMB Instance.
SetupACI	Creates entries required for ACI.
SetupAQ	Creates the entries needed to set up an Oracle AQ queue and topic.
SetupAQLite	Creates the entries needed to set up an Oracle AQ Lite queue and topic.
SetupMQSeries	Creates the entries needed to set up an MQSeries queue.
SetupMcast	Creates the entries needed to set up a Multicast topic.
SetupProp	Creates the entries needed to set up a propagation job. This references several entries created in the SetupOMB, SetupMQSeries, and SetupAQ scripts. Execute these scripts before SetupProp.
SetupRv	Creates the entries needed to set up a TIB/Rendezvous topic.
SetupVol	Creates the entries needed to set up Oracle Volatile queues and topics.

Using the Oracle Message Broker Instance Configuration Script

The SetupOMB script creates an Oracle Message Broker Instance (OMB Instance) and the required top level directory entries. An OMB Instance contains the administrative objects, as directory entries, required for an administrator to start or modify an Oracle Message Broker. The SetupOMB script also sets values for several required attributes. To execute the SetupOMB script, perform the following steps (this assumes that $OMB_HOME is set to the Oracle Message Broker installation directory):

1. Set the environment for your system:

   On Unix with the Bourne or Korn shell:

   ```
   $ . ./$OMB_HOME/bin/ombenv.sh
   ```

   or with the C-Shell environment:

   ```
   % source $OMB_HOME/bin/ombenv.csh
   ```

On Windows NT run the batch file:

```
> C:%OMB_HOME%\bin\ombenv.bat
```

The Oracle Message Broker installation creates these startup scripts (ombenv.bat, ombenv.sh, and ombenv.csh).

2. Execute `AdminUtil` using the SetupOMB script. `AdminUtil` displays its progress as it runs.

   On Unix:

   ```
   % cd $OMB_HOME/samples/admin
   % AdminUtil -f SetupOMB
   ```

   On Windows NT:

   ```
   > cd %OMB_HOME%\samples\admin
   > AdminUtil -f SetupOMB
   ```

Note: `AdminUtil` prompts for an authentication DN and password. In addition, when the SSL level is set to 2 or 3, it prompts for the SSL wallet location and password (the default SSL level is 0). You can enter these parameters either at the prompt, or on the command line. See Chapter 12 for information on Security. Table 4–27 contains a complete list of `AdminUtil` command line options.

Using the Driver Configuration Scripts

To run the driver sample scripts, perform the following steps:

1. Run the SetupOMB script, (see the section, "Using the Oracle Message Broker Instance Configuration Script" on page 2-3).
2. Refer to the comments in the driver setup script for information on changes you can make to customize the script.
3. Execute `AdminUtil` using the selected script for the driver you want to add. Run the Setup*Driver* script, where *Driver* is one of the following: AQ, AQLite, Vol, MQSeries, Mcast, or Rv. `AdminUtil` displays its progress as it executes.

   For example, to setup the Oracle Volatile Driver on Unix systems:

```
% AdminUtil -f SetupVol
```

On Windows NT systems, run the command:

```
> AdminUtil -f SetupVol
```

Using the Propagation Configuration Script

The propagation job sample script sets up a propagation job for transferring messages between targets. See Chapter 7 for more information on propagation and propagation jobs. To run the propagation sample script, perform the following steps:

1. Run the SetupOMB script, (see the section, "Using the Oracle Message Broker Instance Configuration Script" on page 2-3).
2. Run the Setup driver scripts for the drivers you need to support, (see the section, "Using the Driver Configuration Scripts" on page 2-4).
3. Refer to the comments found in the SetupProp script for information on changes you need to make to customize the script.
4. Execute `AdminUtil` using the SetupProp script for the queues or topics that you want to propagate. `AdminUtil` displays its progress as it executes.

   On Unix:

   ```
   % AdminUtil -f SetupProp
   ```

   On Windows NT:

   ```
   > AdminUtil -f SetupProp
   ```

Verifying Directory Contents

Use the Oracle Message Broker Manager to view the contents of the directory and verify that the entries you created are in the directory. The command to start the Oracle Message Broker Manager is:

```
% ombadmin
```

When you view your OMB Instance using the Oracle Message Broker Manager, you can verify your configuration entries.

You can also use AdminDirCheck to validate entries. Refer to "Checking Directory Entries with AdminDirCheck" on page 4-64 for information on AdminDirCheck.

Starting and Stopping the Oracle Message Broker

After creating the administrative entries using the setup scripts, execute the MsgBroker command with the -start option to start the Oracle Message Broker. Table 2–2 shows the MsgBroker command line options. Use the MsgBroker command as follows:

On Unix:

```
% MsgBroker -omb RDN [options] &
```

On Windows NT:

```
> MsgBroker -omb RDN [options]
```

where *RDN* is:

RDN — the relative distinguished name, RDN, for the message broker entry (for more information on distinguished names and LDAP, refer to Chapter 4).

For example, to start the Oracle Message Broker using the RDN, cn=msg_Broker,cn=your_Inst,cn=OMB, use the following command:

On Unix:

```
% MsgBroker -omb cn=msg_Broker,cn=your_Inst,cn=OMB -start &
```

On Windows NT:

```
> MsgBroker -omb cn=msg_Broker,cn=your_Inst,cn=OMB -start
```

For more information on starting the Oracle Message Broker, see "The msg_broker Entry and Distinguished Names" on page 2-9.

You can also start the Oracle Message Broker in Local Mode without using the MsgBroker command. For information on starting the Oracle Message Broker in Local Mode, refer to "Running in Local Mode" on page 5-8.

Table 2–2 MsgBroker Command Options

Option	Description
–D *auth_dn*	The *auth_dn* supplies the DN to use for user name authentication.
–errorlevel *level*	Set the error reporting level. The parameter *level* is set to an integer value in the range 1-4: 1 – print error message for the top exception 2 – print error messages for all linked exceptions 3 – print stack trace for the top exception 4 – print stack trace for all linked exceptions The default value for errorlevel is 2.
–fullVersion	Displays the full program version information.
–heap *size*	Supplies a heap size to the specified, *size*, in Megabytes. This sets the size of the JVM heap used by the Oracle Message Broker process.
–noauth	Specifies that LDAP authentication is not required on the LDAP server.
–omb *RDN*	Specifies the relative distinguished name, RDN, for the message broker entry (for more information on distinguished names and LDAP, refer to Chapter 4). The RDN must be enclosed in quotes on Windows NT systems.
–P *wallet_password*	Specifies the wallet password. This is ignored if the value of –U is 0 or 1.
–ping	Displays the status of the specified Oracle Message Broker. The status message indicates if the Oracle Message Broker is running. The `MsgBroker` command returns 0 if the `MsgBroker` is running. Otherwise, the `MsgBroker` command returns non-zero if the `MsgBroker` cannot be contacted. This option is not available for Local Mode operation.
–start	Use this option to start the specified `MsgBroker`.
–stats *format*	Produces a DMS log file containing Oracle Message Broker DMS statistics. The name of the DMS log file is the same as the associated Oracle Message Broker log file, prepended with “dms-”. Refer to "Collecting Runtime Metrics" on page 6-3 for details on the format of the DMS log file. The parameter *format* is set to an integer value in the range 1-4: `1` – dump DMS statistics by appending to the existing log file and use pretty print format for the data. If the DMS log file does not exist, it is created. `2`– dump DMS statistics by appending to the existing log file, do not use pretty print format for printing the data. If the DMS log file does not exist, it is created. `3` – dump DMS statistics by replacing the existing DMS log file and use pretty print format for the data. `4` – dump DMS statistics by replacing the existing DMS log file, do not use pretty print format for printing the data. This option is ignored when –start or –stop are also specified on the same command-line.

Table 2–2 (Cont.) MsgBroker Command Options

Option	Description
–stop	Using this option to shutdown the specified MsgBroker.
–U *value*	Specifies if SSL is used, and the authentication level. Valid *values* are: 0, 1, 2, and 3. `0` – no SSL. This is the default if –U is not specified. `1` – SSL with no authentication. `2` – SSL with server-side authentication. `3` – SSL with server-side and client-side authentication.
–version	Displays the program version number.
–w *auth_passwd*	Supplies a password, *auth_passwd*, for authentication on the LDAP server.
–W *wallet_path*	Specifies the path to an exported wallet file. This is ignored if the value of –U is 0 or 1.

If you do not supply the security command line options to `MsgBroker`, a dialogue box prompts for a user DN and password. Enter a user DN and password. If the directory does not use authentication, or if you have set properties to indicate the user DN and password, leave these fields blank and select the Continue button. If you select the Exit button, the `MsgBroker` command exits (for more information security, refer to Chapter 12, "Security").

If –w is specified without –D, then a dialogue box prompts for the user DN.

If –D is specified without –w, then a dialogue box prompts for the user password associated with the DN supplied with the –D.

If –noauth is specified, the –D and -w options are ignored. If no authentication properties are defined, the Oracle Message Broker attempts an anonymous bind to the LDAP Directory.

Authentication and authorization are delegated to the LDAP server. The credentials required to start the Oracle Message Broker are those required for updating Oracle Message Broker entries.

The msg_broker Entry and Distinguished Names

Starting and stopping the Oracle Message Broker using the `MsgBroker` command requires that you enter a RDN for a `msg_broker` entry. For example:

```
% MsgBroker -start -omb cn=msg_Broker,cn=your_Inst,cn=OMB
```

Where the RDN for the message broker entry is:

```
cn=msg_Broker,cn=your_Inst,cn=OMB
```

In this example, the full DN for the `msg_Broker` entry is the relative distinguished name shown above, plus the following:

```
cn=Products,cn=OracleContext,ou=sales,o=oracle,c=us
```

The Oracle Message Broker installer writes several distinguished name components to the startup scripts, including: the country, c=, the organization, o=, and the organizational unit, ou=. These scripts set environment variables that the `MsgBroker` command uses to start the Oracle Message Broker. Oracle standards specify the following entries:

```
cn=OMB,cn=Products,cn=OracleContext
```

The initial naming context is the top level component of the full message broker DN. For example, in the sample above, the initial naming context is:

```
cn=Products,cn=OracleContext,ou=sales,o=oracle,c=us
```

The OMB_IC environment variable, that the `MsgBroker` command uses contains the initial context. This variable is set in the startup scripts: ombenv.bat, ombenv.csh, and ombenv.sh (see "Using the Oracle Message Broker Instance Configuration Script" on page 2-3 for information on these scripts).

Note: The `MsgBroker` command, with the `-omb` option fails with an "`unexpected error: entry not found`" when a full DN is supplied rather than a RDN for the message broker entry.

Required Environment Variables

Oracle Message Broker clients, and the MsgBroker command require the environment variables, OMB_EF, OMB_IC, OMB_LP, and OMB_OF. The ombenv scripts define the values for these environment variables (for more information on the ombenv scripts, see "Using the Oracle Message Broker Instance Configuration Script" on page 2-3). Table 2–3 shows the required environment variables.

Table 2–3 Oracle Message Broker Environment Variables

Variable	Description
OMB_EF	Entry Factory
OMB_IC	LDAP Initial Context
OMB_LP	LDAP Provider Properties
OMB_OF	Object Factory

Stopping the Oracle Message Broker

The Oracle Message Broker command `MsgBroker -stop` executes a shutdown of all drivers, connections, and open transactions. To shut down the Oracle Message Broker and stop the system, issue the command:

On Unix:

```
% MsgBroker -omb RDN -stop &
```

On Windows NT:

```
> MsgBroker -omb RDN -stop
```

Where the value *RDN* is the relative distinguished name (RDN) for the message broker entry of the active Oracle Message Broker.

> **Note:** Stopping the Oracle Message Broker using `MsgBroker` with the `-stop` option can take several seconds, or longer.

For example:

```
% MsgBroker -omb cn=msg_broker,cn=your_Inst1,cn=OMB -stop
```

If you do not supply the security command line options to `MsgBroker`, a dialogue box prompts for a user DN and password. Enter a user DN and password. If the

directory does not use authentication, or if you have set properties to indicate the user DN and password, leave these fields blank and select the Continue button. If you select the Exit button, the `MsgBroker` command exits (for more information security, refer to Chapter 12, "Security").

Authentication and authorization are delegated to the LDAP server. The credentials required to stop the Oracle Message Broker are those required for updating Oracle Message Broker entries.

You can also stop the Oracle Message Broker when it is running in Local Mode. For information on using the Oracle Message Broker in Local Mode, refer to "Running in Local Mode" on page 5-8.

Checking the Status of the Oracle Message Broker

Use the –`ping` option to MsgBroker to check if an Oracle Message Broker is available. Table 2–2 shows the `MsgBroker` command line options, including the –`ping` option. To check the status of the Oracle Message Broker, use the `MsgBroker` command as follows:

On Unix:

```
% MsgBroker -ping -omb RDN [options]
```

On Windows NT:

```
> MsgBroker -ping -omb RDN [options]
```

where *RDN* is:

RDN — the relative distinguished name, RDN, for the message broker entry that you want to check the status for (for more information on distinguished names and LDAP, refer to Chapter 4).

Note: Status checking using MsgBroker with the –`ping` option is not available for Local Mode.

If the specified Oracle Message Broker is available, `MsgBroker` displays the following message and returns with a 0 return value:

```
Broker answered
```

If the specified Oracle Message Broker is not available, `MsgBroker` displays the following message and returns with a non-zero return value:

```
Broker unreachable
```

Running Oracle Message Broker as an NT Service

When running on Windows NT systems with Java 1.2, the Oracle Message Broker can be installed as a service that can be started automatically when the system starts up. To start Oracle Message Broker as a service, use the command:

```
%OMB_HOME%\bin\Register options
```

Where, *options* are the same as available for the `MsgBroker` command (Table 2–2 shows the `MsgBroker` command line options). For example, the following command registers an Oracle Message Broker instance as a service:

```
> Register -noauth -omb cn=msg_broker,cn=testomb,cn=OMB -start
```

Note that only one instance of Oracle Message Broker can be started as a service per system. Before running the register command, set the default JVM for the system to the JVM installed by Oracle Message Broker. In addition, the default PATH environment variable for the system must contain %OMB_HOME%\jdk\bin, %OMB_HOME%\bin, and %OMB_HOME%\..\..\orb\bin.

To unregister a previously registered Windows NT service, use the command:

```
%OMB_HOME%\bin\Unregister
```

Running the JMS Sample Programs

The $OMB_HOME/samples/client/java/queue directory contains sample programs for sending and receiving messages (on Windows NT systems, the directory is, %OMB_HOME%\samples\client\java\queue). Sample programs for publishing and subscribing using JMS topics are in the directory $OMB_HOME/samples/client/java/topic (on Windows NT systems, %OMB_HOME%\samples\client\java\topic). The Readme files in these directories provide information on compiling and running the sample programs.

All Java client programs should include the environment variables shown in Table 2–3 on the Java command line. These environment variables set values that allow the Oracle Message Broker to run.

3

JMS Programming

This chapter covers information about programming using the Oracle Message Broker and the JMS API. The following sections describe basic JMS programming using both JMS messaging domains, point-to-point (PTP), and publish/subscribe (Publish/Subscribe). Most client applications use only one of these JMS messaging domains.

This chapter covers the following:

Deployment Options for an Oracle Message Broker Application

There are four deployment options for an application, including:

1. Using the LDAP Directory with a Local Mode Oracle Message Broker
2. Using the LDAP Directory with a Remote Mode Oracle Message Broker
3. Using Lightweight Configuration with a Local Mode Oracle Message Broker
4. Using Lightweight Configuration with a Remote Mode Oracle Message Broker

Using each of these configuration and deployment options, an Oracle Message Broker application must perform the following tasks:

1. Obtain a connection factory instance.
2. Obtain a destination instance for a topic or a queue.
3. Cleanup and shutdown.

This chapter covers the methods you need to use to perform these tasks using an LDAP Directory and either Local Mode or Remote Mode (Non-Local Mode). Chapter 13, "Lightweight Configuration" covers the details for the application programming tasks using Lightweight Configuration.

Note: Using Local Mode or Remote (Non-Local Mode), the commands you use to start and stop the Oracle Message Broker differ. Refer to "Starting and Stopping the Oracle Message Broker" on page 2-6 for information on starting and stopping the Oracle Message Broker.

Programming Roadmap (Using an LDAP Directory)

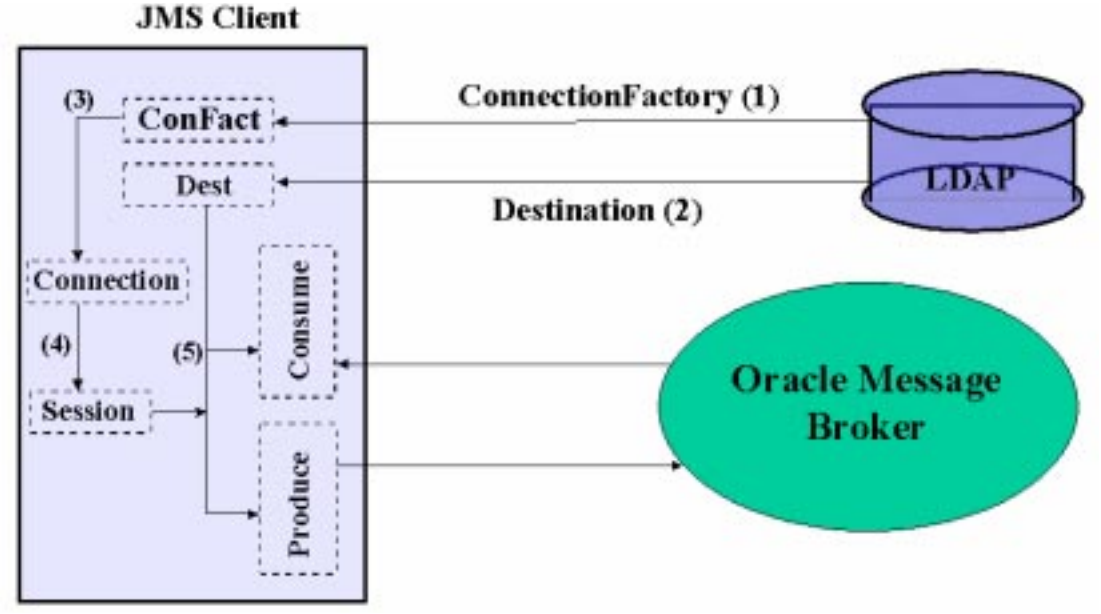
This section describes JMS programming using the Oracle Message Broker. Figure 3–1 shows the standard JMS client programming steps when using the LDAP Directory configuration option in Remote Mode. This section explains JMS programming for both Local Mode and Remote Mode.

The steps a JMS client program executes to access and use the Oracle Message Broker are as follows:

1. Use JNDI to lookup and create a *Connection Factory*. Refer to "Accessing Objects in the Directory" on page 3-4 for more information.
2. Use JNDI to lookup a *Destination*. Refer to "Accessing Objects in the Directory" on page 3-4 for more information.
3. Create and start a *Connection* using the connection factory. Refer to "Creating and Starting a Queue Connection" on page 3-9 for PTP messaging or "Creating and Starting a Topic Connection" on page 3-14 for Publish / Subscribe messaging.
4. Use the new connection to create a *Session*. Refer to "Getting a Queue Session" on page 3-10 for PTP messaging or "Getting a Topic Session" on page 3-15 for Publish / Subscribe messaging.
5. (Optional) When JNDI is not used to lookup a destination, you can use the session to create a *Destination*. Refer to "Working with Queue Destinations - QueueSender and QueueReceiver" on page 3-11 for PTP messaging or "Working with Topic Destinations - TopicPublisher and TopicSubscriber" on page 3-15 for Publish / Subscribe messaging.
6. Create and handle messages. This step differs depending on the needs of the client. Clients may need to create messages or to consume them, or both. In a PTP messaging system, this step involves sending and receiving messages along with any other required conversions or processing. In a Publish / Subscribe messaging system, this step involves publishing messages, receiving messages, and subscribing to topics. Refer to "Sending and Receiving Messages" on page 3-11 for PTP messaging or "Publishing and Subscribing" on page 3-16 for Publish / Subscribe messaging.
7. Finally, when clients finish, they need to cleanup and shutdown.

   Refer to "Shutting Down" on page 3-12 for PTP messaging or "Shutting Down (Publish / Subscribe)" on page 3-17 for Publish / Subscribe messaging.

Figure 3–1 JMS Client Programming Steps (using an LDAP Directory)



Accessing Objects in the Directory

JMS clients obtain Oracle Message Broker administrative information from an LDAP Directory. Using JMS, clients find administrative objects by looking them up using JNDI. Oracle Message Broker clients follow this convention when the configuration mode uses the LDAP Directory. In this mode, the Oracle Message Broker uses the LDAP Directory to store administrative information, including the administrative objects for both PTP and Publish/Subscribe messaging.

The Oracle Message Broker also supports domain-specific methods for creating destinations dynamically. Dynamically created destinations do not use the LDAP Directory (see "Creating Destinations" on page 6-10 for more details). Dynamic destinations are intended for use by advanced clients for specialized applications.

Using PTP messaging, client programs that send or receive messages need to obtain the following from the directory:

- A QueueConnectionFactory
- A Queue

Similarly, using Publish / Subscribe messaging, client programs need to obtain the following from the directory:

- A TopicConnectionFactory
- A Topic

Accessing Objects for Point-to-Point Messaging

Before you use JMS for PTP messaging, the administrator needs to create certain administrative objects (see Chapter 2 for information on creating administrative objects). After creating the administrative objects and starting the Oracle Message Broker, a client program obtains and uses the information stored in the directory to initialize a connection with the Oracle Message Broker.

Accessing the Connection Factory with JNDI (for Point-to-Point)

A JMS client that sends or receives messages needs to create a queue connection factory. The client uses the queue connection factory to create a connection with the Oracle Message Broker.

To access the connection factory, the administrator creates a directory entry for the queue connection factory. The client uses the JMS API, and the connection factory that it obtains from the directory lookup to establish a connection with an active Oracle Message Broker. The Oracle Message Broker also supports a JMS extension for a connection factory that supports creating topic and queue connections (see "Universal Connections and Universal Sessions" on page 6-16 for more information).

The following code shows the JNDI lookup for accessing a connection factory:

```
SimpleSession(DirContext initCtx, String cfDn)
{
  QueueConnectionFactory queueConnectionFactory;
  try
  {
    queueConnectionFactory = (QueueConnectionFactory)
      initCtx.lookup(cfDn);
    if (queueConnectionFactory == null)
      {
        System.out.println("Lookup object returned null");
        System.exit(-1);
      }
   }catch () { }
 }
```

As is the example above, to obtain a `QueueConnectionFactory` instance, the client program uses a JNDI lookup on a connection factory entry in the LDAP Directory.

Remote Mode Connection Factory Support The Oracle Message Broker's JNDI support code obtains the IOR of a remote mode Oracle Message Broker process from an LDAP Server. The Oracle Message Broker client side runtime uses the IOR to communicate with the remote mode Oracle Message Broker using IIOP.

Note the following limitations when obtaining the `QueueConnectionFactory` instance:

- It is possible that the Oracle Message Broker process associated with the IOR stored in the LDAP Directory has died.
- It is possible that several Oracle Message Brokers have been started and written their IORs into the same directory entry. In this case, the last IOR written is retrieved.

Local Mode Connection Factory Support Obtains a connection factory instance using a JNDI lookup on a connection factory entry in the LDAP Directory. Oracle Message Broker's JNDI support code either starts an Oracle Message Broker within the Oracle Message Broker client process or obtains a handle to a previously started Oracle Message Broker within the Oracle Message Broker client process in the process before the lookup returns.

Accessing the Queue with JNDI (for Point-to-Point)

You can obtain instances of javax.jms.Queue using JNDI lookups on destination directory entries. The administrator creates the queue entries and queues that a client JMS program uses to specify, and that the Oracle Message Broker uses to determine where to store messages. A client program that needs to send or receive messages uses JNDI to locate the queue entry. The JNDI lookup returns a reference to an object that implements javax.jms.Queue.

The following code shows a queue lookup that creates a `javax.jms.Queue` object:

```
queue = (Queue) initCtx.lookup(qDn);
```

Accessing Objects for Publish/Subscribe Messaging

Before you can use JMS for Publish / Subscribe messaging, the administrator needs to create certain administrative objects (see Chapter 2 for information on creating administrative objects). After creating the administrative objects and starting the Oracle Message Broker, a client program obtains and uses the information stored in the directory to initialize a connection with the Oracle Message Broker.

Accessing the Connection Factory with JNDI (for Publish/Subscribe)

A JMS client program that publishes messages to a topic, or that subscribes to a topic, needs to create a topic connection factory. The client uses the topic connection factory to create a connection with the Oracle Message Broker.

To access the connection factory, the administrator creates a directory entry for the topic connection factory. The client uses the JMS API, and the connection factory that it obtains from the directory lookup to establish a connection with an active Oracle Message Broker. The Oracle Message Broker also supports a JMS extension for a connection factory that creates topic connections and queue connections (see "Universal Connections and Universal Sessions" on page 6-16 for more information).

The following code shows the JNDI lookup for accessing a topic connection factory:

```
TopicConnectionFactory topicConnectionFactory;
try
{
  topicConnectionFactory = (TopicConnectionFactory)initCtx.lookup(cfDn);
  if (topicConnectionFactory == null)
  {
      System.out.println("Lookup object returned null");
      System.exit(-1);
  }
 }catch () { }
}
```

As is the example above, to obtain a `TopicConnectionFactory` instance, the client program uses a JNDI lookup on a connection factory entry in the LDAP Directory.

Remote Mode Connection Factory Support The Oracle Message Broker's JNDI support code obtains the IOR of an Oracle Message Broker process from an LDAP server. The client program uses the Oracle Message Broker client-side runtime to interact with an Oracle Message Broker process using IIOP.

Note the following limitations when obtaining the `TopicConnectionFactory` instance:

- It is possible that the Oracle Message Broker process associated with the IOR stored in the LDAP Directory has died.
- It is possible that several Oracle Message Brokers have been started and written their IORs into the same directory entry. In this case, the last IOR written is retrieved.

Local Mode Connection Factory Support In Local Mode, the client program obtains a connection factory instance using a JNDI lookup on a connection factory entry in the LDAP Directory. Oracle Message Broker's JNDI support code either starts an Oracle Message Broker within the Oracle Message Broker client process or obtains a handle to a previously started Oracle Message Broker within the Oracle Message Broker client process in the process before the lookup returns.

Accessing Topics with JNDI (for Publish/Subscribe)

You can obtain instances of javax.jms.Topic are using JNDI lookups on destination directory entries. The administrator creates the topic entries and queues that a client JMS program uses to specify and the Oracle Message Broker uses to determine where to store messages. A client program that needs to publish or subscribe to messages uses JNDI to locate the topic entry. The JNDI lookup returns a reference to an object that implements javax.jms.Topic.

The following code shows a topic lookup that creates a topic object:

```
topic = (Topic) initCtx.lookup(topicDn);
```

Point-to-Point Messaging

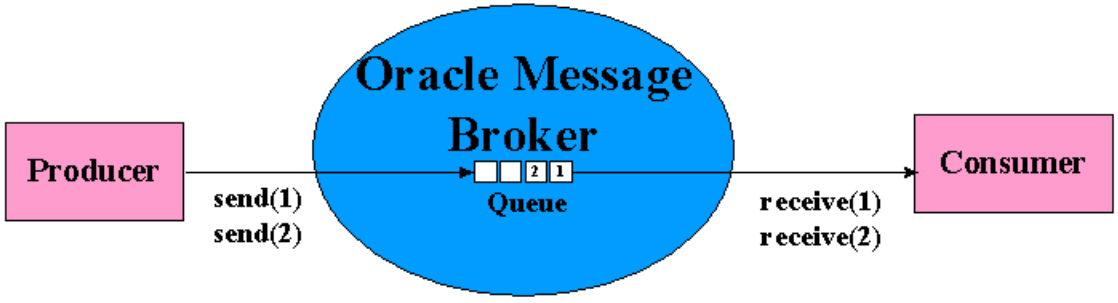
This section describes the programming steps required to send and receive messages using the JMS PTP methods. The following steps assume that the administrator has created the queue connection factory and one or more queues, and that these Oracle Message Broker administrative objects were obtained as described in "Accessing Objects in the Directory" on page 3-4.

This section covers the following:

- Creating and Starting a Queue Connection
- Getting a Queue Session
- Working with Queue Destinations - QueueSender and QueueReceiver
- Sending and Receiving Messages

Figure 3–2 shows the path of messages for a simple point to point communication using a single queue.

Figure 3–2 JMS Programming Using the Point to Point Model



Creating and Starting a Queue Connection

Once you obtain a queue connection factory, use it to create a QueueConnection and use the `start` method to establish communications with an active Oracle Message Broker. When you first create the QueueConnection, the connection is stopped.

Note: When the QueueConnection is created, it is in the stopped mode. When a connection is stopped, consumers cannot receive messages and calls to receive do not return.

The QueueConnection is an active connection to the JMS provider (Oracle Message Broker). A client uses the QueueConnection to create one or more QueueSessions for producing and consuming messages.

```
QueueConnection queueConnection = null;
queueConnection = queueConnectionFactory.createQueueConnection();
queueConnection.start();
```

The JMS specification contains information that client programs should follow on the preferred use of the `start` method. When the QueueConnection is created, it is in the stopped mode. In general, it is best to leave the connection in the stopped mode while its sessions are being created, and then use the `start` method to start the stopped connection (for more information on conventions for using a session, see section 4.4.6 of the JMS specification).

Getting a Queue Session

After creating the QueueConnection, the client program needs to create a QueueSession. A session is single threaded, therefore concurrent operations are not allowed on a session. The session is used to create destinations for sending or receiving messages. A single connection can support multiple sessions, each maintaining its own state. The Oracle Message Broker supports a feature called Universal Sessions. Universal Sessions support both topic and queue sessions within a single session. See "Universal Connections and Universal Sessions" on page 6-16 for more information on Universal Connections.

The following example shows how to create a session using the `createQueueSession` method:

```
QueueSession queueSession = null;
queueSession = queueConnection.createQueueSession(false,
                                        MercurySession.IMMEDIATE_ACKNOWLEDGE);
```

The first parameter to `createQueueSession` indicates if the session is transacted. The parameter's value is either true or false.

The second parameter to `createQueueSession` indicates the mode of acknowledging message receipt (for more information on message acknowledgment, see section 4.4.13 in the JMS specification and see "Message Listeners and Threads" on page 3-17).

If the session is transacted, the first parameter to `createQueueSession` is true, and the acknowledge mode parameter is ignored. If the session is not transacted,

the first parameter to `createQueueSession` is false, and the acknowledge mode parameter must be the following:

```
MercurySession.IMMEDIATE_ACKNOWLEDGE
```

Working with Queue Destinations - QueueSender and QueueReceiver

Once a session is created, messages can be sent within the session by creating queue senders, and messages can be received by creating queue receivers. A destination must be specified when a queue receiver is created.

Creating a QueueSender

Use the QueueSession `createSender` method to create a QueueSender.

```
QueueSender sender;
sender = session.createSender(queue);
```

The `createSender` method creates an object that can be used to send messages to a queue. The queue administrative object is obtained as described in "Accessing Objects for Point-to-Point Messaging" on page 3-5.

Creating a QueueReceiver

Use the QueueSession method `createReceiver` to create a QueueReceiver for receiving messages.

```
QueueReceiver receiver;
receiver = session.createReceiver(queue);
```

The `queue` parameter is the queue to receive messages from. The queue administrative object is obtained as described in "Accessing Objects for Point-to-Point Messaging" on page 3-5.

Sending and Receiving Messages

Finally, the client program needs to create a message of the desired type and send it to a queue. The client program that is receiving messages also needs to create a message.

Creating and Sending Messages

This example shows how to create and send a text message. The Oracle Message Broker and JMS support several message types in addition to a text message.

```
TextMessage message_out;
message_out = session.createTextMessage();
message_out.setText("Data");
sender.send(message_out);
```

Creating and Receiving Messages

This example shows how to receive a text message.

A client can receive messages either synchronously or have the provider asynchronously deliver the messages as they arrive. For information on asynchronous delivery and the message listener interface, see the JMS specification, section 4.5.2.

The following code shows the use of the `QueueReceiver.receive` method for synchronous delivery:

```
TextMessage message_in;
message_in = session.createTextMessage();
message_in = receiver.receive();
String inData = message_in.getText();
```

Shutting Down

Client programs run in either Local Mode or Remote Mode (Non-Local Mode). A client program in either mode needs to shut down when it finishes. Refer to Chapter 5 for information on Local Mode and Remote Mode clients.

When a client is done with the Oracle Message Broker, it should call `Mercury.shutdownClient()`. This shuts down any Oracle Message Brokers running in Local Mode, and the ORB, if remote Oracle Message Brokers have been accessed, and performs certain cleanup operations. Clients call this method as follows:

```
oracle.oas.mercury.Mercury.shutdownClient()
```

Note: The shutdown methods are not part of the JMS specification. They assist the Oracle Message Broker in cleanup, and allow clients to exit and cleanup.

After the Oracle Message Broker is shut down, keep the following in mind:

- The ORB cannot be restarted after it is shut down. Thus a client should not call `Mercury.shutdownClient()` and then attempt to access a Remote Mode Oracle Message Broker by obtaining a connection factory.

Publish/Subscribe Messaging

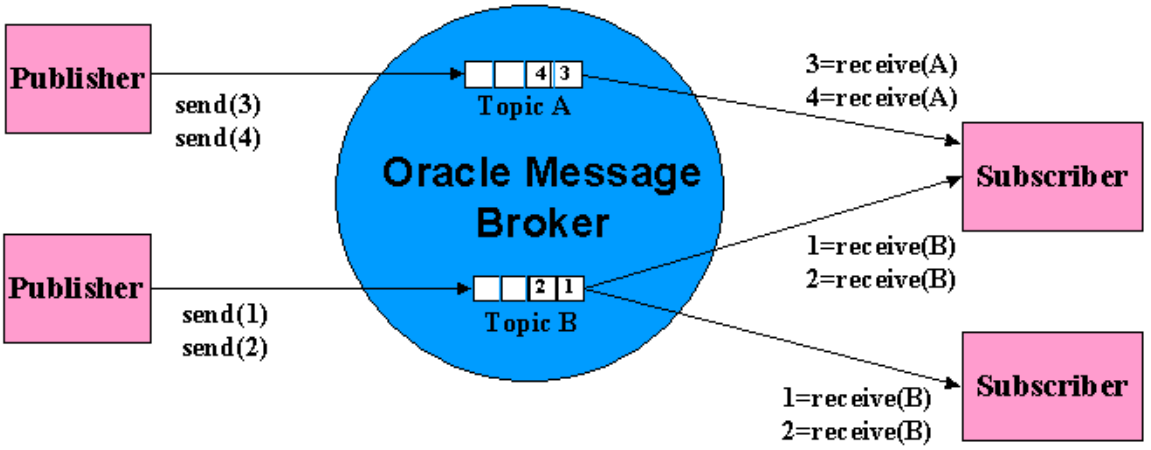
This section describes the programming steps required to publish messages to a topic and subscribe to topics to receive messages using the JMS Publish / Subscribe methods. The following steps assume that the administrator has created the topic connection factory and one or more topics, and that these Oracle Message Broker administrative objects were obtained as described in "Accessing Objects in the Directory" on page 3-4.

This section covers the following:

- Creating and Starting a Topic Connection
- Getting a Topic Session
- Working with Topic Destinations - TopicPublisher and TopicSubscriber
- Publishing and Subscribing

Figure 3–3 shows the path of messages for a simple Publish / Subscribe model.

Figure 3–3 JMS Programming Using a Publisher and a Subscriber



Creating and Starting a Topic Connection

Once you obtain a topic connection factory, use it to create a TopicConnection and use the `start` method to establish communications with the Oracle Message Broker. When you first create the TopicConnection, the connection is stopped.

> **Note:** When the TopicConnection is created, it is in the stopped mode. When a connection is stopped, consumers cannot receive messages and calls to receive do not return.

The TopicConnection is an active connection to the provider. A client uses the TopicConnection to create one or more TopicSessions for producing and consuming messages.

```
TopicConnection topicConnection = null;
topicConnection = topicConnectionFactory.createTopicConnection();
topicConnection.start();
```

The JMS specification contains information that client programs should follow on the preferred use of the `start` method. When the TopicConnection is created, it is in the stopped mode. When a connection is stopped, consumers cannot receive messages. In general, it is best to leave the connection in the stopped mode while its sessions are being created, and then use the `start` method to start the stopped

connection (for more information on conventions for using a session, see section 4.4.6 of the JMS specification.)

Getting a Topic Session

After creating the TopicConnection, the client program needs to create a session. A session is single threaded, therefore concurrent operations are not allowed on a session. Subscribers and Publishers are created within the session. A single connection can support multiple sessions, each maintaining its own state (transactions, publishers, and subscribers).

The following example shows how to create a session using the TopicConnection's `createTopicSession` method:

```
TopicSession topicSession = null;
topicSession = topicConnection.createTopicSession(false,
                                        MercurySession.IMMEDIATE_ACKNOWLEDGE);
```

The first parameter to `createTopicSession` indicates if the session is transacted. The parameter value is either true or false.

The second parameter to `createTopicSession` indicates the mode of acknowledging message receipt (for more information on message acknowledgment, see section 4.4.13 in the JMS specification and see "Message Listeners and Threads" on page 3-17).

If the session is transacted, the first parameter to `createTopicSession` is true, and the acknowledge mode parameter is ignored. If the session is not transacted, the first parameter to `createTopicSession` is false, and the acknowledge mode parameter must be the following:

```
MercurySession.IMMEDIATE_ACKNOWLEDGE
```

Working with Topic Destinations - TopicPublisher and TopicSubscriber

To publish messages to a topic, create a publisher within a session. To receive messages from a topic, create a subscriber within a session.

Creating a TopicPublisher

Use the TopicSession method `createPublisher` to create a TopicPublisher.

```
TopicPublisher publisher;
publisher = session.createPublisher(topic);
```

The `topic` parameter is the topic to send messages to. The topic administrative object is obtained as described in "Accessing Objects for Point-to-Point Messaging" on page 3-5.

Creating a TopicSubscriber

Use the TopicSession method `createSubscriber` to create a TopicSubscriber.

```
TopicSubscriber subscriber;
subscriber = session.createSubscriber(topic);
```

The `topic` parameter is the topic to subscribe to messages. The topic administrative object is obtained as described in "Accessing Objects for Point-to-Point Messaging" on page 3-5.

Publishing and Subscribing

Finally, the client program needs to create a message of the appropriate type and send it to a topic.

Creating and Publishing Messages

This example shows how to create and send a text message.

```
TextMessage message_out;
message_out = session.createTextMessage();
message_out.setText("Data");
publisher.publish(message_out);
```

Subscribing to Topics

A client can receive messages either synchronously or have the provider asynchronously deliver the messages as they arrive. For information on asynchronous delivery and the message listener interface, see section 4.5.2 in the JMS specification.

The following code shows a synchronous delivery using the `TopicSubscriber.receive` method:

```
TextMessage message_in;
message_in = session.createTextMessage();
message_in = subscriber.receive();
String inData = message_in.getText();
```

Shutting Down (Publish/Subscribe)

Client programs run in either Local Mode or Remote Mode (Non-Local Mode). A client program in either mode needs to shut down when it finishes. Refer to Chapter 5 for information on Local Mode and Remote Mode clients.

When a client is done with the Oracle Message Broker, it needs to shut down. The procedure for shutting down is the same for PTP and Publish / Subscribe Messaging. See "Shutting Down" on page 3-12 for more information.

Message Listeners and Threads

This section lists important points to keep in mind when you are using JMS with message listeners.

1. When the Oracle Message Broker is running, the following methods can be called by any thread after a message listener has been registered:
   - `Session.close()` When message listeners have been registered within a session, no listeners will be executed when `Session.close` returns. The implication is that `Session.close` blocks if a listener is executing the `onMessage` method for that session.
   - `MessageConsumer.setMessageListener()` only if the connection is stopped.
2. The `MessageConsumer.close()` method cannot be called after a listener has been registered within the session.
3. All JMS methods, other than those mentioned above, can only be called by the thread executing the message listener. The client-side runtime enforces this restriction when a listener has been registered in that session.

   The following methods must be called by the thread executing the message listener if one has been registered (they cannot be called by any other thread after a message listener has been registered):
   - `Session.rollack()` (can only be called by the listener)
   - `Session.commit()` (can only be called by the listener)

   `Session.close` and `MessageConsumer.setMessageListener` are the only methods that can be called in a session, or in the consumers, producers, and message listeners within that session once a message listener has been registered, if the caller is not executing in the thread that executes the message listeners.

4. As described in the JMS specification [JMS 4.5.2], the result of a MessageListener throwing a *RuntimeException* depends on the session's acknowledgment mode. In the Oracle Message Broker, the following rules apply for *RuntimeException*s.
   - `MecurySession.IMMEDIATE_ACKNOWLEDGE`: the message will not be redelivered. The message is lost. In this mode, message delivery is at-most-once delivery.
   - For a transactional session, the next message for the listener is delivered. The client can then either commit or rollback the Session. A *RuntimeException* does not automatically rollback the session. If the client fails, the Oracle Message Broker performs a rollback for the Session.
5. For a durable subscriber with a message listener, note the following:
   - An unsubscribe cannot be performed when the subscriber is in use.
   - The subscriber is in use until the session is closed. The only way to close a message consumer once a listener has been registered is to close the session.
   - Since the session cannot be used after it has been closed, to unsubscribe, use another session to perform the unsubscribe.
   - The subscription created for a durable subscriber is only cancelled by calling `Session.unsubscribe`. Subscriptions for non-durable subscribers are cancelled when the consumer is closed.

Closing JMS Objects and Death Detection

The Oracle Message Broker has a mechanism to detect JMS connections and sessions that have been closed implicitly (this means the JMS connection is closed by the finalizer rather than by a call to `connection.close` or `session.close`). When a JMS connection is closed, either implicitly or by calling `connection.close`, there is no need to explicitly close the sessions, consumers, or producers that had been created within that JMS connection.

When message listeners have been registered within a session, no listeners will be executed when `Session.close` returns. The implication is that `Session.close` blocks if a listener is executing the `onMessage` method for that session.

`Connection.close` closes each session within that connection. `Connection.close` blocks if a listener is executing within one of the connection's sessions. When `Connection.close` returns, no message listeners will be executed for any listeners that had been set for the connection's sessions.

Oracle Message Broker objects other than JMS connections and sessions do not have a mechanism to detect consumers or producers that have been closed implicitly. When you are programming with Oracle Message Broker, it is important to explicitly close sessions, consumers, and producers using the corresponding close method when you are done with the object. When a session is closed, there is no need to explicitly close the consumers and producers that had been created within that session.

When a JMS connection is closed, explicitly or by the finalizer, Oracle Message Broker frees resources for the JMS connection, and the sessions, consumers, or producers within that JMS connection. Oracle Message Broker does not free resources for consumers or producers that have been garbage collected when the corresponding JMS connection or session has not been garbage collected. Oracle Message Broker detects client processes that have stopped running. If the Oracle Message Broker had allocated resource for the client it will cleanup the resources (rollback transactions, release Database Server connections). Note that the detection mechanism is not instantaneous, it takes some amount of time.

Leaked Resources and Death Detection

Several conditions may affect performance and available resources. For this section, use the following definitions:

- A leaked session is a JMS session for which Session.close has not been called and Connection.close has not been called on the session's connection.
- A leaked connection is a JMS connection for which Connection.close has not been called.

The Oracle Message Broker uses a death detection thread that runs every 20 seconds to find leaked connections and leaked sessions. Leaked connections and leaked sessions will be detected within two intervals (40 seconds). Leaked connections and sessions, and the resources associated with them, are released when the leak is detected.

The death detection thread is run more frequently when any of the following occur:

1. Low memory is detected in the Oracle Message Broker.
2. The number of sessions in AQ Driver approaches maxPrivateSessions (this operation is AQ Driver specific).

3. A consumer attempts to use a durable subscriber name that is in use. This is handled by pinging the current user of the durable subscriber name to detect when a client exits and then restarts and attempts to create a durable subscriber with the same name.

Setting the Message Priority

The Oracle Message Broker sets the message priority after a client invokes the send or publish method on a message. This overwrites any value set by the client using a call to `Message.setJMSPriority()` before the message is sent or published.

Oracle Message Broker sets the priority value for a message as follows:

1. The value of the *priority* argument in the call to send or publish if there is a *priority* argument.
2. The priority value set as the default for the message producer if there is not a *priority* argument in the call to send or publish. The priority for a message producer is set using a call to `MessageProducer.setPriority()`.

Use a call to `Message.setJMSPriority()` to change the priority value for a message that has been received.

4

Administration

The Oracle Message Broker supports two means of specifying configuration information for administration: using an LDAP Directory, and using lightweight configuration. With lightweight configuration, the Oracle Message Broker reads configuration information from a file or from Java properties when it begins its execution. This chapter describes using an LDAP Directory for configuration information. Refer to Chapter 13, "Lightweight Configuration" for details on using a configuration file or Java properties to specify configuration information.

When administrative information is stored in an LDAP Directory, directory entries contain the names and configuration options for Oracle Message Broker administrative objects such as JMS destinations (topics or queues). This chapter introduces the LDAP Directory and describes the directory entries. The chapter also covers the administrative utilities that modify and check directory entries.

When the Oracle Message Broker starts, and it is using the directory, it writes an address into a directory entry. Clients use this address to contact the Oracle Message Broker. In addition, while the Oracle Message Broker is running, the administrative utilities notify it of configuration changes. For example, Oracle Message Broker is notified when administrators change configuration parameters.

This chapter includes the following sections:

What is the Oracle Internet Directory?

Oracle Internet Directory is a directory service implemented as an application on the Oracle 8*i* database. It enables retrieval of information about dispersed users and network resources. It combines Lightweight Directory Access Protocol (LDAP) Version 3, the open Internet standard directory access protocol, with the high performance, scalability, robustness, and availability of the Oracle8*i* Server.

What is a Directory?

A directory is a way of organizing information so that it can be found easily. It lists objects—for example, people, books in a library, merchandise in a department store—and gives details about each one. Information in the directory is stored in a hierarchy of directory entries. A directory entry consists of attributes and their values. The directory contains administrative objects that store Oracle Message Broker configuration options.

What is LDAP?

LDAP (Lightweight Directory Access Protocol) is the emerging Internet standard for directory services. It is based on the earlier ISO X.500 Directory Access Protocol (DAP) standard, but simplifies that standard considerably, allowing LDAP to be more efficient, straightforward, and easier to implement.

Oracle Internet Directory implements Version 3 of LDAP, which was approved as a proposed Internet Standard by the Internet Engineering Task Force (IETF) in December 1997.

The LDAP specification is contained in a number of public documents of the Internet Engineering Task Force called RFCs (Requests for Comments). In particular, LDAP Version 3 is defined in RFCs 2251-2256. These are available on the worldwide web at:

`http://www.ietf.org/rfc`

The building blocks of an LDAP Directory service are called directory entries. Each entry has a unique name, called a distinguished name. Each entry contains information items, called attributes. Attributes are grouped into categories, called object classes.

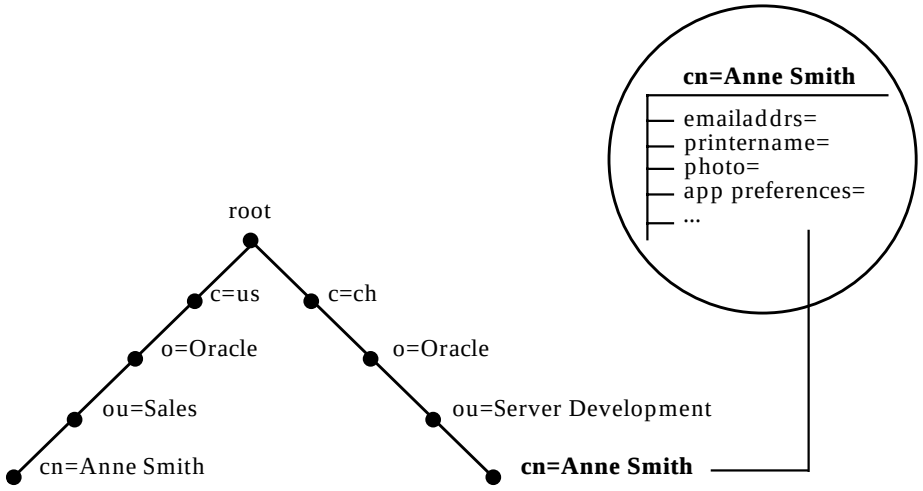
Directory Entries

In a directory, each collection of information about an object is called an entry. For example, a typical telephone directory includes entries for people, and a library card catalog contains entries for books. Similarly, the Oracle Message Broker directory includes entries containing configuration information for Oracle Message Broker components.

Each entry in a directory is uniquely identified by a distinguished name (DN). The distinguished name tells you exactly where the entry resides in the directory's hierarchy represented by the Directory Information Tree (DIT).

To understand the relation between a distinguished name and a DIT, look at Figure 4–1.

Figure 4–1 A Directory Information Tree



The DIT shown in Figure 4–1 is structured along geographical and organizational lines. It diagrammatically represents entries for two employees who have the same common name (cn), namely, Anne Smith.

The DIT branch on the right represents the entry for the Anne Smith who works in an organizational unit (ou) named Server Development, in the organization (o) Oracle, in the country (c) of Switzerland (ch).

The DN for this “Anne Smith” entry is:

cn=Anne Smith,ou=Server Development,o=Oracle,c=ch

Note that the conventional format of a distinguished name places the lowest DIT component at the left, then follows it with the next highest component, and thus moving progressively up to the root.

Within a distinguished name, the lowest component is called the relative distinguished name (RDN). For example, in the above entry for Anne Smith, the RDN is *cn=Anne Smith*. Similarly, the RDN for the entry immediately above Anne Smith’s RDN is *ou=Server Development*, the RDN for the entry immediately above *ou=Server Development* is *o=Oracle*, and so on. A DN is made up of a sequence of RDNs separated by commas.

To locate a particular entry within the overall DIT, a client uniquely identifies that entry by using the entry’s full DN—not simply the RDN. For example, within the global organization in Figure 4–1, there are two employees with the same RDN, namely, *Anne Smith*. To avoid confusion between these two entries, you would use each one’s full DN. If there are potentially two employees with the same name in the same organizational unit, you could use additional mechanisms, such as uniquely identifying each employee with an identification number.

Distinguished Name Format

When you use a DN with the Oracle Message Broker utilities, or in an Oracle Message Broker client program, be aware of the following:

1. A DN is case-insensitive.
2. The RDN portion of a DN can be separated by either a comma, ',' or a semi-colon, ';'.
3. White spaces between the components of a DN are ignored.

   For example, the following DNs are equivalent:

   ```
   cn=sales,cn=oracle,c=us
   cn=sales;cn=oracle;c=us
   cn  = sales; cN=oRacLE;C=US
   ```

4. In a DN, you can escape the following characters using a backslash (\).

"	Quote
,	Comma
:	Colon
+	Plus
=	Equals
;	Semi-colon
\	Backslash
#	Pound
<	Less than
>	Greater than

Attributes

In a typical telephone directory, an entry for a person contains such information items as an address and phone number. In an online directory, such information items are called attributes. Attributes in a typical employee entry could include a job title, e-mail address, and phone number.

In Figure 4–1, the entry for Anne Smith in Switzerland has several attributes, each providing specific information about her. These are listed in the balloon to the right of the tree, and they include *emailaddrs=*, *printername=*, *jpegPhoto=*, and *app preferences=*. Moreover, each bullet in Figure 4–1 is also an entry with attributes, although the attributes for each are not shown.

Each attribute consists of an attribute *type* and one or more attribute *values*. The attribute type identifies the kind of information that the attribute contains—for example, *job title*. The attribute value is the particular instance of information appearing in that entry. For example, in the entry for Anne Smith, the value for the *job title* attribute could be *manager*.

Kinds of Attribute Information

Attributes contain two kinds of information.

Application Information	This information is maintained and retrieved by the directory clients and is unimportant to the operation of the directory. A telephone number, for example, is application information.
Operational Information	This information pertains to the operation of the directory itself. Some operational information is specified by the directory to control the server—for example, the time stamp for an entry. Other operational information, such as access information, is defined by administrators and is used by the directory program in its processing.

Any given attribute can hold either application information, or operational information, but not both.

Single-Valued and Multiple-Valued Attributes

Attributes can be either single-valued or multiple-valued. Single-valued attributes carry only one value in the attribute, whereas multiple-valued attributes can have more than one value. An example of a multiple-valued attribute is a group membership list that includes the names of all the members of a given group, such as an e-mail list.

Object Classes

The Oracle Message Broker defines several directory entries, and assigns object classes to the entries. These object classes define groups of attributes. An object class is a category of objects, and it provides both *mandatory* and *optional* attributes for particular objects.

For example, the *organizationalPerson* object class includes the mandatory attributes commonName (cn) and surname (sn). The Oracle Internet Directory provides standard LDAP object classes, several proprietary object classes, as well as object classes that are added to support the Oracle Message Broker.

Schemas

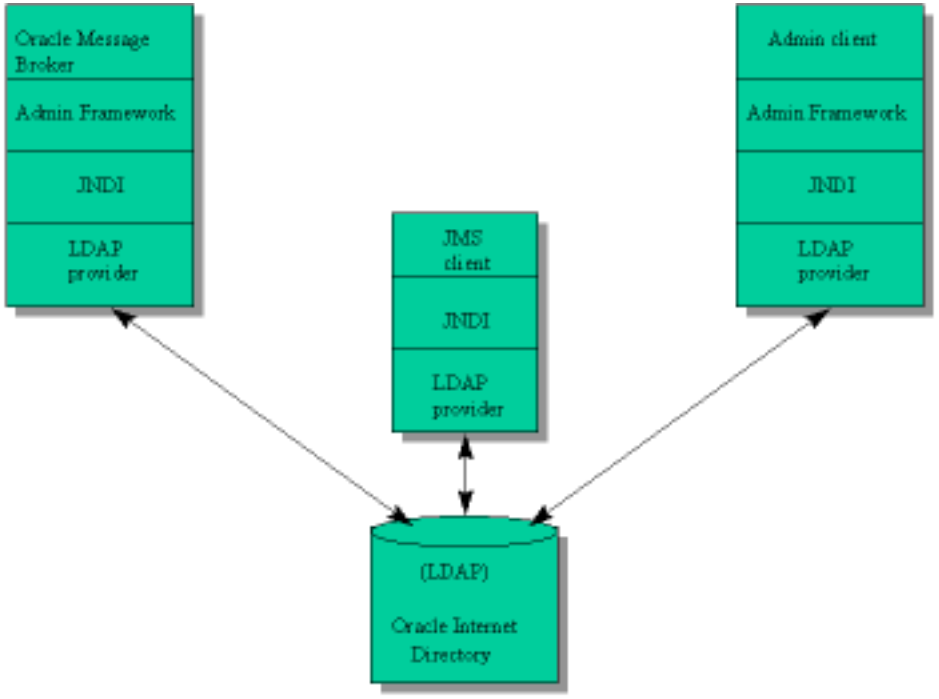
All information about how data is organized in the DIT—that is, metadata such as object classes, attributes, matching rules, and syntaxes—is contained in the directory schema. This information is contained in a special class of entry called a subentry. Oracle Internet Directory holds schema definitions in the subentry called cn=subSchemaSubentry. To provide directory support for the Oracle Message Broker the installation includes scripts that add object classes and attributes to the directory entry cn=subSchemaSubentry.

Accessing LDAP with the Administrative Framework

The Oracle Message Broker and the administrative utilities use the LDAP Directory to read or modify administrative objects. To facilitate use of the directory, the Oracle Message Broker includes a support layer called the administrative framework. The administrative framework provides methods to access entries in the directory, validate DNs, map LDAP attributes to Oracle Message Broker options, and perform a number of other functions. The Oracle Message Broker administrative framework hides LDAP from the administrator and provides significant error checking for directory modifications. Using the administrative framework and the administrative utilities, an administrator only needs to use a few commands and understand distinguished names to perform Oracle Message Broker administration.

The administrative framework uses JNDI to access the directory. Figure 4–2 shows the Oracle Message Broker components and shows how these components use the administrative framework to access the directory.

Figure 4–2 Directory Access with the Administrative Framework



Oracle Message Broker Directory Information Tree

This section describes the structure and the organization of Oracle Message Broker administrative information. This structure is defined when the directory is modified to support the Oracle Message Broker. The steps required to modify the directory are installation tasks (see the *Oracle Message Broker Installation Guide* for details). After the directory is modified, an administrator can use the Oracle Message Broker administrative utilities to create, delete, or modify Oracle Message Broker entries.

Figure 4–3 shows a DIT containing several Oracle Message Broker instance entries, and the entries that are created to support the Oracle Message Broker security service (for information on the security service, see Chapter 12, "Security"). An Oracle Message Broker instance, OMB Instance, contains the administrative objects needed to start or modify the Oracle Message Broker. These entries also allow a JMS client to contact the Oracle Message Broker. Figure 4–4 shows the contents of an OMB Instance. Table 4–1 describes the OMB Instance entries in more detail.

An administrator creates an OMB Instance for each active Oracle Message Broker (Local Mode allows for sharing of OMB Instances. See Chapter 5, "Oracle Message Broker Features" for information on running in Local Mode). An administrator creates multiple OMB Instances to support different working groups in an organization, for improved performance, or to support multiple networks of communicating Oracle Message Brokers using one or more directories.

Figure 4–3, Table 4–1, and Figure 4–4 show fixed container entries in **bold**; the administrator uses the Oracle Message Broker administration utilities to create these entries in the directory with the name exactly as shown. User defined container entries, such as *topic1*, are shown in italics; these entries can have any name that an Oracle Message Broker Administrator chooses, and multiple entries are allowed.

Figure 4–3 Top Level Directory Organization

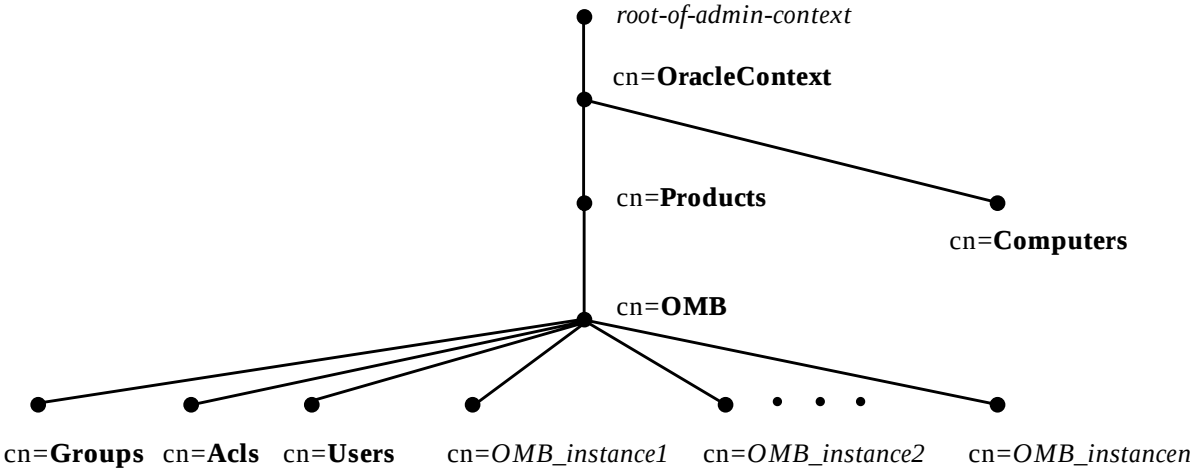


Figure 4–4 OMB Instance Directory Organization

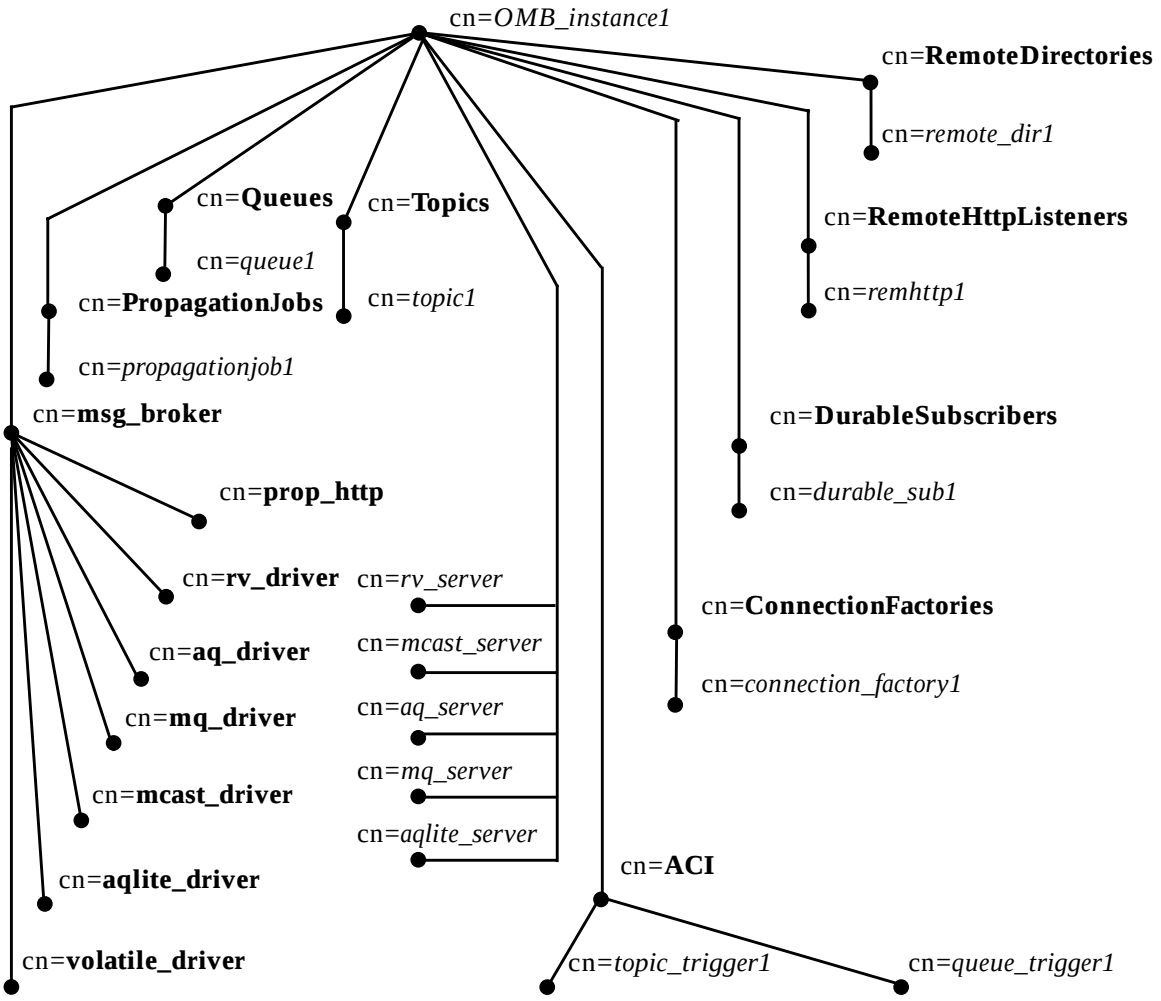


Table 4–1 OMB Instance Entries

Entry	Description
OMB_instance1	This is the top-level of an Oracle Message Broker directory instance. A complete OMB Instance contains all the entries required to use an Oracle Message Broker, or for a JMS client to contact an active Oracle Message Broker.
ACI	A fixed container defining asynchronous component invocation
queuetrigger1	An entry defining an ACI queue trigger
topictrigger1	An entry defining an ACI topic trigger
aq_driver	A driver entry for an Oracle AQ Driver
aq_server	An entry for Oracle AQ Server information. An instance may have 0 or 1 *aq_server* entries.
aqlite_driver	A driver entry for an Oracle AQ Lite Driver.
aqlite_server	An entry for Oracle AQ Lite Server information. An instance may have 0 or 1 *aqlite_server* entries.
ConnectionFactories	A fixed container defining the instance connection factories
connection_factory1	A connection factory entry. At least one connection factory must be defined per driver.
DurableSubscribers	A fixed container defining durable subscribers
durable_sub1	A durable subscriber entry. All attributes for a durable subscriber are read-only-after-create. This means that you can only view the durable subscriber attributes. To modify a durable subscriber, delete the entry and then recreate it.
prop_http	Specifies the HTTP propagation listener options
mcast_driver	A driver entry for an Oracle Multicast Driver
mcast_server	An entry for Oracle Multicast Server information. An instance may have 0 or 1 *mcast_server* entries.
mq_driver	A driver entry for an MQSeries Driver
mq_server	An entry for MQSeries Server information. An instance may have 0 or 1 *mq_server* entries.
msg_broker	A fixed name container for Oracle Message Broker configuration parameters. A **msg_broker** entry contains 1 (one) or more driver entries.
PropagationJobs	A fixed container defining propagation jobs
propagation_job1	A propagation job entry

Table 4–1 (Cont.) OMB Instance Entries

Entry	Description
Queues	A fixed container defining queues
queue1	A queue entry
RemoteDirectories	A fixed container defining remote directories
remote_dir1	A remote directory entry
RemoteHttpListeners	A fixed container defining remote HTTP listeners
remotehttp1	An HTTP remote listener entry
rv_driver	A driver entry for a TIBCO Driver
rv_server	An entry for TIBCO Server information. An instance may have 0 or 1 *rv_server* entries.
Topics	A fixed container defining topics
topic1	A topic entry
volatile_driver	A driver entry for an Oracle Volatile Driver

Oracle Message Broker Configuration

This section covers the options for configuring the Oracle Message Broker and provides tables listing the Oracle Message Broker directory entries and attributes. Refer to "Working with the Administration Utilities" on page 2-2 for configuration examples.

To configure an OMB Instance, you perform two types of actions, creating directory entries and modifying attributes. This section provides information on these actions.

Oracle Message Broker Configuration Roadmap

After the LDAP Directory and the Oracle Message Broker are installed, you can add and configure OMB Instances. Using the Oracle Message Broker `AdminUtil` command, the following checklist shows the Oracle Message Broker configuration steps:

- Create an OMB Instance. This step creates the OMB Instance entry, and several required sub-entries. For more information, see "Creating an Oracle Message Broker Instance" on page 4-14.

- Determine which message servers your instance needs to support and create the message servers. For more information, see "Creating and Configuring Message Servers" on page 4-15.
- Create a message broker entry and add drivers for the message servers you added in the previous step. For more information, refer to "Configuring the Message Broker Entry and Drivers" on page 4-18.
- Add at least one connection factory per driver. See "Creating and Configuring Connection Factories" on page 4-27 for information on adding and configuring connection factories.
- (Optional) Add queues as needed. For information on adding queues, see "Adding Queues" on page 4-28.
- (Optional) Add topics as needed. For information on adding topics, see "Adding Topics" on page 4-31.
- (Optional) Add remote directory entries if you are using remote directories. For information on remote directories, see "Creating and Configuring Remote Directories" on page 4-33.
- (Optional) Add HTTP listener entries and one or more HTTP propagation entries if you are using HTTP for propagation. For more information, see "Creating and Configuring Remote HTTP Listeners" on page 4-34 and "Configuring Message Broker HTTP Propagation Options" on page 4-27.
- (Optional) Add propagation job entries if you are using propagation. For information on propagation, see "Creating and Configuring Propagation Jobs" on page 4-35.
- (Optional) Add durable subscribers if you are using the administration tools to create durable subscribers. Normally JMS clients create and delete durable subscribers, not the administrator. See "Creating and Configuring Durable Subscribers" on page 4-35 for information on durable subscribers.
- (Optional) Add Asynchronous Component Invocation (ACI) Triggers if you are using ACI, see "Creating and Configuring Asynchronous Component Invocation Triggers" on page 4-36.
- Review the values in the new entries to make sure that all attribute values meet your requirements. For more information, see "Showing Directory Attributes and Entries" on page 4-38.

Reserved Internal Attributes

The reserved, internal attributes are common to all Oracle Message Broker entries. These attributes are reserved for internal use or for future use and should never be used, or modified with the Oracle Message Broker administration utilities, or with any other utilities that modify Oracle Message Broker entries. The list of attributes in Table 4–2 applies to all of the following tables: Table 4–3 through Table 4–22.

Table 4–2 Reserved Attributes (for internal use only)

Attribute	Description
`binary`	Reserved for internal use only
`internal`	Reserved for internal use only
`xml`	Reserved for internal use only

Creating an Oracle Message Broker Instance

A valid OMB Instance contains all the directory entries required for an administrator to run an Oracle Message Broker. Each OMB Instance is created with a user defined container name. The OMB Instance container name is supplied when the OMB Instance is created. Refer to Figure 4–4 for a view of the contents of an OMB Instance.

Use the `AdminUtil createombinstance` command to create an Oracle Message Broker instance. The following sections show how to add entries and set attribute values for the entries in the OMB Instance. Refer to "Command-line Administration Utility - AdminUtil" on page 4-41 for information on using `AdminUtil`.

For example, to create the OMB Instance named TestOMB using the `createombinstance` command (the Oracle Message Broker installation tasks create the required Oracle Message Broker directory entry named OMB, as well as other required directory entries).

```
% AdminUtil
> createombinstance TestOMB cn=OMB ;
```

Creating and Configuring Message Servers

Oracle Message Broker server entries provide information about message servers. Message server entries contain configuration options and access information for a supported message storage facility. The Oracle Message Broker supports the following message server types:

- aq_server – supports the Oracle AQ Driver
- aqlite_server – supports the Oracle AQ Lite Driver
- mq_server – supports the IBM MQSeries Driver
- mcast_server – supports the Oracle Multicast Driver
- rv_server – supports the TIBCO Driver

> **Note:** Since the Oracle Message Broker stores messages for the Oracle Volatile Driver internally, the Oracle Volatile Driver does not use a server entry.

The Oracle Message Broker only supports 0 or 1 server entries of a given type within an OMB Instance. The administration utilities do not prevent you from creating multiple server entries. The Oracle Message Broker, at startup, reports the following error if it finds multiple server entries of the same kind in an OMB Instance:

```
Error starting broker, caught BadStateException: Entry defined twice
```

Setting AQ Server Options

An `aq_server` contains the attributes listed in Table 4–3. To configure the AQ Server, create the `aq_server` entry and then set the appropriate attributes.

Table 4–3 AQ Server Attributes

Attribute	Description
`aq_password`	Stores the password required to use AQ on the Oracle 8*i* Database Server. The value stored is encrypted in the directory.
`aq_service_name`	Specifies the service name for the AQ Oracle 8*i* Database Server
`prop_recv_log_queue`	(Optional) Specifies the propagation receive log queue

Table 4–3 (Cont.) AQ Server Attributes

Attribute	Description
prop_send_log_queue	(Optional) Specifies the propagation send log queue
aq_username	Stores the user name required to use Oracle AQ on the Oracle 8*i* Database Server

Note: After installation, the AQ password can be changed using the Database Server Enterprise Manager, or using the Database Server ALTER USER command.

Setting AQ Lite Server Options

This entry specifies the AQ Lite database that is associated with a given AQ Lite Driver, and the password that is needed to access the database.

An aqlite_server contains the attributes listed in Table 4–4. To configure the AQ Lite Server, create the aqlite_server entry and then set the appropriate attributes.

Table 4–4 AQ Lite Server Attributes

Attribute	Description
aqlite_database_name	The AQ Lite database name. When the aqlite_database_name is null the default database is used.
aqlite_database_password	Stores the password required to use AQ Lite in the Oracle 8*i* Lite Database. The value stored is encrypted in the directory.
prop_recv_log_queue	(Optional) The propagation receive log queue
prop_send_log_queue	(Optional) The propagation send log queue

Setting MQSeries Server Options

The mq_server entry contains the attributes listed in Table 4–5. To configure the MQSeries Server, create the mq_server entry and then set the appropriate attributes.

Table 4–5 MQSeries Server Attributes

Attribute	Description
connection_type	Defines the connection type. Standard is the only connection type supported. A standard connection is established with the MQSeries Queue manager.
prop_recv_log_queue	(Optional) The propagation receive log queue
prop_send_log_queue	(Optional) The propagation send log queue
queue_mgr	MQSeries queue manager

Setting Multicast Server Options

The mcast_server entry contains the attributes listed in Table 4–6. To configure the Multicast Server, create the mcast_server entry and then set the appropriate attributes.

Table 4–6 Multicast Server Attributes

Attribute	Description
ip	IP multicast address used by the Oracle Multicast protocol. The value must be between 225.0.0.0 and 239.255.255.255.
port	IP multicast port number used by the Oracle Multicast protocol. This is an integer with a lower limit of 1024 (inclusive) and an upper limit of 65535 (0xFFFF) (inclusive).

Setting TIBCO Server Options

The rv_server entry contains the attributes listed in Table 4–7. To configure the TIBCO Server, create the rv_server entry and then set the appropriate attributes.

Table 4–7 TIBCO Server Attributes

Attribute	Description
service	The name of the Rendezvous service (see the *Rendezvous Administration Guide*, for more information).

Configuring the Message Broker Entry and Drivers

The message broker entry contains attributes that set general parameters for an Oracle Message Broker instance. Also, when you add drivers or HTTP propagation entries, they are added beneath the message broker entry. A HTTP propagation entry (`prop_http`) contains configuration information for the HTTP propagation.

Only one message broker entry is allowed per OMB Instance. The message broker entry, named `msg_broker`, contains the attributes shown in Table 4–8.

This section covers the following Oracle Message Broker directory entries:

- Adding an Oracle AQ Driver
- Adding an Oracle AQ Lite Driver
- Adding A Volatile Driver
- Adding An MQSeries Driver
- Adding A Multicast Driver
- Adding A TIBCO Driver
- Configuring Message Broker HTTP Propagation Options

Table 4–8 Message Broker Attributes

Attribute	Description
`local`	Specifies that the client and server run as a single JVM process. For more information, see "Running in Local Mode", in Chapter 5. The value is a boolean. Default value: false
`max_concurrent_reqs`	Limits the number of concurrent blocking receives to the specified value. The Oracle Message Broker throws a BusyException when a blocking receive request exceeds this limit. A blocking receive is a receive that blocks for at least one (1) second. The JMS specification provides three types of receive methods: blocking, timeout, and non-blocking. Therefore, a BusyException is thrown for blocking receives and timeout receives, when the timeout is 1 second or more and receives are attempted after the specified limit is reached. This option also sets the number of threads that the ORB uses to handle execute method invocations. The value must be an integer. The lower limit is 10. Default value: 10.
`max_memory`	Heap memory allocated in Megabytes. For more information see, "Oracle Message Broker Memory Management" on page 4-19. The value must be an integer. The lower limit is 1. Default value: 16.

Table 4–8 (Cont.) Message Broker Attributes

Attribute	Description
propagation_http_handlers	Specifies the number of HTTP propagation handler threads to start. The value must be an integer. Default value: 1
propagation_recv_threads	Number of receive threads for propagation. Increasing the number of receive threads allows the propagation manager to handle more propagation receives concurrently. See Chapter 8 for information on propagation. The value must be an integer. The lower limit is 0 and the upper limit is 100. Default value: 1.
propagation_send_threads	Number of send threads for propagation. Increasing the number of send threads allows the propagation manager to handle more propagation sends concurrently. See Chapter 8 for information on propagation. The value must be an integer. The lower limit is 0 and the upper limit is 100. Default value: 1.

Oracle Message Broker Memory Management

The Oracle Message Broker uses the JVM heap to maintain its state, to store Volatile Driver messages, control consumers, and to manage producers, sessions, and connections. Any operation that uses resources can cause an exception when there is not enough heap memory available.

The `max_memory` attribute determines the amount of heap memory to use in the JVM where the Oracle Message Broker runs. However, setting this value is just one factor in JVM memory management. Keep the following points in mind when running the Oracle Message Broker:

- Javasoft JVMs run with a maximum heap size. A default value is used when a value is not specified on the command line. When you start Oracle Message Broker using the `MsgBroker` command, the value set on the command line with the –heap option specifies the maximum heap size.
- When the Oracle Message Broker is running, it does not allow `max_memory` to be updated to a value that is less than the amount of memory currently in use (as set with the –heap option on the `MsgBroker` command line).
- JVM does not allow the maximum heap size to be queried. The `MsgBroker` command sets a property when starting the JVM. This property holds the value of the maximum heap size. The Oracle Message Broker uses the property to determine if new values for `max_memory` are valid.

- If the value for the `max_memory` attribute is equal to the value of the `–heap` option in the `MsgBroker` command, the `max_memory` attribute cannot be increased at run-time, unless it is first decreased.
- If you start the Oracle Message Broker using Java on the command line, without using `MsgBroker` command, make sure the following relationships are maintained between the JVM's initial heap size, the maximum heap size, and the Oracle Message Broker's message broker `max_memory` attribute:
  - The `max_memory` attribute for the Oracle Message Broker must be set to a value less than or equal to the JVM maximum heap size.
  - The Oracle Message Broker does not allow the value of `max_memory` to be changed to a value greater than the maximum heap size for the JVM.
  - The value set for the JVM initial heap size must equal the JVM maximum heap size setting.

The Oracle Message Broker forces garbage collection prior to throwing memory exceptions. It is also possible for the JVM to throw an exception or fail if attempts to use more memory than is currently available in the JVM.

For example, an application that creates 10,000 Sessions may cause the Oracle Message Broker to throw an exception after creating 5,689 sessions, if no additional heap memory is available and garbage collection does not free sufficient memory.

Adding an Oracle AQ Driver

An aq_driver entry contains configuration information for the AQ Driver, including the DN for the AQ Server. Table 4–9 describes the Oracle AQ Driver attributes. To add an AQ Driver, set up the AQ Server entry and then the AQ Driver attributes.

Table 4–9 AQ Driver Administrative Attributes

Attribute	Description
max_private_sessions	This is the maximum number of concurrent JMS sessions that the driver allows to be created. The Database Server should be configured to support more than max_private_sessions connections. The value must be an integer. The lower limit is 1.
max_shared_sessions	The number of JDBC connections that are created in a pool and used internally for administrative purposes. Administrative uses include creating and destroying AQ queues and subscriptions. The AQ Driver in the OCI mode creates one connection per JMS session to provide operational access to AQ queues. The AQ Driver in the JDBC mode creates one JDBC connection per JMS session to provide operational access to AQ queues. The value must be an integer. The lower limit is 1.
propagation_recv_sessions	The number of receive sessions started for propagation. The sum of propagation_recv_sessions + propagation_send_sessions must be less than max_private_sessions. Default value: 2
propagation_send_sessions	The number of send sessions started for propagation. The sum of propagation_recv_sessions + propagation_send_sessions must be less than max_private_sessions. Default value: 2
push_sessions	The number of threads which will push messages to message listeners. The value must be an integer. The lower limit is 1.
query_interval	The number of milliseconds at which AQ is queried, polled, to determine if messages are available for message listeners. The value must be an integer. The lower limit is 1.
server_dns	The DN pointing to the server that supports the AQ Driver. The type of the DN that this references must be aq_server.
thin_jdbc	If true, use the thin JDBC based AQ driver. If false, use the OCI based JDBC AQ Driver. Default value: false

Table 4–9 (Cont.) AQ Driver Administrative Attributes

Attribute	Description
thin_jdbc_conn_string	The URL suffix to use for the JDBC thin connection to AQ.
use_jdbc	If true, use JDBC based AQ Driver. If false, use the OCI based AQ Driver. See "JDBC Mode" and "OCI Mode" on page 7-8 for details on AQ Driver Modes. Default value: false

Adding an Oracle AQ Lite Driver

A aqlite_driver entry contains configuration information for the Oracle AQ Lite Driver. Table 4–10 describes the driver entry attributes.

Table 4–10 AQ Lite Driver Administrative Attributes

Attribute	Description
max_private_sessions	The AQLite Driver ignores this value. The value must be an integer. The lower limit is 1.
max_shared_sessions	The AQLite Driver ignores this value. The value must be an integer. The lower limit is 1.
query_interval	The interval, in milliseconds, at which the AQ Lite Driver is polled to determine if messages are available for message listeners. The value must be an integer. The lower limit for this is 1.
push_sessions	The number of threads which will push messages to message listeners. The value must be an integer. The lower limit is 1.
server_dns	The dn pointing to the server that supports the AQ Lite Driver. The type of the dn that this references must be aqlite_server.

Adding A Volatile Driver

A volatile_driver entry contains configuration information for the Oracle Volatile Driver. Table 4–11 describes the driver entry attributes.

When using a Volatile Driver, set the msg_broker entry attribute max_memory to an appropriate value to support both the Oracle Message Broker, and the Volatile Driver. This attribute determines the total memory available to the JVM running the Oracle Message Broker. Messages stored using the Volatile Driver consume memory in the active Oracle Message Broker. The amount of memory consumed per

message depends on the message type, and the size of the message. When a message is consumed, the memory associated with the message is freed. The Volatile Driver stops accepting messages when the JVM running the Oracle Message Broker does not have enough free memory.

Since the Oracle Message Broker stores messages for the Volatile Driver internally, the Oracle Volatile Driver does not use an external message server for storing messages and the volatile_driver entry does not contain a `server_dns` attribute (all other driver entries use the `server_dns` attribute to point to the message server).

Table 4–11 Volatile Driver Administrative Attributes

Attribute	Description
`max_private_sessions`	The Volatile Driver ignores this value. The value must be an integer. The lower limit is 1.
`max_shared_sessions`	The Volatile Driver ignores this value. The value must be an integer. The lower limit is 1.
`propagation_recv_sessions`	The number of receive sessions started for propagation. Default value: 2
`propagation_send_sessions`	The number of send sessions started for propagation. Default value: 2
`push_sessions`	The number of threads which will push messages to message listeners. The value must be an integer. The lower limit for this is 1.
`query_interval`	The number of milliseconds at which the Volatile Driver is polled to determine if messages are available for message listeners. The value must be an integer. The lower limit for this is 1.

Adding An MQSeries Driver

An mq_driver entry contains configuration information for the MQSeries Driver. Table 4–12 describes the MQSeries Driver attributes.

Table 4–12 MQSeries Driver Administrative Attributes

Attribute	Description
`max_private_sessions`	The maximum number of connections to the queue manager for the driver. Each JMS session consumes one connection. If the Oracle Message Broker is running, and this attribute is updated, a new value that is less than the old value causes the following behavior. If the current number of sessions in use exceeds the new maximum value, all existing sessions are allowed to continue but no new sessions can be created until the number in use drops below the new maximum value. The value must be an integer. The lower limit is 1.
`max_shared_sessions`	The maximum number of shared sessions for the driver. The value must be an integer. The lower limit is 1.
`query_interval`	The number of milliseconds at which the MQSeries Driver is polled to determine if messages are available for message listeners. The value must be an integer. The lower limit is 1.
`propagation_recv_sessions`	The number of receive sessions started for propagation. The sum of propagation_recv_sessions + propagation_send_sessions must be less than max_private_sessions. Default value: 2
`propagation_send_sessions`	The number of send sessions started for propagation. The sum of propagation_recv_sessions + propagation_send_sessions must be less than max_private_sessions. Default value: 2
`push_sessions`	The number of threads which will push messages to message listeners. The value must be an integer. The lower limit is 1.
`server_dns`	The DN pointing to the server that supports the MQSeries Driver. The value must be a DN that points to an entry of the type mq_server.

Adding A Multicast Driver

The mcast_driver entry contains configuration information for the Oracle Multicast Driver. Table 4–13 describes the Multicast Driver attributes.

Table 4–13 Multicast Driver Administrative Attributes

Attribute	Description
`max_private_sessions`	The Multicast Driver ignores this value. The value must be an integer. The lower limit is 1.
`max_shared_sessions`	The Multicast Driver ignores this value. The value must be an integer. The lower limit is 1.
`network`	The Multicast network interface
`propagation_recv_sessions`	The number of receive sessions started for propagation. Default value: 2
`propagation_send_sessions`	The number of send sessions started for propagation. Default value: 2
`push_sessions`	The number of threads which will push messages to message listeners. The value must be an integer. The lower limit is 1.
`query_interval`	The number of milliseconds at which the Multicast Driver is polled to determine if messages are available for message listeners. The value must be an integer. The lower limit is 1.
`server_dns`	The DN pointing to the server that supports the Multicast Driver. The value must be a valid DN that points to an entry of type mcast_server.

Adding A TIBCO Driver

The rv_driver entry contains configuration information for the TIBCO Driver. Table 4–14 describes the driver entry attributes.

Table 4–14 TIBCO Driver Administrative Attributes

Attribute	Description
daemon	Rendezvous daemon used by the Oracle Message Broker (see the *Rendezvous Administration Guide*, for more information).
max_private_sessions	The maximum number of JMS sessions for the driver. The value must be an integer. The lower limit is 1.
max_shared_sessions	The TIBCO Driver ignores this value. The value must be an integer. The lower limit is 1.
network	Network interface used by Rendezvous (see the *Rendezvous Administration Guide*, for more information).
propagation_recv_sessions	The number of receive sessions started for propagation. The sum of propagation_recv_sessions + propagation_send_sessions must be less than max_private_sessions. Default value: 2
propagation_send_sessions	The number of send sessions started for propagation. The sum of propagation_recv_sessions + propagation_send_sessions must be less than max_private_sessions. Default value: 2
push_sessions	The number of threads which will push messages to message listeners. The value must be an integer. The lower limit is 1.
query_interval	The number of milliseconds at which the TIBCO Driver is polled to determine if messages are available for message listeners. The value must be an integer. The lower limit is 1.
server_dns	The DN pointing to the server that supports the TIBCO Driver. The value must be a valid DN of type rv_server.

Configuring Message Broker HTTP Propagation Options

A prop_http entry contains configuration information for the HTTP propagation. Table 4–15 describes the HTTP propagation entry attributes.

Table 4–15 Propagation with HTTP Administrative Attributes

Attribute	Description
`http_host`	Specifies the name of the system that the listener runs on. This needs to be set if the system is a multi-homed system (a system with multiple domain names). The value is a String. Default value: null
`http_port`	Specifies the port that the HTTP listener to listen on. The value must be an integer. Default value: none
`http_ssl_level`	Specifies the SSL level used to secure the HTTP connections. The value must be an integer. Valid values are: 0, 1, 2, or 3. Default value: 0
`wallet_location`	Specifies the wallet location. This is the full path name of the wallet file. Note: this requires exported wallets. The value is a String. Default value: null
`wallet_password`	Specifies the wallet password to be applied to the specified wallet file. The value stored is encrypted in the directory. The value is a String. Default value: null

Creating and Configuring Connection Factories

The ConnectionFactories entry should contain at least one connection factory for each driver. A connection factory entry specifies the driver type, the provider (this is the DN of the associated message broker entry, and other information that allows a client to connect to the OMB Instance. Table 4–16 describes connection factory attributes.

Table 4–16 Connection Factory Administrative Attributes

Attribute	Description
`client_id`	Used to set the client ID for JMS connection.
`driver_type`	Type of driver associated with the connection factory. Valid types are: `aq`, `aqlite`, `vol`, `mq`, `mcast`, `rv`.

Table 4–16 (Cont.) Connection Factory Administrative Attributes

Attribute	Description
`priority`	The value must be an integer. The lower limit is 0 and the upper limit is 9.
`provider_dns`	This is reserved for future use. Do not set this attribute. If you do set this attribute, results are unpredictable.
`transaction_timeout`	Transaction timeout in milliseconds. The value must be an integer. The lower limit is 1.

Adding Queues

The Queues entry contains all of the OMB Instance queue attributes. Queue attributes define a queue's name and the queue's configuration information. Table 4–17 describes the queue attributes.

Table 4–17 Queue Administrative Attributes

Attribute	Description
`aqlite_message_grouping`	This attribute applies only if the queue is an AQ Lite queue. This specifies the message grouping for the AQ Lite queue. The valid values are: NONE, or TRANSACTIONAL. Default value: NONE
`aqlite_owner`	This attribute applies only if the queue is an AQ Lite queue. This specifies the owner of the AQLite queue. If left blank, the default is used.
`aqlite_storage_clause`	This attribute applies only if the queue is an AQ Lite queue. This specifies the storage clause used to create the AQ Lite queue table.
`aq_adt`	This attribute applies only if the queue is an AQ Queue. This determines the JMS ADT to use for storing messages to AQ queues. See "AQ Messages" on page 7-5 for more information. The valid values are: text, bytes, map, stream, object, all, queriable, raw Default value: raw
`aq_schema`	This attribute applies only if the queue is an AQ Queue. This specifies the schema in which the AQ queue resides.
`aq_rules`	This attribute applies only if the queue is an AQ Queue. If true, use AQ message selection, if false, use Oracle Message Broker message selection. The value is a boolean. Default value: false

Table 4–17 (Cont.) Queue Administrative Attributes

Attribute	Description
acl_dn	A DN pointing to an Acl entry. Refer to Chapter 12, "Security" for more details.
create_provider_q	This attribute is only used for Oracle AQ queues or Oracle AQ Lite queues. Possible values are yes, no, or conditional. This attribute affects how Oracle AQ queues or Oracle AQ Lite queues are created when the administrator creates an LDAP Directory entry corresponding to the AQ queue or the AQ Lite queue using the administrative utility. ■ yes: the administrative utility creates underlying queue. ■ no: the administrative utility does not access the provider. ■ conditional: if the underlying queue already exists in AQ or AQLite, it is used, otherwise the queue is created. Default value: conditional.
is_managed	This is only used for MQSeries queues. Determines if the queue is managed.
is_native	Determines if the queue stores JMS messages or native messages. The value is a boolean.
is_queriable	This is only used for Oracle AQ queues. Determines if the queue is queriable. Note: use of this attribute has been deprecated. Please do not use this attribute. Equivalent functionality is provided with the aq_adt attribute. Default value: false
max_messages	Sets the maximum number of messages in the queue. Note: the AQ Driver and the MQSeries Driver do not support this attribute. The value must be an integer. The lower limit for this attribute is 1.
provider_queue_name	This is only used for Oracle AQ and MQSeries queues. Determines the provider's name for the queue, if different from the cn. If not defined, the default provider_queue_name is the same as the topic or queue name (the cn). See "Notes and Limitations for Configuring Queues" on page 4-30 for additional information on this attribute.
provider_q_created	This attribute is only used for Oracle AQ or Oracle AQ Lite queues. This is a read only attribute. If the administrative utility creates the Oracle AQ queue or the Oracle AQ Lite queue, this is set to true otherwise this is set to false. The value is a boolean.

Table 4–17 (Cont.) Queue Administrative Attributes

Attribute	Description
rm_provider_q	This attribute is only used for Oracle AQ queues or Oracle AQ Lite queues. Possible values are yes, no, or conditional. This attribute affects how the underlying AQ or AQ Lite queues are removed when the administrator removes an LDAP Directory entry corresponding to the queue using the administrative utility. ■ yes: the queue is removed from when the entry is removed from the LDAP Directory. ■ no: the administrative utility does not access the provider and the underlying AQ or AQ Lite queue is not removed. ■ conditional: the queue is only removed if the Oracle Message Broker administrative utilities created the queue. Default value: conditional.
server_dn	DN of the server that handles this queue. The value must be a valid DN. The DN supplied must point to an entry of one of the following types: msg_broker, aq_server, aqlite_server, mq_server, mcast_ server, or rv_server.

Notes and Limitations for Configuring Queues

Normally, when setting up Queues, destinations should use unique underlying queues on the message store. Thus, the administrator should make sure that the value supplied for the provider_queue_name is unique across all OMB Instances configured to use the Oracle Message Broker or that queue names are unique.

However, the following configuration example provides a case where the previous comment is not applicable. An application could require that an AQ single-consumer queue be accessed as a JMS queue, and the deployment options could demand that several Oracle Message Broker applications accessing the same underlying queue use different Database Server credentials (including the username and password). In the Oracle Message Broker, the Database Server credentials are associated with an OMB Instance's AQ Server Entry, when using the OCI based AQ Driver (see Table 4–3). In this case, the administrator needs to create several separate OMB Instances to access the underlying AQ queue using the Oracle Message Broker, and the administrator needs to configure the multiple OMB Instances using a different username and password in the AQ Server entry. With this configuration, the different OMB Instances have at least one queue entry in the directory that points to the same underlying provider queue (AQ single-consumer queue).

Adding Topics

The Topics entry contains all of the OMB Instance topic entries. Each topic entry specifies attributes defining a topic name and the topic configuration information. Table 4–18 describes the topic attributes.

Table 4–18 Topic Administrative Attributes

Attribute	Description
`aqlite_message_grouping`	This attribute applies only if the topic uses an AQ Lite server. Specifies the message grouping for the AQ Lite queue. The valid values are: NONE, or TRANSACTIONAL. Default value: NONE
`aqlite_owner`	This attribute applies only if the topic uses an AQ Lite server. Specifies the owner of the AQLite queue. If left blank, the default is used.
`aqlite_storage_clause`	This attribute applies only if the topic uses an AQ Lite server. Specifies the storage clause used to create the AQ Lite queue table.
`aq_adt`	This attribute applies only if the topic uses an AQ server. This determines the JMS ADT to use for storing messages to AQ queues. See "AQ Messages" on page 7-5 for more information. The valid values are: text, bytes, map, stream, object, all, queriable, raw Default value: raw
`aq_schema`	This attribute applies only if the topic uses an AQ server. Specifies the schema in which the AQ queue resides.
`aq_rules`	This attribute applies only if the topic uses an AQ server. If true, use AQ message selection, if false, use Oracle Message Broker message selection. The value is a boolean. Default value: false
`acl_dn`	A DN pointing to an Acl entry. Refer to Chapter 12, "Security" for more details.
`create_provider_q`	This is only used for Oracle AQ or AQ Lite topics. Possible values are yes, no, or conditional. This attribute affects how Oracle AQ or AQ Lite queues are created when the administrator creates an LDAP Directory entry corresponding to the AQ or AQ Lite topic using the administrative utility. ■ yes: the administrative utility creates the underlying queue. ■ no: the administrative utility does not access the underlying queue. ■ conditional: if the queue already exists, it is used, otherwise, it is created. Default value: conditional.

Table 4–18 (Cont.) Topic Administrative Attributes

Attribute	Description
is_managed	This is reserved for future use.
is_native	Determines if the topic stores JMS messages or native messages
is_queriable	This is only used for Oracle AQ topics. Determines if the topic is queriable. The value is a boolean. Default value: false Note: use of this attribute has been deprecated. Please do not use this attribute. Equivalent functionality is provided with the aq_adt attribute.
max_messages	Sets the maximum number of messages in the topic. Note: the AQ Driver and the MQSeries Driver do not support this attribute. The value supplied must be an integer. The lower limit for this attribute is 1.
provider_queue_name	Determines the provider's name for the queue that stores messages for the topic. For Oracle AQ topics, if this attribute is not defined, the default provider_queue_name is the same as the supplied cn for the topic. See "Notes and Limitations for Configuring Topics" on page 4-33 for additional information on this attribute.
provider_q_created	This attribute is only used for Oracle AQ or AQ Lite topics. This is a read only attribute. If the administrative utility creates the Oracle AQ or AQ Lite topic this is set to true otherwise this is set to false. The value is a boolean.
rm_provider_q	This attribute is only used for Oracle AQ or AQ Lite topics. Possible values are yes, no, or conditional. This attribute affects how Oracle AQ or AQ Lite queues are removed when the administrator removes an LDAP Directory entry corresponding to the topic using the administrative utility. ■ yes: the queue is removed from Oracle AQ or AQ Lite when the topic is removed from the LDAP Directory. ■ no: the administrative utility does not access Oracle AQ or AQ Lite. ■ conditional: the queue is only removed if it was created by the Oracle Message Broker administrative utilities. Default value: conditional.
server_dn	DN of the server that handles this topic.

Notes and Limitations for Configuring Topics

Normally, when setting up topics, destinations should use unique underlying queues (or topics) on the message store. Thus, the administrator should make sure that the value supplied for the provider_queue_name is unique across all OMB Instances configured to use the Oracle Message Broker, or that topic names are unique.

However, the following configuration example provides a case where the previous comment is not applicable. An application could require that an AQ multi-consumer queue be accessed as a JMS topic. This topic could be required by the application to use an AQ rules engine to specify a message selector. The application could also require access to the topic that does not use the AQ rules engine to specify a message selector. The configuration property that determines whether the AQ rules engine is used for durable subscriber message selectors is associated with the topic entry in the directory. In this case, the administrator would set up two directory entries within the OMB Instance that point to the same underlying AQ multi-consumer queue. One entry would have the `aq_rules` attribute set to true, and the other would have the `aq_rules` attribute set to false.

Creating and Configuring Remote Directories

The RemoteDirectories entry contains remote directory entries. Table 4–19 describes the remote directory entry attributes. Refer to Chapter 8 for more information on remote directories and their use with propagation.

Table 4–19 Remote Directories Administrative Attributes

Attribute	Description
`remote_directory_host`	Specifies the name of the host where the LDAP server runs.
`remote_directory_password`	Specifies the password used for authentication on the LDAP Directory. The value stored is encrypted in the directory.
`remote_directory_port`	Specifies the port used to connect to the LDAP Directory. The value must be an integer. Default value: 389.
`remote_directory_username`	Specifies the DN of the user entry that is used for authentication on the LDAP Directory.

Creating and Configuring Remote HTTP Listeners

The RemoteHTTPListener entry contains remote HTTP entries. Table 4–20 describes the remote HTTP entry attributes. Refer to Chapter 8 for more information on remote HTTP listeners and their use with propagation.

Table 4–20 RemoteHTTPListener Administrative Attributes

Attribute	Description
`proxy_host`	Specifies the hostname for the proxy server (on the sending side). The value is a String. Default value: null
`proxy_port`	Specifies the port for the proxy server (on the sending side). The value must be an integer. Default value: null
`remote_http_host`	Specifies the hostname of the system that the remote HTTP listener runs on. The value is a String. Default value: none
`remote_http_port`	Specifies port number that the HTTP listener on the remote system listens on. The value must be an integer. Default value: no default
`remote_http_path`	Specifies the path of the propagation servlet when using servlet based HTTP propagation. This is the path component of the URL of the HTTP propagation servlet. For example, the attribute `remote_http_path` should be set to /servlet/PropHttpServlet, if the servlet's URL is: `http://company.com/servlet/PropHttpServlet` The value is a String. Default value: null
`remote_http_ssl_level`	Specifies the SSL level of the remote HTTP listener. The valid values are: 0, 1, 2, or 3. The value must be an integer. Default value: 0
`remote_wallet_location`	The full pathname of the wallet file on the sending site. The wallet file must be in "exported file" format. The value is a String. Default value: null
`remote_wallet_password`	The password applied to the specified wallet file. The value stored is encrypted in the directory. The value is a String. Default value: null

Creating and Configuring Propagation Jobs

The PropagationJobs entry contains propagation job configuration information, including the specification of the destinations, source and target, for the propagation job. Table 4–21 describes the propagation job attributes. Refer to Chapter 8 for more information on propagation, and for information on setting attributes in a PropagationJobs entry.

Table 4–21 Propagation Jobs Administrative Attributes

Attribute	Description
activation_state	Specifies the activation state of the propagation job. This attribute specifies whether the propagation job is activated or deactivated. See Activation State in "Propagation Job Configuration" on page 8-16 for details. When the value is true, the propagation job is activated. When the value is false, the propagation job is deactivated. The value is a boolean. Default value: false
create_timestamp	This is reserved for internal use only.
propagation_msg_selector	Contains the propagation message selector string. See Propagation Message Selector "Propagation Job Configuration" on page 8-16 for details.
propagation_password	Contains the password to use for the specified username. The value stored is encrypted in the directory.
propagation_source	DN of the source queue or topic.
propagation_target	DN of the target queue or topic.
propagation_timeout	The timeout value in seconds. Default value: 120
remote_dn	DN of the remote entry. This is used when the propagation_target is specified in a remote directory or using a remote_http_entry.
propagation_username	Contains the username DN to use to access the remote Oracle Message Broker when the security service authentication is enabled.
valid_status	This is reserved for internal use.

Creating and Configuring Durable Subscribers

The DurableSubscribers entry contains durable subscribers. Table 4–22 describes durable subscriber attributes.

When you create or delete a durable subscriber entry for an AQ topic, the create or delete results in the creation or deletion of a subscriber in AQ. This allows durable

subscribers to be created and messages accumulated for the AQ topic before the durable subscriber is available.

Table 4–22 Durable Subscriber Administrative Attributes

Attribute	Description
`aqlite_address`	This attribute applies only if the topic uses an AQ Lite server. Address of the AQ Lite subscriber.
`aqlite_protocol`	This attribute applies only if the topic uses an AQ Lite server. Protocol used by the AQ Lite subscriber.
`aqlite_rule`	This attribute applies only if the topic uses an AQ Lite server. Rule use to filter messages to the AQ Lite subscriber.
`client_id`	This attribute contains the client identifier of the client that registered the durable subscription (see [*JMS 4.3.2*] for more information on client identifiers.)
`jms_user`	JMS user
`msg_selector`	Message selector for the user
`topic_dn`	The DN for the topic. The value must be a valid DN.

Creating and Configuring Asynchronous Component Invocation Triggers

The ACI container contains entries for ACI topic triggers and ACI queue triggers. An ACI trigger entry defines the condition under which a notification is sent to an EJB. Each trigger acts on one and only one destination (queue or topic). It is not possible to set triggers that involve multiple destinations. There are two types of triggers, Queue Triggers and Topic Triggers. Table 4–23 and Table 4–24 describe the attributes for these two types of triggers.

Table 4–23 ACI Queue Trigger Administrative Attributes

Attribute	Description
`authentication`	Specifies the database connection authentication method.
`cf`	The JNDI name of the connection factory. The value is a String.
`component`	The JNDI name of the EJB component to invoke. The value is a String.
`concurrency`	The number of ACIs that can execute concurrently for the same event. The value must be an integer. Default value: 0

Table 4–23 (Cont.) ACI Queue Trigger Administrative Attributes

Attribute	Description
dest	The JNDI name of the destination.
enabled	Activates or deactivates the ACI trigger. The value is a boolean. Default value: false
password	The database schema password.
retries	The number of retries to perform when the ACI does not complete normally. The value must be an integer. Default value: 0
selector	The message selector associated with the trigger. The value is a String.
threshold	The trigger is generated when messages reach a multiple of the threshold. The value must be an integer. Default value: 1
username	The database schema name.

Table 4–24 ACI Topic Trigger Administrative Attributes

Attribute	Description
authentication	Specifies the database connection authentication method.
cf	The JNDI name of the connection factory. The value is a String.
component	The JNDI name of the EJB component to invoke. The value is a String.
enabled	Activates or deactivates the ACI trigger. The value is a boolean. Default value: false
password	The database schema password
retries	The number of retries to perform when the ACI does not complete normally. The value must be an integer. Default value: 0

Table 4–24 (Cont.) ACI Topic Trigger Administrative Attributes

Attribute	Description
`subscription`	The subscription name for topics. The value is a String.
`threshold`	The trigger is generated when messages reach a multiple of the threshold. The value must be an integer. Default value: 1
`username`	The database schema name.

Showing Directory Attributes and Entries

You can use either `AdminUtil`, or the Oracle Message Broker Manager to view the contents of the directory and verify that the entries you created are in the directory. The command to start the Manager is:

```
% ombadmin
```

When you navigate to your OMB Instance, you should verify the entries you created in the previous steps.

To use `AdminUtil` to view directory entries, refer to the `show` command in the section, "Command-line Administration Utility - AdminUtil" on page 4-41.

Dynamic Configuration

When the Oracle Message Broker is running, the set of valid administration operations is restricted. This section describes the operations allowed, and the restrictions on dynamic configuration.

This section covers the following:

- Create Entry Restrictions
- Update Entry Restrictions
- Delete Entry Restrictions

Create Entry Restrictions

The following list contains the entries that cannot be created when the Oracle Message Broker is active. All other types of entries can be created when the Oracle Message Broker is active.

```
aqlite_driver       mcast_server
aqlite_server       mq_driver
aq_driver           mq_server
aq_server           rv_driver
prop_http           rv_server
mcast_driver        volatile_driver
```

Update Entry Restrictions

The following list shows the entries that cannot be updated in any way when the Oracle Message Broker is active.

```
aq_server           mq_server
mcast_server        rv_server
```

Table 4–25 lists the attributes for the specified entries that can be updated when the Oracle Message Broker is running. Other attributes for these entries are not allowed to be updated while the Oracle Message Broker is active.

Table 4–25 Allowed Update Operations for an Active Oracle Message Broker

Entry	Attributes that can be updated
aq_driver	query_interval, max_push_sessions
mcast_driver	query_interval, max_push_sessions
mq_driver	query_interval, max_push_sessions, max_private_sessions
msg_broker	max_memory, max_concurrent_reqs
propagation_job	activation_state, remote_dn, propagation_timeout, propagation_username, propagation_password
prop_http	http_sll_level, wallet_location, wallet_password

Table 4–25 (Cont.) Allowed Update Operations for an Active Oracle Message Broker

Entry	Attributes that can be updated
queues	max_messages **Note**: With the Oracle Message Broker active, max_messages can only be updated for a queue using the Volatile Driver.
queue_trigger	concurrency, enabled, retries, threshold
remote_dir	all attributes
remote_http	all attributes
rv_driver	query_interval, max_push_sessions, max_private_sessions
topics	max_messages **Note**: With the Oracle Message Broker active, max_messages can only be updated for a topic using the Volatile Driver.
topic_trigger	enabled, retries, threshold
volatile_driver	query_interval, max_push_sessions

Delete Entry Restrictions

Table 4–26 contains the entries that cannot be deleted when the Oracle Message Broker is active. All other types of entries can be deleted when the Oracle Message Broker is active.

Table 4–26 Delete Entry Restrictions

Entry	Comment
aqlite_driver	
aqlite_server	
aq_driver	
aq_server	
durable_subscriber	Allowed only if the durable subscriber is not in use
mcast_driver	
mcast_server	
mq_driver	
mq_server	

Table 4–26 (Cont.) Delete Entry Restrictions

Entry	Comment
msg_broker	
prop_http	
queue	Allowed only if the queue is not in use
remote_dir	
remote_http	
rv_driver	
rv_server	
topic	Allowed only if the topic is not in use
volatile_driver	

Deleting a Propagation Job Entry

Deleting an activated propagation job is disallowed when the Oracle Message Broker is running (an activated propagation job is one with the `activation_state` attribute set to true).

To delete a propagation job entry when the Oracle Message Broker is running, first deactivate the propagation job. Refer to "Activating and Deactivating a Propagation Jobs" on page 8-26 for more information.

Command-line Administration Utility - AdminUtil

The `AdminUtil` command-line utility modifies entries in the LDAP Directory and allows the administrator to create, delete, and manage Oracle Message Broker configuration information.

The `AdminUtil` command operates in two modes: interactive and batch:

- In interactive mode an administrator enters commands at an input prompt. The interaction continues until the administrator enters a `quit` or `exit` command.
- In batch mode the utility processes files containing `AdminUtil` commands.

By default, `AdminUtil` starts in interactive mode and assumes that simple authentication is required. You are prompted for a user DN and a password if these are not specified on the command line (using the –noauth option you are not prompted for a user DN or a password).

Note: The LDAP Directory specified using the `OMB_IC` environment variable must be running before using `AdminUtil` (see "Required Environment Variables" on page 2-10 for details).

Enter a user DN and password and select the Continue button if your directory is set up for authentication. If the directory does not use authentication, or if you have set properties to indicate the user DN and password, leave these fields blank and select the Continue button. If you select the Exit button, `AdminUtil` exits (for more information on setting properties for security, refer to the section, "Enabling SSL and Authentication for the LDAP Directory" on page 12-20).

This section uses the following abbreviations:

cwo	the current working object
DN	an LDAP distinguished name
RDN	an LDAP relative distinguished name

To run `AdminUtil`, use the following syntax:

```
AdminUtil [options] [--] [arguments]
```

Table 4–27 shows `AdminUtil` options. The command line options are case sensitive.

The optional *arguments* can be specified on the command line. If optional arguments are present, named variables are set for each of the arguments. The variables have hardwired names of ARGV*n* where *n* is the argument number 0..*n*-1 (for example, ARGV0, ARGV1, ...). The variable can be used as any other named variables and managed using the `set` command.

The following examples illustrate the variables that would be set for the given command line.

```
AdminUtil myOmb myQueue
ARGV0:  myOmb
ARGV1:  myQueue

AdminUtil -f myFile myOmb myQueue
ARGV0:   myOmb
ARGV1:   myQueue
```

```
AdminUtil -f myFile --  -dashArg abc xyz
ARGV0:   -dashArg
ARGV1:   abc
ARGV2:   xyz
```

Note: `AdminUtil` does not remove JNDI related Java authentication properties. It does set JNDI properties based on command line options. You can modify the OMB_LP variable to set security properties, then run `AdminUtil` with the –noauth option to access a secured LDAP server.

Table 4–27 Command-line Options for AdminUtil

Option	Description
`--`	Marks the end of the command line options. Use `--` when the first optional argument starts with a dash.
–b *LDAP_basedn*	The *LDAP_basedn* supplies a base DN to use for the initial context. If you use the –b option, –h is required.
–D *auth_DN*	The *auth_DN* supplies the DN to use for user name authentication.
–echo	Echoes each input line. By default, the echo feature is disabled.
–errorlevel *level*	Set the error reporting level. The parameter *level* is set to an integer value in the range 1-4: 1 – print error message for the top exception 2 – print error messages for all linked exceptions 3 – print stack trace for the top exception 4 – print stack trace for all linked exceptions The default value for errorlevel is 2.
–f *inputFile*	Process a batch file. The file, *inputFile*, contains `AdminUtil` commands. Multiple –f options specify a sequence of files to be processed. Multiple files are processed in the order they appear on the command line.
–fullversion	Displays the full program version information.
–h *LDAP_host*	Specifies an LDAP server. The host, *LDAP_host*, must be a host available on the network.
–help	Displays the program usage.

Table 4–27 (Cont.) Command-line Options for AdminUtil

Option	Description
–ign	Causes the program to ignore errors when processing a batch file. If this option is not specified, processing ends if a batch command fails. This option is ignored for interactive mode.
–infolevel *level*	Set the information reporting level. The parameter *level* is set to an integer value in the range 1-3: 1 – print the minimum amount of informational messages 2 – print the standard amount of informational messages 3 – print the verbose informational messages. When a command fails, using infolevel 3 causes the command tokens to be shown after the error. The default value for infolevel is 2.
–int	Causes the program to continue in interactive mode after processing any batch files that may have been listed. Interactive mode is the default if the –f option is not specified.
–n	Causes most commands to operate in a "no execute" mode. When a command string is processed, a certain degree of validation may be performed but the command is not executed. AdminUtil may display a text message indicating command status. This option is useful for checking batch files for command syntax problems. Using `AdminUtil`, most commands honor the "no execute" flag while some general commands, including: echo, exit, help, run, set, and setopt behave normally; that is, they ignore the –n flag.
–noauth	Specifies that LDAP authentication is not required on the LDAP server.
–p *LDAP_port*	TCP port to use for the LDAP connection on the LDAP server. If you use the –p option, –h is required.
–P *wallet_password*	Specifies the wallet password. This is ignored if the value of –U is 0 or 1.
–U *value*	Specifies if SSL is used, and the authentication level. Valid *values* are: 0, 1, 2, and 3. `0` – no SSL. This is the default if –U is not specified. `1` – SSL with no authentication. `2` – SSL with server-side authentication. `3` – SSL with server-side and client-side authentication.
–version	Displays the program version number.
–w *auth_passwd*	Supplies a password, *auth_passwd*, for authentication on the LDAP server.
–W *wallet_path*	Specifies the path to an exported wallet file. This is ignored if the value of –U is 0 or 1.

Command Line Syntax

Each `AdminUtil` command is terminated with a semi-colon ';'. If a literal semi-colon is required in an input line, it must be escaped or quoted.

A single command can be split over multiple input lines. However, individual tokens must be fully specified on a given input line, including quoted strings. Quoted strings are not allowed to start on one input line and continue to the next. The beginning and ending quote characters must be on the same input line and an error will occur for unmatched quote characters.

`AdminUtil` processes commands until a `quit` or `exit` command is encountered, or the end of file is read in batch mode.

`AdminUtil` is not case sensitive for command names.

Quotation

`AdminUtil` provides three types of quotation:

- The escape character
- Single quotes
- Double quotes

A non-quoted backslash (\) is used as the escape character. It preserves the literal value of the following character.

Single quotes preserve the literal value of each character within the quotes. No escaping is allowed for single quotes so a single quote literal can not occur between single quotes.

Double quotes provide a weaker form of quotation than single quotes. All enclosed characters are preserved with the exception of $ and \. The $ character retains its special meaning within double quotes. The backslash (\) is used as an escape character only when followed by a $, \, or “ (dollar, backslash and double quote). For any other character, the \ is treated as a literal character.

A double quote can be part of a quoted string by escaping the double quote with a backslash (\”). For example,

```
"embedded\"quote"
"embedded\\backslash"
```

Comments and Echo Mode

An input line may contain comments. The '#' character marks the beginning of a comment. A comment starts when a word begins with '#' and extends to the end of the line. A '#' in the middle of a word or within a quoted string does not start a comment.

If a line begins with "#!" the remainder of the line is printed to the screen when echo mode is disabled. When echo mode is enabled, the entire line is echoed just like any other input line. This feature is useful for displaying user comments when processing a batch file.

LDAP Directory Interaction

Many `AdminUtil` commands result in an LDAP Directory operation. Other `AdminUtil` commands provide information for commands and set internal options to make the program easier to use. For example, `attrl` allows you to build a list of attributes for the `create` or `setattr` commands. Also, the `builddn` command allows you to construct a DN for an entry.

Object Binding

Commands are applied to a specific object reference or to the current bound object. The `cd` command sets the current bound object (or the –bind option of the `lookup` and `create` commands). See "Object References" on page 4-59, for more information on object references.

Named Variables

The named variable feature allows user-defined variable names to be associated with a string.

The `set` command associates a string with a named variable. The –set option also associates a string with a named variable for commands such as `cd`, `pwd`, `lookup`, `create` and `builddn`.

A named variable is a word consisting of only alphanumeric characters and the underscore, and must begin with an alphabetic character or underscore (`[a-zA-Z_][a-zA-Z0-9_]*`).

There are two forms used to reference a variable: $name and ${name}. Use the second form when the characters following the name are themselves valid identifier characters.

Variable expansion is done for any variable reference within a non-quoted string and double quotes. The $ is treated as a literal character if it occurs within single

quotes. A literal $ can be obtained by escaping it with a backslash (‘\$’). A $ followed by a whitespace is taken as literal whether or not it is escaped.

A undefined variable expands to the empty string.

Concatenation of Named Variables

A named variable, referenced as $name or ${name}, can occur at any position in a command string. The parameter is expanded if it occurs in a non-quoted token or is within double quotes.

Evaluation Rules

Input is evaluated, expanded and parsed into a command and list of arguments in a consistent manner. The following outlines the basic steps of the evaluation process:

1. Get an input line.
2. Remove comments.
3. Look for a command terminator (;) at the end of the line. If not found, repeat 1-3 until one is found.
4. Expand and parse the command string (see below).
5. Process the command.

The expansion and parsing rules for step four are as follows:

1. Split the initial command string on word boundaries. Steps 2-4 are applied to each token.
2. Expand variables. An empty string will be used for any undefined variables.
3. Split the token on word boundaries. This may produce additional arguments if a variable is not within a quoted string and its value has whitespace characters.
4. Remove quotation. This includes escaped characters in a non-quoted word or double quotes and the quote characters used to enclose a quoted string.

Be careful when including quotes with variables. Explicit null, empty string, arguments are retained as arguments, including those that surround undefined variables (explicit null arguments are sequences of “” or ‘’). However, implicit null arguments that result from the expansion of non-quoted parameters are removed.

For example, if you use variables as follows:

```
>set OPTION “-set name” ;
>cd $OPTION ~/path ;
```

This could create an invalid command, as in the following case when $OPTION is undefined:

```
>cd "$OPTION" ~/path;
```

The following example shows that when $OPTION is undefined, the command is valid when quotes do not enclose the undefined argument, since the implicit null arguments are removed:

```
>cd $OPTION ~/path;
```

Error Reporting

When `AdminUtil` processes a script file, and an error occurs, the last command string is printed after the error information. Printing this information can help identify the command that failed. The last command string is not printed when an error occurs in interactive mode.

If an error occurs when the infolevel is set to level 3 (verbose), the command token strings are printed after the error information. This information shows the actual command, and the command arguments after expansion is performed on a command string. The command tokens are printed for both interactive and file mode.

For example, the following `set` and `cd` command sequence causes an error because the value '-opt' is not a valid option for `cd`. The last command helps identify the command that failed. The command tokens show the command and arguments after the expansion. When the infolevel is set to 3, `AdminUtil` displays the following:

```
set A "-opt abc";
cd $A;

LAST COMMAND:
cd $A

COMMAND TOKENS:
[0]: cd
[1]: -opt
[2]: abc
```

Command List

Table 4–28 provides a description of the `AdminUtil` commands in alphabetical order.

Command Examples

Table 4–29 provides examples of the `AdminUtil` commands.

Table 4–28 AdminUtil Commands

Command	Description
activate [*entry*]	Reserved for future use.
attrl {–a | –r | –s} [*args*]	Manages an attribute list. Builds a set of attributes for commands such as `create` and `setattr`. There is one unnamed internal attribute list.
attrl –a *attrName* *attrValue* . . .	Adds one or more values for an attribute to the attribute list. If the list already contains an attribute with the name *attrName*, the values are added to the existing values. *attrName* specifies the attribute name. *attrValue* specifies the value(s) to be added for the attribute.
attrl –r [{* | *attrName*} [*attrValue*]]	Removes all attributes from the attribute list or values for a specific attribute. *attrName* specifies the name of the attribute for which values are to be removed. If this is not specified, or it is "*", all attributes are removed from the list. *attrValue* specifies the value(s) to be removed for the attribute.
attrl [–s {* | *attrName*}]	Shows the values for all attributes in the list or values for a specific attribute. *attrName* specifies the name of the attribute to show. If this is not specified, or it is "*", the command shows all attributes.
builddn [–set *var*] *entry*	Builds a distinguished name from an entry reference. This is similar to the `lookup` command except it does not check whether or not such an entry exists. This is useful for constructing a full DN and saving it to a named variable for later use as an attribute value. *entry* is a reference from which a DN is constructed –set *var* saves the built reference to the named variable *var*.
cd [–set *var*] [*entry*]	Binds the cwo to the specified entry. *entry* is a reference to the object to be bound as the cwo. If entry is not included, the cwo is bound to the initial context used to access the LDAP Directory. –set *var* saves an entry reference to the named variable *var*. See "Object References" on page 4-59, for more details on entry references.

Table 4–28 (Cont.) AdminUtil Commands

Command	Description
create [–bind]	Creates a new OMB entity.
[–set *var*]	The attribute values are a combination of those specified and the default values for all mandatory attributes not specified. Create fails if the object has mandatory attributes that are not supplied, that is, the user did not specify a value and the attribute is mandatory and does not have a default value.
[–al]	
[{–av "*name value*"}]	
[{–avpasswd *name*}]	*entryType* specifies the type of object to create.
entryType	*entryName* specifies the name of the object.
entryName	*parentEntry* specifies a reference to the parent of the object being created. If not specified, use the cwo as the parent.
[*parentEntry*]	
	–bind causes the cwo to be bound to the object that was created.
	–set *var* saves the object reference to the named variable *var*.
	–al causes attributes from an internal attribute list to be used for object creation. The `attrl` command is used to manage the attribute list.
	–av explicitly specifies an attribute value. Multiple –av options are allowed. The –av argument must be a quoted string with two or more words. The first word is the attribute name, and the subsequent word(s) specify the attribute value(s). Do not use the –av option when the name or the value contains whitespace or quote characters. If you need to include whitespace or quote characters, use the `attrl` command to set the value in the attribute list, and then use the `–al` option with the `create` command.
	–avpasswd displays a dialog that prompts the user for an attribute value. The text input field shows an echo character rather than the entered text. The dialog is displayed for both interactive and batch modes. A named variable can be used for the attribute name.
createombinstance	Constructs an OMB Instance container and other containers that are commonly found under the instance container. This command can also be used to repair an OMB Instance container by creating any entries that might be missing. This command is provided solely for ease of use.
[–bind]	
[–set *var*]	
[–repair]	*entryName* specifies the name of the omb_instance_container to create.
entryName	*parentEntry* specifies a reference to the parent entry. If not specified, use the cwo.
[*parentEntry*]	–bind causes the cwo to be bound to the object that was created.
	–set *var* saves the object reference to the named variable *var*.
	–repair allows an existing omb_instance_container to be repaired by only creating those entries that currently do not exist. If –repair is not specified, the command fails if the container already exists.
deactivate [*entry*]	Reserved for future use.

Table 4–28 (Cont.) AdminUtil Commands

Command	Description
delete [–subonly]	Deletes a specific entry, or deletes entries of a subtree.
[–ign]	Note: Use extreme caution when using this command. Depending on the options used, this command deletes a single object or deletes all entries in the subtree. An attempt to delete an object might fail even when that object has no children. This might occur if there is some dependency on that object by another object. The delete might fail with a notification error.
[–recurse]	
[*entry*]	
	entry specifies a reference for the target object or without an *entry*, the cwo.
	–ign option tells `delete` to silently ignore errors if the entry to be deleted does not exist.
	–subonly deletes only entries in the subtree of the target object. The target object is not deleted, even if –recurse is specified.
	–recurse deletes all objects in a subtree.
dir [*entry*]	Lists the contents (children) of an entry.
	entry is a reference to the object whose children are listed, or the cwo.
echo [*arguments*]	Writes the arguments to output. This is useful for diagnostic purposes and displaying variables. The arguments are separated by a blank and terminated with a newline. If no arguments are supplied, only a newline is output.
exit [all]	Exits the current command reader. This is the same as `quit`.
	`The` option, `all` for the `exit` command quits all command readers and terminates the program. If `all` is not specified, only the current command reader stops. If the reader was nested, started using the `run` command, processing continues using the previous reader.
help [*command*]	Displays usage information, or information for a specific command.
[–out *file*]	–out *file* sends the output to the specified file.
[*command*]	*command* displays help information for a specific command.
lookup [–bind]	Looks up a reference to retrieve such information as the entry name, entry type and distinguished name.
[–set *var*]	
entry	*entry* is a reference to the object that is the target of the lookup, or the cwo if not specified.
	–bind causes the cwo to be bound to that object.
	–set *var* saves a reference to the named variable *var*.
pwd [–set *var*]	Displays information for the current working object.
	–set *var* saves a reference for the current bound object to the named variable *var*.

Table 4–28 (Cont.) AdminUtil Commands

Command	Description
quit [all]	Exits the current command reader. This is the same as the `exit` command. The option, `all`, for the `quit` command quits all command readers and terminates the program. If `all` is not specified, only the current command reader terminates. If the reader was nested (started using the `run` command) processing continues using the previous reader.
`run` [*`filename`*]	Run suspends the current command reader, creates a new command reader and begins processing commands using the new reader. The current reader is resumed once the new reader has finished. An interactive reader is started if no filename is specified, otherwise a file reader is started. This command is useful for nesting batch files, running a batch file from interactive mode or switching to interactive mode from a batch file. When a series of readers have been started (that is, nested `run` commands) and an error occurs when processing a command from a file reader, that reader will be cancelled as well as all preceding file readers, up to the most recent interactive reader. If no interactive reader was started, the program terminates. The reader is not cancelled if the "ignore error" flag has been enabled and an error occurs. The "ignore error" can be enabled using the –ign command line option or the `setopt ignoreerror` command.
set [{–u | –s} [*args*]]	Manages named variables. There are two ways to assign a value to a variable. The set command associates a string to a named variable. The –set option for `cd`, `pwd`, `lookup`, `create` and `builddn` associates an entry DN to a named variable. A $varName or ${varName} syntax references the value associated with a named variable.
set *name value*	Saves an arbitrary string to a named variable. This variable can then be referenced by other commands (for example, `attrl`) as $name or ${name} when the value is needed. *name* specifies the variable name. *value* specifies the value.
set [–s {* | *name*}]	Shows the value of a specific variable or all variables. *name* specifies the named variable to show. If not specified, all named variables are shown.
set –u [{* | *name*}]	Removes (unsets) a specific variable or removes all variables. *name* specifies the named variable to clear. If not specified, all named variables are cleared.

Table 4–28 (Cont.) AdminUtil Commands

Command	Description
setattr {[–al] [{–av "*name value* "}] [{–avpasswd *name*}]} [*entry*]	Modifies attributes for an object. The attributes being changed are a combination of those from an internal attribute list (–al option) and / or those explicitly specified (–av and –avpasswd option). *entry* specifies a reference for the target object. If not specified, use the cwo. –al causes attributes from an internal attribute list to be used. The `attrl` command is used to manage the internal attribute list. –av allows the user to explicitly specify an attribute value. Multiple –av options are allowed. The –av argument must be a quoted string which consists of two or more words. The first word is the attribute name, and the subsequent word(s) specify the attribute value(s). Do not use the –av option when the name or the value contains whitespace or quote characters. If you need to include whitespace or quote characters, use the `attrl` command to set the value in the attribute list, and then use the –al option with `setattr`. –avpasswd displays a dialog that prompts the user for an attribute value. The text input field shows an echo character rather than the entered text. The dialog will be displayed for both interactive and batch modes. A named variable can be used for the attribute name.
setopt [*option* [*value*]]	Sets or displays program options. *option* specifies the option. *value* specifies the new value. If not specified, the current value of the option is shown.
setopt errorlevel [*level*]	Sets the error reporting level. The parameter *level* is set to an integer value in the range 1-4: 1 – print error message for the top exception 2 – print error messages for all linked exceptions 3 – print stack trace for the top exception 4 – print stack trace for all linked exceptions The –errorlevel command-line option can be used to set the errorlevel when `AdminUtil` is first started. The default value for errorlevel is 2.
setopt infolevel [*level*]	Displays or sets the information level. This gives the user a certain degree of control over how much information is output. Level should be in the range 0 to 3, from least to most verbose. The –infolevel command-line option can be used to set the infolevel when `AdminUtil` is first started.

Table 4–28 (Cont.) AdminUtil Commands

Command	Description
setopt ignoreerror [true|false]	**I**f enabled (true), ignore errors when processing commands from a batch file and continue processing the file. If disabled (false), cancel processing the commands if an error occurs. The –ign command-line option can be used to enable the ignore error mode when `AdminUtil` is first started.
setopt echo [{true | false}]	Sets the mode for echoing input lines. When echo mode is enabled, each input line is displayed. When echo is disabled, the input line is not echoed.
show ameta [–long | –short] [–type] [*entry* [*attrName*]]	Shows attribute metadata for an existing entry or an entry type. *entry* specifies a entry reference or entry type for which attribute metadata is displayed. This can be "." to reference the cwo. *attrName* specifies the name of the attribute. If not specified, metadata for all attributes is displayed. –long, provides detailed information (the default for a single attribute). –short, provides abbreviated information (the default for multiple attributes). –type, the entry argument specifies an entry type rather than an entry reference.
show attrs [*entry*]	Shows attribute values for an object reference. *entry* specifies a object reference whose attributes are displayed or without an *entry*, the attributes of the cwo.
show emeta [–long | –short] [–type] [*entry*]	Shows entry metadata for an existing entry, a specific type of entry or all entry types. *entry* specifies a entry reference or entry type for which entry metadata is displayed. This can be "." to reference the cwo. –long, provides detailed information (the default for a single entry). –short, provides abbreviated information (the default for multiple types). –type, the entry argument specifies an entry type rather than an entry reference. If –type is specified and entry is not specified, entry metadata for all entry types is displayed.

Table 4–28 (Cont.) AdminUtil Commands

Command	Description
show lasterror [*elevel*]	Shows information about the last error, at the specified error level. When using interactive mode, this command is useful for displaying information at a specific level for a specific command. See `setopt errorlevel` for information on setting the error level. *elevel* specifies the error level setting, and is set to an integer value in the range 1-4.
unsetattr { { –a { *name* | *}} ... } [*entry*]	Unsets the value for an optional attribute or the values for all optional attributes of an entry. Only optional attributes that are not read-only after creation can be unset. –a specifies the *name* of the attribute to unset. Multiple –a options are allowed. Using –a with * for the *name* unsets all optional attributes for the entry. *entry* specifies a reference for the target object or without an *entry*, the cwo.

Table 4–29 AdminUtil Examples

Example	Description
`activate;`	Sends an activate command to the object referenced by the currently bound object.
`attrl -a provider_queue_name "queue1";`	Adds the value queue1 to provider_queue_name.
`attrl -a server_dn “$MYBRK_DN”;`	Adds the value of MYBRK_DN to server_dn.
`attrl -a max_messages 250;`	Adds the value 250 to max_messages.
`attrl –r max_memory;`	Removes all values for an attribute.
`attrl –r *;`	Removes all attributes from list.
`attrl –s max_memory;`	Shows value for max_memory.
`attrl –s;`	Shows all attributes in list.
`builddn -set MYBASE1 "/c=us/o=C1/cn=OracleContext/cn=Products/cn=OMB";`	Builds a DN for an entry reference that is relative to the root and made up of atomic names. The DN is saved to the variable MYBASE1.
`builddn -set MYBASE2 "/$CO/$FIXEDDIT";`	Builds a DN for an entry reference that is relative to the root and made up of named variables. The DN is saved to MYBASE2.
`builddn -set MYOMB_DN "~/cn=myomb";`	Builds a DN for an entry reference that is relative to the initial context. The DN is saved to the variable MYOMB_DN.
`cd;`	Binds the cwo to the initial context.
`cd ~;`	Binds the cwo to the initial context.
`cd ..;`	Binds the cwo to the parent of the current cwo.
`cd $BRK_DN;`	Binds the cwo to the entry referenced by the named variable BRK_DN that is a child of the current cwo.

Table 4–29 (Cont.) AdminUtil Examples

Example	Description
`cd -set BRK_DN "~/$OMB_DN/msg_broker";`	Binds the cwo to an entry having the entry name msg_broker that is a child of an entry referenced by the named variable OMB_DN which is relative to the initial context. The reference is saved to BRK_DN.
`cd -set BRK_DN "./omb1/msg_broker";`	Binds the cwo to the entry referenced by an entry path made up of simple names. The entry reference is saved to BRK_DN.
`cd ~/cn=omb1/cn=Queues/queue1;`	Binds the cwo to an entry referenced by a path that is relative to the initial context and made up of simple and atomic name components.
`create msg_broker msg_broker "~/$OMB_ DN";`	Creates a new object under the entry referenced by the named variable OMB_DN. Use default values for the attributes.
`create -al topic topictest;`	Creates a new object under the cwo. The object is created using values from the internal attribute list and defaults for all others.
`create -bind -set MYTOPIC -al topic topicTest "~/$TOPICS";`	Creates a new object under the entry referenced by the named variable TOPICS. Uses attributes from the internal attribute list and default value for all other attributes. Binds the newly created object as the cwo and saves an object reference to the named variable MYTOPIC.
`create -set BRK_DN -al -av "max_memory 5" -av "max_concurrent_reqs $MAXREQ" msg_broker msg_broker "~/cn=myomb";`	Creates a new object under an entry named cn=myomb that is relative to the initial context. The creation attributes are a combination of those from the internal attribute list, those explicitly given via the –av options and defaults for all others. The value for the max_concurrent_ reqs attribute is the value of the named variable MAXREQ. The reference to the new entry is saved to BRK_DN.
`create -set AQSVR -av "aq_service_name $SVCNAME" -avpasswd aq_password aq_server aqsvTest ~/cn=myomb ;`	Creates a new object under an entry named cn=myomb that is relative to the initial context. The aq_service_name attribute is set to the value of the SVCNAME variable. A dialog prompts for the value of the aq_ password attribute. The reference to the new entry is saved to the named variable AQSVR.
`createombinstance -set OMB_DN ombtest ~;`	Creates a new OMB Instance container named ombtest and its associated sub–containers. It is created at the initial context and the reference saved to the named variable OMB_DN. This fails if ombtest already exists.
`createombinstance -set OMB_DN -repair ombtest "/$MYBASE";`	Repairs, or creates, an OMB Instance container whose parent is the entry referenced by the named variable MYBASE. This creates any entries that do not exist, including the OMB Instance container itself. The reference is saved to the named variable OMB_DN.
`deactivate "$BRK_DN";`	Sends a deactivate command to the object reference by the named variable BRK_DN.
`delete;`	Deletes the cwo. This will fail if any objects exist under the target object, cwo in this example.
`delete "~/$OMB/Queues/$QUEUE1";`	Deletes the referenced entry.
`delete -subonly;`	Deletes only the immediate children (one level) of the target object, the cwo. The target object is not deleted. Since the recurse option is not specified, any object which has children of its own is not deleted.

Table 4–29 (Cont.) AdminUtil Examples

Example	Description
`delete -subonly "~/$OMB_DN/Queues";`	Deletes the immediate children of the referenced entry.
`delete -recurse "~/$OMB_DN";`	Deletes all objects in the subtree of the object referenced by the named variable OMB_DN, including the target object.
`delete -subonly -recurse "~/$OMB_DN";`	Deletes all objects in the subtree of the object referenced by the named variable OMB_DN. The target object is not deleted.
`dir .;`	Lists the contents of the current bound object.
`dir "~/$OMB_DN";`	Lists the contents of the entry referenced by the named variable OMB_DN.
`echo A=$A ;`	Display the resulting string after the standard expansion and parsing has taken place.
`echo /$MYOMB_DN/cn=queues/cn=$MYQUEUE ;`	Display the resulting string after the standard expansion and parsing has taken place.
`help;`	Displays help for all commands.
`help set;`	Displays help for the set command.
`help –out cmd.hlp –all ;`	Writes help information for all commands to a file named cmd.hlp.
`lookup -bind -set OMB_DN ~/cn=ombtest;`	Looks up an atomic name that is relative to the initial context, binds the cwo to that object and saves an object reference to the named variable OMB_DN.
`lookup -var MYTOPIC ~/cn=topic1,cn=Topics,cn=ombtest;`	Looks up an object that is relative to the initial context and saves the entry DN to the named variable MYTOPIC.
`lookup msg_broker;`	Looks up an object which is a child of the cwo and has an entry name of msg_broker.
`lookup "~/$BRK_DN";`	Looks up the object that is relative to the initial context and referenced by the named variable BRK_DN.
`lookup -set BRK_DN ~/ombtest/msg_ broker;`	Looks up the object which from the initial context, there is an object that has an entry name of ombtest which has a child whose name is msg_broker. The entry DN is saved to BRK_DN.
`pwd;`	Displays information for the current bound object.
`pwd -set BRK_DN;`	Displays information for the current bound object and saves its reference to the named variable BRK_DN.
`run myfile ;`	Starts a new command reader for a file.
`run ;`	Starts a new interactive reader.
`set MYROOT "cn=OMB,cn=Products,cn=OracleContext,o=Comp,c=us";`	Sets a distinguished name value for the named variable MYROOT.
`set MAXMEM 5;`	Sets the value of MAXMEM to 5.
`set -s MYROOT;`	Shows value for MYROOT.

Table 4–29 (Cont.) AdminUtil Examples

Example	Description
`set -s *;`	Shows all named variables.
`set -u MYROOT;`	Clears variable MYROOT.
`set -u *;`	Clears all named variables.
`set FIXEDDIT cn=OMB,cn=Products,cn=OracleContext;`	Sets the value for the named variable FIXEDDIT.
`set COMP1 o=Comp1,c=us;`	Sets COMP1 to the specified value.
`setattr -al;`	Modifies attributes for the cwo using values from the internal attribute list.
`setattr -av "max_memory 6";`	Modifies a specific attribute for the cwo.
`setattr -al -av "max_memory 5" -av "max_concurrent_reqs $MAXREQ" ~/cn=myOmb ;`	Modifies attributes for an object having atomic name of cn=myOmb that is relative to the initial context. The attributes being set are a combination of those from the internal attribute list and those explicitly given via the –av options. The value for the max_ concurrent_reqs attribute comes from MAXREQ.
`setAttr -av "aq_service_name $SVCNAME" -avpasswd aq_password ~/cn=myOmb ;`	Modifies attributes for an object having atomic name of cn=myOmb that is relative to the initial context. The value for the aq_service_ name attribute comes from the named variable SVCNAME. A dialog prompts for the value of the aq_password attribute.
`setopt infolevel ;`	Show current information level.
`setopt infolevel 3 ;`	Sets a new information output level setopt.
`setopt ignoreerror ;`	Display the current ingore error setting.
`setopt ignoreerror true;`	Enable ignore error mode.
`setopt errorlevel 3;`	Set a new error information level.
`setopt errorlevel ;`	Display the current error information level.
`setopt echo;`	Shows the current echo mode.
`setopt echo true;`	Enables the input line echo mode.
`show lasterror 4;`	Display error information for the last error. A stack trace for all linked exceptions is displayed.
`show emeta . ;`	Shows entry metadata for the current cwo.
`show emeta ~/ombtest/msg_broker ;`	Shows entry metadata for a referenced entry.
`show emeta -type ;`	Shows entry metadata for all entry types.
`show emeta -type msg_broker;`	Shows entry metadata for a specific entry type.
`show ameta . ;`	Shows attribute metadata for the current cwo.
`show ameta . max_memory;`	Shows attribute metadata for a specific attribute of the cwo. By default, detailed information is displayed.

Table 4–29 (Cont.) AdminUtil Examples

Example	Description
`show ameta ~/ombtest/bkrtst max_memory;`	Shows attribute metadata for a specific attribute of a referenced entry.
`show ameta -type msg_broker;`	Shows attribute metadata for all attributes of a specific entry type.
`show ameta -type msg_broker max_memory;`	Show attribute metadata for a specific attribute of a specific entry type.
`show attrs . ;`	Shows attributes for the current cwo.
`show attrs ~/cn=ombtest/cn=msg_broker ;`	Shows attributes for the referenced entry.
`unsetattr -a max_memory ;`	Unsets the max_memory attribute for the cwo.
`unsetattr -a propagation_send_threads -a propagation_recv_threads;`	Unsets two attributes for the cwo.
`unsetattr -a * ~/cn=myOmb/cn=msg_broker;`	Unsets all optional attributes for a msg_broker entry.

Object References

Using `AdminUtil`, an object can be referenced using its LDAP distinguished name. This can be a full DN or an DN relative to an initial context. References use a most-significant to least-significant naming convention. For example, the following is a full DN for cn=myOmb:

```
cn=myOmb,cn=OMB,cn=Products,cn=OracleContext,ou=sales,o=oracle,c=us
```

If the initial context were set to the following: cn=Products,cn=OracleContext, ou=sales,o=oracle,c=us, the DN specified would be:

```
cn=myOmb,cn=OMB
```

`AdminUtil` supports a path syntax for entry references. The syntax uses a least-significant to most-significant naming convention (right to left). Each path component is separated by a '/' character. An absolute or relative path may be specified. A path reference that starts with "/" is interpreted as an absolute path. A path that starts with "~/" is interpreted as relative to the initial context. Anything else is interpreted as relative to the currently bound object.

For example, `AdminUtil` resolves the following reference to a full DN:

```
/c=us/o=oracle/ou=sales/cn=OracleContext/cn=Products/cn=OMB/cn=myOmb
```

For the initial context, cn=Products,cn=OracleContext,ou=sales,o=oracle,c=us, a path reference could be:

```
~/cn=OMB/cn=myOmb
```

Using `AdminUtil`, do not use any paths that place the current working object above the initial context set in `OMB_IC`. For example, if the initial context is set to the following:

```
ldap://system1/cn=inst1,cn=OMB,cn=Products,cn=OracleContext,ou=dept,o=comp,c=us
```

Then, if the current working object is `inst1`, you cannot use the command `cd..` to the change the current working object to `cn=OMB`. The upper limit in this example is:

```
cn=inst1
```

However, the `buildDn` command can use a path referring to an entry above the initial context. For example, if the current working object is `inst1`, then the command below sets the `BROKER` variable to the distinguished name of a broker under a different `omb_instance_container`:

```
buildDn -set BROKER ../cn=inst2/cn=abroker;
```

Valid path components include the following:

`~`	The initial context currently being used. This is similar to the Unix home reference.
`.`	The current working object (cwo).
`..`	The parent of the cwo.
`DN`	An LDAP distinguished name.
`RDN`	An LDAP relative distinguished name.
`name`	The simple name (for example, the Products part of cn=Products).
`atomic`	An LDAP atomic name (for example, o=oracle or cn=OracleContext).
`$`*`varName`*	A reference associated with the named variable varName.

Table 4–30 shows several sample object references.

Table 4–30 Sample Object References

Reference	Description
~	The initial context.
.	The cwo.
..	Parent of the cwo.
~/ombtest	An object named ombtest, a child of the initial context.
msg_broker	An object having an entry name of msg_broker that is a child of the cwo.
cn=msg_broker	An object using an atomic name that is a child of the cwo.
$BRK_DN	An object referenced by the named variable BRK_DN, a child of the cwo.
$OMB_DN/msg_broker	An object having the entry name msg_broker that is a child of the object referenced by the named variable OMB_DN which in turn is a child of the cwo.
../msg_broker	An object named msg_broker that is a child of the parent of the cwo. The msg_broker object is a sibling of the cwo (hierarchical-wise).
../msg_broker/aq_driver	An object named aq_driver which is a child of an object named msg_broker which in turn is a child of the parent of the cwo.
~/cn=ombtest/msg_broker	An object path relative to the initial context that has atomic and entry name components.
~/cn=ombtest/$QUEUES/qtest	An object path relative to the initial context that has atomic, named variable and entry name components.
/cn=OMB,cn=Products, cn=OracleContext,o=oracle, c=us/ombtest/msg_broker	An absolute object path which has RDN, atomic, and entry name components.
/c=us/o=oracle/ cn=OracleContext/cn=Products/ cn=OMB/ombtest/msg_broker	An absolute object path which has RDN, atomic, and entry name components.

Entry Attributes

Each entry type has attributes that define the type of data associated with the entry. Metadata exists for both the entry types and the attributes associated with the entries. Using `show emeta` displays the entry metadata for an existing entry or an entry type. Using `show ameta` displays attribute meta information for attributes of an existing entry or an entry type.

The attribute metadata details various information that might prove useful to the user. It indicates whether the attribute is single-valued or multi-valued. It indicates whether the attribute value can be set by the user when the entry is first created and if it can be modified once the entry has been created. It also indicates the syntax or format of the value.

The attribute syntax can be one of the following types.

Table 4–31 Entry Attribute Syntax

Value Type	Description
String	The value is a string.
Integer	The value is a string representing a numeric integer value.
Float	The value is a string representing a numeric float value.
Boolean	The value is a string that should be either "true" or "false".

Distinguished Name	The value is a string representing an LDAP distinguished name (DN). The DN string should be the full DN versus a DN relative to the initial context.
	For example, the full distinguished name of an entry that is relative the following initial context would be:
	Initial Context:
	`cn=OMB,cn=Products,cn=OracleContext,o=oracle,c=us`
	`Given DN:`
	`cn=msg_broker,cn=testOmb`
	`Resulting DN:`
	`cn=msg_broker,cn=testOmb,cn=OMB,cn=Products, cn=OracleContext,o=oracle,c=us`

Octet String	The value is an octet (byte) array. No value is displayed. This attribute type is used for binary data. The value stored is the UTF-8 encoding of the user-specified string.
Name Value Pair	The value is a string that represents a set of name value pairs. The value for such an attribute has a required syntax of *name=value*.

AdminUtil Limitations

1. The `set` command assigns an arbitrary string to a named variable. When used for a path component, a named variable will usually be the entry's simple name, atomic name or an RDN. If a simple name is specified but is not unique at the hierarchical level for which it is being used, it is arbitrary as to which object would actually be referenced. In that case, an atomic name must be used for that particular component to guarantee that the desired object is referenced.
2. It is recommended that double quotes be added to object references for a named variable. The double quotes are required if the variable value contains an embedded blank.
3. It is possible to delete the initial context from within an interactive session of `AdminUtil`. However, `AdminUtil` cannot perform any other actions if the initial context is deleted. To perform other actions, exit `AdminUtil` and set the initial context to an existing entry, and then restart `AdminUtil`.
4. Be careful when modifying the `server_dn` or `provider_queue_name` attributes for an existing Oracle AQ topic entry or Oracle AQ queue entry. Refer to the `create_provider_q`, `rm_provider_q`, and `provider_q_created` attribute information for queues and topics to determine the actions that are taken to create or remove a corresponding AQ queue when the Oracle Message Broker administrative tools modify queues or topics in the LDAP Directory.
5. The LDAP Directory server should not be shutdown when running AdminUtil.

Directory Utilities

This section covers the Oracle Message Broker utilities that allow you to check directory entries and migrate directory entries from previous Oracle Message Broker releases.

This section covers the following:

- Checking Directory Entries with AdminDirCheck
- Migrating Directory Entries Between Releases

Checking Directory Entries with AdminDirCheck

The `AdminDirCheck` command performs basic validation for Oracle Message Broker configuration data. It identifies problems with Oracle Message Broker entries in an LDAP Directory. The command uses Java Naming and Directory Interface (JNDI) to access the directory and then it checks either a single entry or all entries in a subtree. Starting at a given entry relative to the initial context, `AdminDirCheck` checks:

- Unknown entries and reports possible problems with the unknown entries.
- That required Oracle Message Broker DNs exist and are of the correct type.
- That entries are in the correct location in the Oracle Message Broker DIT hierarchy.
- That entries with attributes that are constrained to a set of predefined values, or a predefined range, contain valid values.

`AdminDirCheck` only accesses the LDAP Directory to check entries, it does not connect to an active Oracle Message Broker or validate any other Oracle Message Broker components.

You can use `AdminDirCheck` whenever the Oracle Message Broker is having unexpected problems or throwing unexpected exceptions. Such problems might range from problems a JMS client is having as it tries to use a queue, topic or connection factory obtained using JNDI to problems starting Oracle Message Broker.

If `AdminDirCheck` reports a problem, it does not attempt to fix the problem. You need to take the appropriate action based on the report to fix the problem. This may involve either modifying an entry or deleting and recreating an entry. An entry that contains a bad DN reference or a value out-of-range can be updated and the problem will be fixed by assigning a new, valid value.

`AdminDirCheck` classifies LDAP entries into three categories:

- An Oracle Message Broker entry
- An unknown entry
- A corrupt Oracle Message Broker entry. A corrupt Oracle Message Broker entry is an unknown entry that `AdminDirCheck` determines to be an Oracle Message Broker entry with invalid or missing data.

`AdminDirCheck` uses the OMB_IC environment variable to determine the LDAP server to connect to.

AdminDirCheck Options

`AdminDirCheck` uses the following syntax:

`AdminDirCheck [`*`initial_DN`*`] [`*`options`*`]`

By default, the program uses the following options: +all, –def, –unk. By default simple authentication is required. You are prompted for the user and password if these are not specified on the command line (using the –noauth option you are not prompted for a user and password).

Table 4–32 AdminDirCheck Options

Option	Description
initial_DN	Specifies the LDAP entry that is the starting point for the validation. This is the DN of the entry, relative to the initial context used to connect to the LDAP Directory. By default, the validation includes all entries in the entire subtree. Use –nosubtree to limit validation to a single entry.
{+|–}all	Enables or disables all validation options. This is generally used in combination with other options to enable or disable a set of options. For example, +all –dit enables all validation options except DIT checking; –all +ref enables only reference checking. +all enables all validation options –all disables all validation options
–b *LDAP_baseDN*	The *LDAP_baseDN* supplies a base DN to use for the initial context. If you use the –b option, –h is required.
–D *auth_DN*	The *auth_DN* supplies the DN to use for authentication.
{+|–}def	Enables or disables default value checking. Some attributes have a default value that is used if a value is not explicitly assigned for that attribute. +def searches for attributes that do not have an assigned value but have a default –def disables default value checking
{+|–}dit	Enables or disables DIT validation. Checks the hierarchy of the entries to insure the correct parent and child relationship exists. `AdminDirCheck` does not validate the topmost entry if the topmost entry has the same DN as the initial context. +dit enables DIT hierarchy checking –dit disables DIT hierarchy checking

Table 4–32 (Cont.) AdminDirCheck Options

Option	Description
–errorlevel *level*	Set the error reporting level. The parameter *level* is set to an integer value in the range 1-4: 1 – print error message for the top exception 2 – print error messages for all linked exceptions 3 – print stack trace for the top exception 4 – print stack trace for all linked exceptions The default value for errorlevel is 2.
–help	Displays the program usage
–h *LDAP_host*	Specifies an LDAP server. The host, *LDAP_host* must be a host available on the network.
–fullversion	Displays the full program version information.
–noauth	Specifies that LDAP authentication is not required on the LDAP server
–nosubtree	Limits the validation to a single entry specified by the *initial_DN* parameter. If this option is not specified, all entries in the subtree are checked.
{+|–}omb	Enables or disables corrupt Oracle Message Broker entry checking. This option attempts to distinguish corrupt Oracle Message Broker entries from non-Oracle Message Broker entries. +omb searches for unknown entries that are likely to be corrupt Oracle Message Broker entries –omb disables corrupt Oracle Message Broker entry checking
–p *LDAP_port*	TCP port to use for the LDAP connection on the LDAP server. If you use the –p option, –h is required.
–P *wallet_password*	Specifies the wallet password. This is ignored if the value of –U is 0 or 1.

Table 4–32 (Cont.) AdminDirCheck Options

Option	Description
{+|–}ref	Enables or disables reference validation. Checks certain types of attributes that reference other entries. For example, a DN attribute is checked to see whether the referenced entry exists, and if possible, that the referenced entry is of the correct type. +ref enables reference checking –ref disables reference checking For reference validation, it is possible for an entry referenced by a DN attribute to be reported as not existing, when, in fact, it does exist. This might occur for a variety of reasons. For example in the following cases: ■ If the referenced entry is not accessible by the initial context ■ If the user does not have access rights to the referenced entry ■ If the reference is to an entry on another LDAP server For an entry that contains an attribute that is a DN, using the +ref option `AdminDirCheck` does not check whether the DN is within the same OMB Instance. It only tries to verify that the entry exists and that the referenced entry is the correct Oracle Message Broker entry type (if applicable).
{+|–}rng	Enables or disables range validation. Checks the attributes of an Oracle Message Broker entry for range violations such as being outside the lower or upper limits or a value not from a set of valid values. Many attributes have no value limitations. +rng enables range checking –rng disables range checking
–U *value*	Specifies if SSL is used, and the authentication level. Valid *values* are: 0, 1, 2, and 3. `0` – no SSL. This is the default if –U is not specified. `1` – SSL with no authentication. `2` – SSL with server-side authentication. `3` – SSL with server-side and client-side authentication.
{+|–}unk	Enables or disables unknown entry checking. The subtree being scanned may have both Oracle Message Broker (known) and non-Oracle Message Broker (unknown) entries. This option can be used to search for non-Oracle Message Broker entries. An attempt is made to distinguish between a corrupt Oracle Message Broker entry and a non-Oracle Message Broker entry and report only a non-Oracle Message Broker entry. +unk reports any unknown (non-Oracle Message Broker) entries –unk disables unknown entry reporting

Table 4–32 (Cont.) AdminDirCheck Options

Option	Description
–version	Displays the program version number.
–w *auth_password*	Supplies a password, *auth_password*, for authentication on the LDAP server.
–W *wallet_path*	Specifies the path to an exported wallet file. This is ignored if the value of –U is 0 or 1.

Note: `AdminDirCheck` does not remove JNDI related Java authentication properties. It does set certain JNDI properties based on command line options. Thus, you could modify the OMB_LP environment variable to set the JNDI security properties and run `AdminDirCheck` with the –noauth option, and still access a secured LDAP server based on the properties defined in OMB_LP.

Migrating Directory Entries Between Releases

The Oracle Message Broker provides a migration utility for migrating directory entries from release 1.0 to release 2.0. The migration utility performs the following actions:

- Changes the container name cn=oraclesoftware to cn=oraclecontext. This is one requirement to migrate from release 1.0 to release 2.0.
- Changes all attributes that have DNs using the oraclesoftware component to oraclecontext.
- Creates the new containers required for release 2.0.

Note: For a complete migration, the Oracle Message Broker 2.0 version of the command `LDAPSchema` must be run on the LDAP Directory. It is not sufficient to only run `Migrate10To20`. See the *Oracle Message Broker Installation Guide* for your platform for information on running `LDAPSchema`.

Use this utility with caution. Some LDAP Directories, for example the Netscape Directory, do not allow renaming of an entry which is not a leaf entry. The change from cn=oraclesoftware to cn=oraclecontext requires to migration utility to delete all of the entries underneath cn=oraclesoftware and then recreate them.

Partial failures or cancelling the command before it completes can lead to problems. Before deleting any entries, the utility saves the Oracle Message Broker directory data in a file named OMB10DATA.ldif, LDIF format, in the current directory.

Take the following precautions while running the `Migrate10To20`:

1. While `Migrate10To20` is in use, there should be no other directory operations in progress.
2. The base DN (–b option) should be carefully selected. Make the base DN match the initial context set in the environment variable OMB_IC (and remove the leading components, cn=Products,cn=OracleSoftware,).
3. The `Migrate10To20` does not work correctly if an entry, cn=OracleSoftware, which is of type container exists underneath an entry, cn=OracleSoftware, which is also of type container.

   That is, if both of the following entries exist in the same LDAP Directory, and both entries are of type container:

   ```
   cn=OracleSoftware,ou=bizzare,o=acme,c=us,
   cn=OracleSoftware,cn=foo,cn=foo,cn=OMB,cn=Products,cn=OracleSoftware,ou=bizz
   are,o=acme,c=us
   ```

Migrate10To20 Options

`Migrate10To20` uses the following syntax:

```
Migrate10To20 [options]
```

Table 4–33 shows the available options. By default, the program uses no options, and assumes that simple authentication is required. You are prompted for the user and password if these are not specified on the command line (using the –noauth option you are not prompted for a user and password).

For example:

```
% Migrate10To20 -h hal -p 389 -b ou=bizarre,o=acme,c=us
```

Table 4–33 Migrate10To20 Options

Option	Description
–b *LDAP_baseDN*	The *LDAP_baseDN* , or the initial context supplies a base DN to use for the initial context.
–D *auth_DN*	The *auth_DN* supplies the DN to use for authentication.
–h *LDAP_host*	Specifies an LDAP server. The host, *LDAP_host* must be a host available on the network.
–fullversion	Displays the full program version information.
–ldapv2	Use this option when the LDAP Directory only supports LDAP version 2. The default, without this option is support for LDAP version 3.
–noauth	Specifies that LDAP authentication is not required on the LDAP server
–p *LDAP_port*	TCP port to use for the LDAP connection on the LDAP server.
–P *wallet_password*	Specifies the wallet password. This is ignored if the value of –U is 0 or 1.
–U *value*	Specifies if SSL is used, and the authentication level. Valid *values* are: 0, 1, 2, and 3. `0` – no SSL. This is the default if –U is not specified. `1` – SSL with no authentication. `2` – SSL with server-side authentication. `3` – SSL with server-side and client-side authentication.
–version	Displays the program version number.
–w *auth_password*	Supplies a password, *auth_password*, for authentication on the LDAP server.
–W *wallet_path*	Specifies the path to an exported wallet file. This is ignored if the value of –U is 0 or 1.

Authorization for Running Migrate10To20

The LDAP Directory username and password associated with the `Migrate10To20` are supplied on the command-line using the –D and –w options (or provided from the values supplied in the pop-up window if command-line options are not supplied). `Migrate10To20` uses the supplied username and password when it accesses the LDAP Directory.

The username provided needs to have create and delete permissions in the subtree rooted at the supplied base DN or initial context (supplied with the –b option). It is recommended that the username should be the superuser (or equivalent, such as administrator).

If there are user entries underneath cn=oraclesoftware, they are migrated and any password associated with the entry is not changed. However, the username, which

is the DN of the entry, does change due to replacing cn=oraclesoftware with cn=oraclecontext.

For example, if the following two entries show the old, and the new username, before, and after running `Migrate10To20`:

```
cn=akarmark,cn=users,cn=omb,cn=products,cn=oraclesoftware,<initial-context>
cn=akarmark,cn=users,cn=omb,cn=products,cn=oraclecontext,<initial-context>
```

All encrypted attributes, such as the AQ password stored in the `aq_password` are migrated without changes.

ACIs that are set on an LDAP Directory entry are migrated without changes. However, if the ACIs grant / refuse permission to a user / group which is directly underneath the container, cn=oraclesoftware, that was migrated, then the ACIs are invalid, since the DN is not longer valid.

5

Oracle Message Broker Features

This chapter covers JMS programming features and provides information on programming using the Oracle Message Broker's implementation of JMS. Chapter 6, "Oracle Message Broker Extensions" covers additional Oracle specific features of the Oracle Message Broker.

This chapter covers the following:

- Working With JMS Messages
- Using a QueueBrowser
- Using Durable Subscribers
- Using the PL/SQL Operational Interface
- Running in Local Mode
- Running in Remote Mode
- Oracle Message Broker Version Checking

Working With JMS Messages

This sections covers information on JMS messages. The Oracle Message Broker supports all JMS message types. Each JMS message consists of the following parts:

- Header – header fields contain values used by clients and by the Oracle Message Broker to identify and route messages.
- Properties – properties fields add a built-in facility for adding optional header information. Properties are a variable-length list of name / value pairs.
- Body – JMS defines several types of message body. The Oracle Message Broker supports all JMS body types. The body contains the actual message data, the format of which depends on the type of the JMS message: TextMessage, BytesMessage, MapMessage, ObjectMessage, or StreamMessage.

Message Properties

JMS supports message *properties* that provide a built-in facility for adding optional header fields to a message. Properties allow Oracle Message Broker to support message selectors that select specific messages to distribute to a destination (see "Using Message Selectors" on page 5-3, for more information). The JMS specification includes three types of properties:

- Application-specific properties
- JMS standard properties
- Provider specific properties

Using the JMS_Oracle_Delay Message Property

The Oracle Message Broker defines the JMS_Oracle_Delay provider specific property. The JMS_Oracle_Delay is an integer whose value represents the number of milliseconds to delay before the message is available for delivery. Using Remote Mode, the JMS_Oracle_Delay property is an offset from the current time on the system on which the remote mode Oracle Message Broker runs. Client programs can set this property based on the current time to assure that messages are only sent during specific periods, for example, at night.

JMS_Oracle_Delay is only supported for the Volatile Driver and the AQ Driver.

For example to set the delivery delay property to 60 seconds:

```
// Delay delivery for 60 seconds
javax.jms.Message msg = ...;
msg.setIntProperty("JMS_Oracle_Delay", 60000);
```

And the following code sets the delivery delay to 10 seconds:

```
// Delay delivery for 10 seconds
javax.jms.Message msg = ...;
msg.setObjectProperty("JMS_Oracle_Delay", new Integer(10000));
```

Using Message Selectors

Message selectors allow the Oracle Message Broker to filter the messages that are sent to consumers based on a message selection criteria. This allows the Oracle Message Broker to handle filtering so that the client application does not need to receive messages that it is not interested in. Message selectors simplify a JMS client application and eliminate the overhead associated with sending messages to a client that it does not need.

A JMS client specifies message selection criteria when it creates a message consumer (either a QueueReceiver, QueueBrowser, or TopicSubscriber). JMS message selectors always reference either a header field or a message property. JMS message selectors do not apply to the message body. Table 5–1 lists the parts of the message header that can be specified in a message selector.

Note: The AQ Driver supports JMS message selectors and AQ Rules based selectors. Refer to "Using AQ Rules for Message Selection" on page 6-18 for information on AQ Rules.

Table 5–1 Message Selector Identifier References

Reference	Description
JMSPriority	The corresponding message header field
JMSDeliveryMode	The corresponding message header field
JMSMessageID	The corresponding message header field
JMS_Oracle_Delay	See "Using the JMS_Oracle_Delay Message Property" on page 5-2 for details
JMSTimeStamp	The corresponding message header field
JMSCorrelationID	The corresponding message header field
A name that does not begin with JMS	An application specific property name

Message Selector Format

A message selector is a Java String whose syntax is based on the SQL92 conditional expression syntax. Section 3.8.1.1 of the JMS Specification defines the message selector syntax. In addition, Section 3.10 of the Java Language Specification defines the syntax for floating-point and integer literals that are valid in message selectors.

Keep the following points in mind when creating a message selector:

1. Integer literals of type long must have a suffix of 'l' or 'L'.
2. Floating point literals of type float must have a suffix of 'f' or 'F'.
3. Floating point literals of type double can have a suffix of 'd' or 'D'.
4. The exponent indicator can be 'e' or 'E'.

Example 5–1 shows code for a sample message selector for both point-to-point and publish/subscribe messaging.

Example 5–1 Sample Message Selector

```
String selector;
QueueReceiver receiver;
TopicSubscriber subscriber;

selector = new String("JMSType = 'car' ");
receiver = session.createReceiver(queue, selector);
subscriber = session.createSubscriber(topic, selector);
```

QueueReceivers and Message Selectors (Limitation)

Queue receivers that use message selectors can be inefficient. When a queue receiver is created with a message selector, any message that does not satisfy the selector cannot be delivered to that receiver and the message remains on the queue. This implies that the Oracle Message Broker performs a non-destructive read to examine the messages and then uses the message ID to dequeue messages when a message is found that satisfies the message selector.

To satisfy these requirements for a QueueReceiver, the Oracle Message Broker reads messages twice for QueueReceivers using message selectors. The first read is a non-destructive read and the second read is the dequeue by message ID.

Another inefficiency for QueueReceivers using message selectors is due to the separation between the Oracle Message Broker and the persistent message store.

There is not an efficient technique to determine when messages have been added to a queue.

Because the Oracle Message Broker cannot always detect state changes in the message store, it must execute non-blocking reads when it attempts to retrieve new messages. Therefore, if there are 10 messages on a queue, and there is an outstanding request for a message by a receiver with a message selector, The Oracle Message Broker must poll the queue.

Therefore, when there is a QueueReceiver and a selector, polling is required.

Using a QueueBrowser

The Oracle Message Broker supports JMS QueueBrowsers. A QueueBrowser enables a client to look at messages on a queue without removing the messages. How the QueueBrowser functions depends upon the driver:

Volatile Driver	Messages that may be fetched are those ready for delivery when the browser is created and any messages sent and committed after the browser is created.
AQ Driver (OCI Mode)	Messages that may be fetched are messages ready for delivery when the browser is created. Messages that are ready for delivery or those in state 0 (see the Oracle 8*i* Database Server documentation for more information).
AQ Driver (JDBC Mode)	Messages may be fetched are those available using browsing as implemented by Oracle AQ.
AQ Lite Driver	Messages that may be fetched are messages ready for delivery when the browser is created. Messages that are ready for delivery when the browser is created.
MQ Driver	Uses native MQ browse functions.

Using Durable Subscribers

JMS publish/subscribe messaging defines methods for both non-durable and durable subscriptions. Non-durable subscriptions only last while the subscriber is active, and the client using the subscriber only sees the messages that are published while the subscriber is active. The Oracle Message Broker supports durable subscribers that retain the messages that are published to a topic until the durable subscriber, identified by a unique ID, receives the messages or until the messages expire (refer to Chapter 7, "Message Servers and Drivers" and the *Oracle Message Broker Release Notes* for limitations and for information on driver specific features related to durable subscribers).

The Oracle Message Broker manages durable subscribers by adding entries to the LDAP Directory for each durable subscriber. As JMS clients create durable subscribers, durable subscriber directory entries are added, and as clients delete durable subscribers directory entries are removed. There are several cases where the Oracle Message Broker handles special conditions for durable subscribers. The following is a list of some of these conditions:

1. If a JMS client creates a durable subscriber and the following are both true then the Oracle Message Broker throws an exception:
   - The durable subscriber exists
   - The durable subscriber is in-use. A durable subscriber is in-use when there is an active subscriber
2. If a JMS client creates a durable subscriber and the following are all true, then the current subscription is deleted and a new subscription is created. Any messages that were in the topic for the durable subscriber are lost when the subscription is deleted.
   - The durable subscriber exists
   - The durable subscriber is not in use
   - A different filter has been specified for the durable subscriber
3. If a durable subscriber is created and the Oracle Message Broker cannot modify the LDAP Directory with the durable subscriber information, the Oracle Message Broker throws an exception.
4. All the directory attributes for a durable subscriber entry are read-only-after-create. This means that you can only view the durable subscriber attributes. If you need to modify a durable subscriber entry, delete the entry and then recreate it.

Using the PL/SQL Operational Interface

The PL/SQL package, `ombaqpublic`, implements the Oracle Message Broker's publicly supported PL/SQL operational interface to AQ queues and topics (multi-consumer AQ queues). This package allows PL/SQL applications running inside the Database Server to enqueue and dequeue messages directly from AQ queues and topics. The AQ Queues must have been created using the Oracle Message Broker administrative utilities, and they must use one of the following supported types:

- `OMBAQ_TEXT_MSG`
- `OMBAQ_BYTES_MSG`
- `MESSAGE_T`

Oracle Message Broker can dequeue messages enqueued using the `ombaqpublic` PL/SQL interface, and convert them into a JMSTextMessage or a JMSBytesMessage format. The messages can then pass to JMS clients or be propagated between Oracle Message Brokers without any loss of information. In addition PL/SQL applications are able to browse or dequeue messages enqueued by the Oracle Message Broker into AQ queues using the `ombaqpublic` package (as long as the AQ Queues are in one of the supported types shown above).

The Oracle Message Broker PL/SQL package, `ombaqpublic`, is modeled after the AQ PL/SQL operational interface. The package allows setting of AQ message properties, including priority, delay, and expiration. The package also supports Oracle AQ enqueue options including: visibility and relative messageid. And the package supports the Oracle AQ dequeue options including dequeue mode and navigation using the types and constants defined by the Oracle AQ PL/SQL interfaces. Oracle Message Broker PL/SQL package, `ombaqpublic`, allows for JMS specific message properties to be set, for which there are no corresponding analogues in AQ. These properties include the SQL equivalents of JMSProperties, JMSType and JMSReplyTo.

The package provides four enqueue and four dequeue subroutines. Table 5–2 shows the Oracle Message Broker PL/SQL package subroutines.

Exceptions, are thrown when the package is improperly used and should be handled by the PL/SQL application developer.

The source for the PL/SQL routines is available in the following files:

```
$OMB_HOME/admin/plsql/ombaqpublic.sql
$OMB_HOME/admin/plsql/ombaqpublicb.sql
```

or, on Windows NT systems:

```
%OMB_HOME%\admin\plsql\ombaqpublic.sql
%OMB_HOME%\admin\plsql\ombaqpublicb.sql
```

Table 5–2 PL/SQL Client Interface Subroutines

PL/SQL Subroutine	Description
`dequeue_blob()`	Is used to dequeue messages from AQ queues in the form of BLOB messages.
`dequeue_clob()`	Is used to dequeue `CLOB` messages from AQ queues of type `OMBAQ_TEXT_MSG` and `MESSAGE_T`.
`dequeue_raw()`	Is used to dequeue messages from AQ queues in the form of `RAW` messages from queues of type `OMBAQ_BYTES_MSG`.
`dequeue_varchar()`	Is used to dequeue VARCHAR messages from AQ queues of type `OMBAQ_TEXT_MSG` and `MESSAGE_T`.
`enqueue_blob()`	Is used to enqueue `BLOB` messages into AQ queues of type `OMBAQ_BYTES_MSG`.
`enqueue_clob()`	Is used to enqueue `CLOB` messages into AQ queues of type `OMBAQ_TEXT_MSG` and `MESSAGE_T`.
`enqueue_raw()`	Is used to enqueue `RAW` messages into AQ queues of type `OMBAQ_BYTES_MSG`.
`enqueue_varchar()`	Is used to enqueue `VARCHAR` messages into AQ queues of type `OMBAQ_TEXT_MSG` and `MESSAGE_T`.

Running in Local Mode

The Oracle Message Broker provides two modes of operation, Remote Mode and Local Mode (Remote Mode is also called Non-Local Mode). When configured in Local Mode, one active Oracle Message Broker is created in each client JVM (that is, in each program that uses the JMS API). Local Mode provides a decentralized architecture, supporting a more efficient and more robust system with no single point of failure.

Local Mode Oracle Message Brokers are especially useful for applications using the Oracle Multicast Driver or the TIBCO Driver with multicast communication. In a typical Local Mode configuration, multiple Oracle Message Brokers, that is multiple clients, use a single directory instance. Running in Local Mode allows the Oracle Message Broker to operate very efficiently, without utilizing significant system resources.

In Local Mode, as in Remote Mode, the Oracle Message Broker provides two choices storing administrative objects:

- Using an LDAP Directory (see Chapter 4 for information on LDAP Directory configuration)
- Using Lightweight Configuration (see Chapter 13 for information on lightweight configuration)

This chapter describes how to use Oracle Message Broker Local Mode with an LDAP Directory, and how to use Oracle Message Broker Local Mode without an LDAP Directory.

Using Local Mode with an LDAP Directory

Operation in Local Mode is the only situation where multiple, active Oracle Message Brokers can share an Oracle Message Broker Instance in the LDAP Directory. Since Oracle Message Brokers working together using the Oracle Multicast Driver or the TIBCO Driver must define the same topics, Local Mode allows LDAP Directory administration to be significantly simplified. Using the LDAP Directory, only one Oracle Message Broker Instance needs to be maintained, since the multiple active Oracle Message Brokers share a single Oracle Message Broker Instance. Alternatively, using lightweight configuration, only one configuration file needs to be maintained.

A Local Mode Oracle Message Broker process is created when the `local` attribute in the Oracle Message Broker Instance's `msg_broker` entry is true, and an Oracle Message Broker client creates a connection (you do not use the `MsgBroker` command for Local Mode).

Note: A remote client cannot use a Local Mode Oracle Message Broker.

Using Local Mode, the client uses the same JVM as the Oracle Message Broker instance. That is, the client and the broker are not only on the same system, but are also in the same process. Using Local Mode, the Oracle Message Broker instance is limited to local operation for the local client.

When a client looks up a connection factory associated with a local Oracle Message Broker, it creates an active OMB Instance in the client's JVM. Because a connection factory establishes a connection with an active Oracle Message Broker, a client program may use Local Mode and Remote Mode Oracle Message Brokers at the

same time by creating multiple connection factories. Note that only one local OMB Instance is allowed per JVM process.

When starting a client that uses a local Oracle Message Broker, start the Java interpreter with the following flags:

`-mx`*`mem`*`m` (for Java 1.1.x)	Java JVM maximum heap size.
`-Xmx`*`mem`*`m` (for Java 1.2)	
`-ms`*`mem`*`m` (for Java 1.1.x)	Java JVM startup heap size.
`-Xms`*`mem`*`m` (for Java 1.2)	
`-Doracle.oas.mercury.maxheap=`*`mem`*	(Optional) – this is used to inform the Oracle Message Broker of the maximum JVM heap size.

Where *mem* is the memory, in megabytes, that you want to allocate to the Oracle Message Broker process. The *mem* value should be at least 8, specifying 8 megabytes per instance. If `max_memory` is set in the `msg_broker` entry, the *mem* value, as shown above, must be greater than or equal to the value of `max_memory`.

Table 2–3 shows the environment variables that Oracle Message Broker Local Mode programs need to start. The environment variables shown are defined in the Oracle Message Broker startup files. See "Working with the Administration Utilities" on page 2-2 for more information on the startup environment.

Authentication in Local Mode

The authentication information and mode used by a JMS client to look up the connection factory will be the same information that the local OMB Instance uses when it is started. Thus, the authentication, user and password, are the same for the client and the Oracle Message Broker when running in Local Mode.

Stopping Oracle Message Broker in Local Mode (using an LDAP Directory)

A client program running the Oracle Message Broker should stop the Oracle Message Broker when it is finished. For information on stopping the Oracle Message Broker, see "Shutting Down" on page 3-12.

Using Local Mode with Lightweight Configuration

Using Local Mode, the client uses the same JVM as the Oracle Message Broker instance. That is, the client and the broker are not only on the same system, but are also in the same process.

Local Mode Limitations

In Local Mode, the following limitations apply:

1. Propagation is disabled – see Chapter 8, "Oracle Message Broker Propagation" for details on using propagation in Remote Mode.
2. DMS is disabled – see "Collecting Runtime Metrics" on page 6-3 for information on displaying runtime metrics in Remote Mode.
3. The `MsgBroker` command is not used.
4. The messages in the Volatile Driver can only be accessed from a single Oracle Message Broker Instance. If the Oracle Message Broker Instance is running in local mode, then all clients must access it from within the same process. Due to this restriction, Oracle Message Broker Remote Mode is recommended when using destinations with the Volatile Driver.

Sample Local and Remote Mode Client Programs

To illustrate a client program that uses a local Oracle Message Broker, we present a sample stock quote system. This system is composed of a stock quote database, one or several publishers, and any number of subscribers that represent the clients requesting quotes.

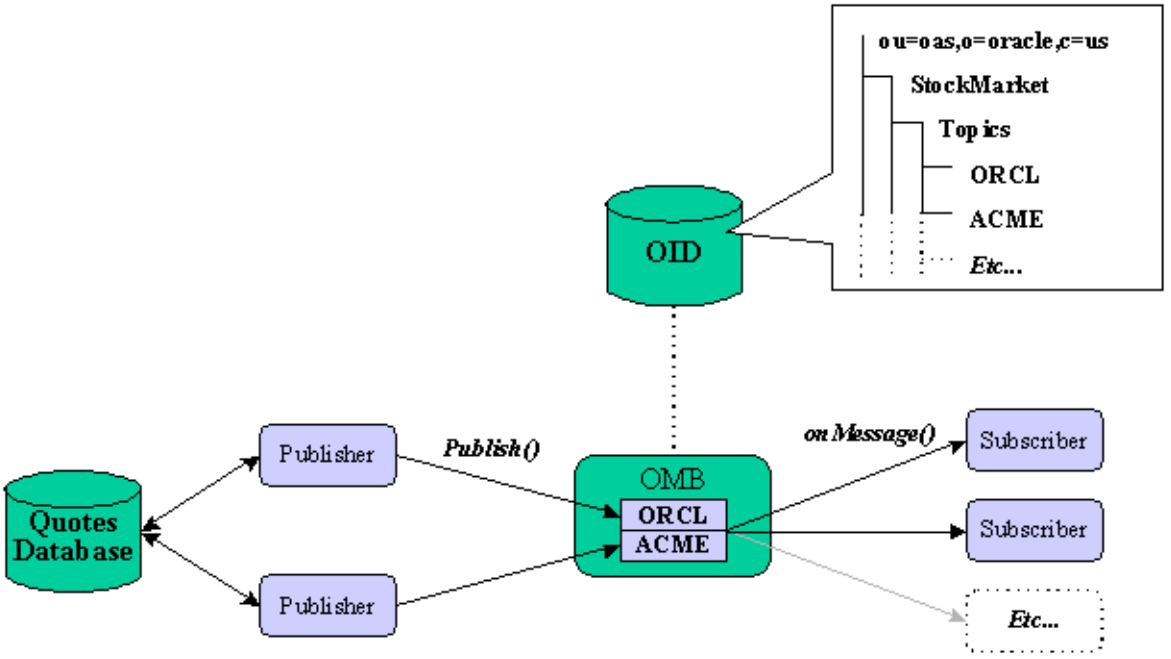
In this application, the publishers retrieve quotes from the databases and publish them to topics. The application supports any number of subscribers. Each subscriber subscribes to topics according to its interest, and receives quotes asynchronously. Topics can be mapped to individual stock quotes (for example, ORCL, for Oracle Corp. quotes), or to sets of similar stock quotes (for example, the high-tech market).

Stock Quotes Using a Remote Oracle Message Broker

When using a Remote Mode Oracle Message Broker to implement the stock quote application, all publishers and subscribers use a centralized Oracle Message Broker to forward stock quotes from the publishers to all of the subscribers (see Figure 5–1). Although this architecture is straightforward to deploy and maintain, it suffers from some limitations when applied to the stock quote application: it does

not scale well to hundreds of subscribers since quotes are typically sent to all subscribers using a sequence of point-to-point messages, and a failure of the centralized Oracle Message Broker completely stops the service for all subscribers.

Figure 5–1 Remote Mode Oracle Message Broker Quote System

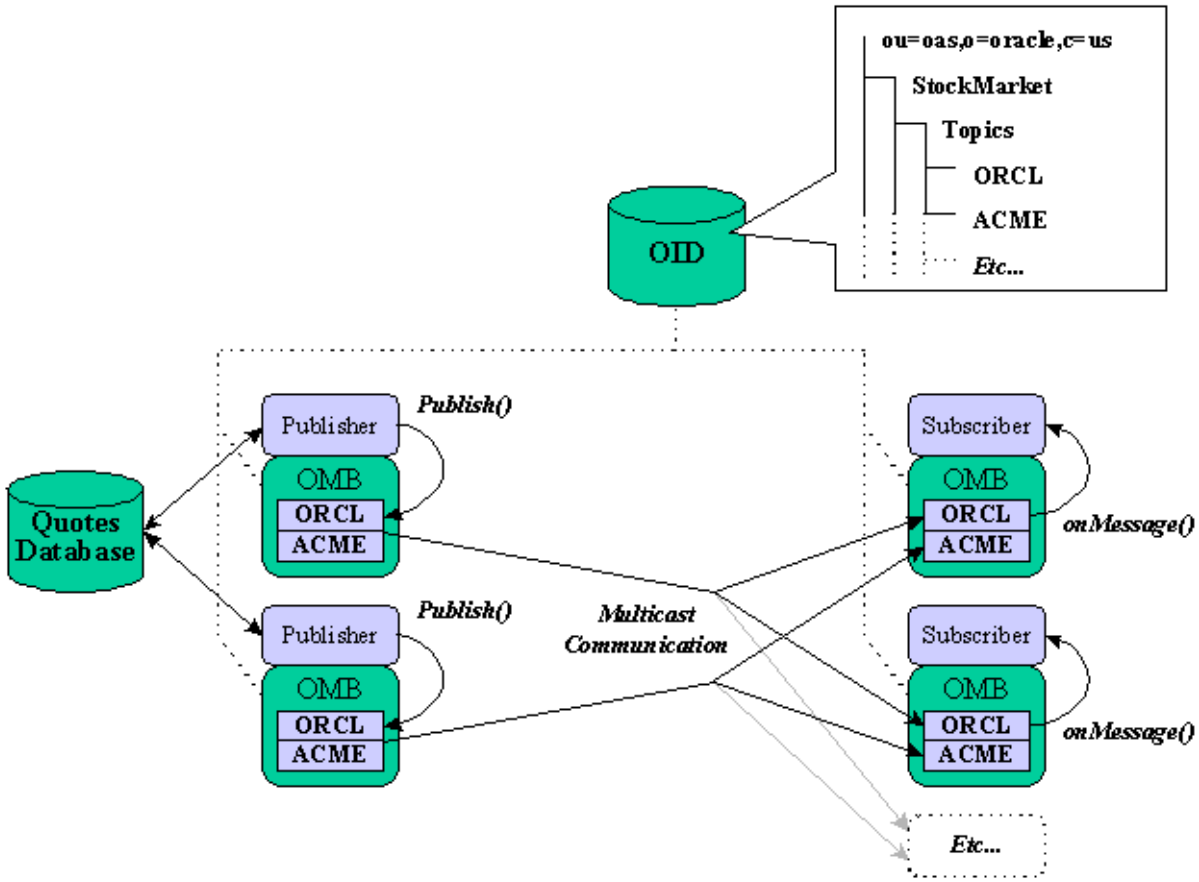


Stock Quotes Using Local Mode Oracle Message Brokers

When using a local Oracle Message Broker with either the TIBCO Driver or the Oracle Multicast Driver to implement the stock quote application, all publishers and subscribers use their own local copy of the Oracle Message Broker. All local Oracle Message Brokers use the same OMB instance in the directory, and thus define the same topics. Quotes sent by publishers are directly multicast to all subscribers without the intervention of a central Oracle Message Broker.

Figure 5–2 shows the stock quotes application configured with the `local` attribute set to true. This local architecture is decentralized, does not present a single point of failure, and scales to any number of subscribers. The local architecture is inherently scalable, since it uses multicast-based communication: multicast communication allows a single message to be sent to many destinations as a single operation, without using a sequence of point-to-point messages.

Figure 5–2 Local Mode Oracle Message Broker Quote System



Running in Remote Mode

The Oracle Message Broker provides two modes of operation, Remote Mode and Local Mode (Remote Mode is also called Non-Local Mode). When configured in Local Mode, one active Oracle Message Broker is created in each client JVM (that is, in each program that uses the JMS API). When configured in Remote Mode, Oracle Message Brokers run as a process using their own JVM.

Running in Remote Mode provides the following features:

- Connection pooling in the AQ driver provides a fixed number of OCI connections that can service a much larger number of application programs. Clients are guaranteed access to the connection pool, even when some clients perform blocking receives.
- Remote Mode Oracle Message Brokers can be accessed by remote processes, and by local processes. A Remote Mode Oracle Message Broker must be used if multiple processes need to share messages using the Volatile Driver.
- Clients may not have the privileges to connect directly to a database. In this case, the Oracle Message Broker may perform the database access.
- Propagation can be enabled – see Chapter 8, "Oracle Message Broker Propagation" for details on using propagation.
- Dynamic Monitoring System (DMS) can be enabled – see "Collecting Runtime Metrics" on page 6-3 for information on displaying runtime metrics in Remote Mode.

Starting Oracle Message Broker in Remote Mode

Table 2–3 shows the environment variables that Oracle Message Broker requires to start in Remote Mode. The environment variables shown are defined in the Oracle Message Broker startup files. See "Working with the Administration Utilities" on page 2-2 for more information on the startup environment.

A remote Oracle Message Broker writes its address, an IOR, to the directory when it starts. The address is used by clients to establish connections, by the propagation manager to transfer messages, and to shutdown the Oracle Message Broker. Refer to "Starting and Stopping the Oracle Message Broker" on page 2-6 for information on the `MsgBroker` command.

Note: If multiple remote Oracle Message Broker use the same OMB Instance, the results may be unpredictable, see the section "Starting Oracle Message Broker Clients in Remote Mode" on page 5-15 for more information.

Starting Oracle Message Broker Clients in Remote Mode

Table 2–3 shows the environment variables that Oracle Message Broker clients need to set to contact the Oracle Message Broker. The environment variables shown are defined in the Oracle Message Broker startup files. See "Working with the Administration Utilities" on page 2-2 for more information on the startup environment.

Remote Mode Limitations

When the Oracle Message Broker is starting in Remote Mode, it checks to see if there is another active Remote Mode Oracle Message Broker process using the same OMB Instance in the same LDAP Directory. If so, the new Oracle Message Broker prints an error message and shuts down. This procedure is intended to prevent two remote Oracle Message Brokers from using the same OMB Instance.

The Oracle Message Broker may fail to detect another active Oracle Message Broker process in two scenarios:

1. If both Oracle Message Broker processes are started at the same time (race condition).
2. If one Oracle Message Broker process is configured to use SSL over IIOP, while the other Oracle Message Broker does not use SSL over IIOP.

Note: If two remote Oracle Message Broker use the same OMB Instance in the LDAP Directory, the second Oracle Message Broker overwrites the first address. This would likely cause problems for one or more clients, and result in unpredictable Oracle Message Broker behavior.

Oracle Message Broker Version Checking

When an Oracle Message Broker JMS client contacts an active Oracle Message Broker, the Oracle Message Broker checks that the client is using a compatible version of the Oracle Message Broker client libraries. If the client is not running with a compatible version, the Oracle Message Broker throws a version exception.

The Oracle Message Broker and the JMS client-side runtime throws an exception if either are started with an unsupported JVM version. Refer to the release notes for information on supported JVM versions.

When the system property oracle.oas.mercury.anyjvm is set, the Oracle Message Broker and the client-side runtime do not check the JVM version.

6

Oracle Message Broker Extensions

This chapter covers features that are not part of the JMS specification. These features are Oracle Message Broker specific features or extensions to the JMS specification.

This chapter covers the following:

Using XML Messages

Oracle Message Broker supports methods that translate between JMS messages and XML with the source being either an XML document or a JMS message. Each message translation retains all information. The translation methods depend on the Document Type Definition (DTD) defined in jms.dtd. The DTD is defined in the directory $OMB_HOME/src on Unix systems or %OMB_HOME%\src on Windows NT.

The jms.dtd provides a complete mapping of JMS message components to XML, including: the message header, standard message properties, and all JMS message body types.

Sending and Receiving XML Messages

Oracle Message Broker supports the following behavior for an application that uses HTTP. The client receives XML data from an HTTP `post` command. The client then uses the Oracle Message Broker translation methods to convert the XML data to JMS messages. If the destination queue or topic were included in the XML data, the client enqueues or publishes the JMS message.

Assuming that HTTP is the transport by which the XML capable application interacts with JMS, the Oracle Message Broker supports the following actions:

- A JMS client could receive HTTP `get` commands that request a message from a queue or topic. The client would perform a non-blocking receive on the queue or topic, and then translate the JMS message to XML, and return the XML data using HTTP.
- The destination queue or topic name is included in the XML data. The client would then enqueue or publish the message.

The XML_to_JMS Method

The `XML_to_JMS` method in the package oracle.oas.mercury.MercuryXML returns a JMS message. Table 6–1 shows the `XML_to_JMS` parameters. Refer to the Javadoc supplied with the Oracle Message Broker for a complete description of the parameters (the Javadoc distributed with the Oracle Message Broker is in the directory $OMB_HOME/doc/javadoc or %OMB_HOME%\doc\javadoc on Windows NT).

Table 6–1 XML_to_JMS parameters

Parameter	Description
xmlDoc	The XML document to be converted to a JMS message.

The JMS_to_XML Method

The `JMS_to_XML` method found in the oracle.oas.mercury.MercuryXML package returns an XML document from the supplied JMS message. Table 6–2 shows the `JMS_to_XML` parameters. Refer to the Javadoc supplied with the Oracle Message Broker for a complete description of the parameters (the Javadoc distributed with the Oracle Message Broker is in the directory $OMB_HOME/doc/javadoc or %OMB_HOME%\doc\javadoc on Windows NT).

Table 6–2 JMS_to_XML parameters

Parameter	Description
msg	The converted JMS message
dtdpath	The dtd describing a JMS message as XML. This should be of the form file:/omb/src/jms.dtd. The path can be changed.

Collecting Runtime Metrics

Oracle Message Broker collects metrics so that you can monitor Oracle Message Broker performance and activity. You can save the Oracle Message Broker metrics using the `MsgBroker -stats` option. Table 6–3 shows the metrics available for queues and topics. Table 6–4 shows the metrics available for Oracle Message Broker instances (for each active msg_broker entry in the directory). Table 6–5 shows the metrics available for the Oracle Message Broker process.

Table 6–3 Oracle Message Broker Queue and Topic Metrics

Metric	Description
message_count	Number of available messages. An available message is a message that can be received by a message consumer. An available message has been sent from a non-transacted session, or from a transacted session where a commit completed after the send. The available message has not been received, or any attempted receives have been rolled back.
receives_attempted	Number of times that a receive has been attempted for a destination. A receive that times out is counted in receives_attempted, but not in receives_completed. A receive that is blocked is counted in receives_attempted.
receives_completed	Number of times receive completed.
sends	Number of times that a message has been sent to a destination.

Table 6–4 Oracle Message Broker Instance Metrics

Metric	Description
memory_used	Number of kilobytes of JVM heap used by the Oracle Message Broker.
percent_memory_free	The percent of the JVM heap used by the Oracle Message Broker.
connections	Number of JMS connections.
commits	Number of times that a transacted session is committed.
rollbacks	Number of times that a transacted session is rolled back.
receives_rejected	Number of times that a blocking receive request is rejected because the maximum number of concurrent requests has been reached.
requests_rejected	Number of requests rejected because the server was low on memory.

Table 6–5 Oracle Message Broker Process Metrics

Metric	Description
uptime	Time in seconds since start of the JVM process running the Oracle Message Broker.
cputime	The cpu time in microseconds used by the JVM process running the Oracle Message Broker.
heapsize	The size of the heap allocated to the JVM process running the Oracle Message Broker.
freemem	Memory in kilobytes allocated by the JVM process running the Oracle Message Broker.

Using DMS

Use the `MsgBroker` command with the `-stats` option to save the collected Dynamic Monitoring Service metrics to the DMS log file. `MsgBroker -stats` saves the DMS log file to a file with the same name as the associated Oracle Message Broker log (omblog) file, prepended with "dms-". Refer to "Working with Log Files" on page 10-1 for information on the omblog file name, and the directory where the DMS log file is written.

The `-stats` option includes a parameter that specifies whether information is appended to the DMS log file, or if an existing log file is replaced. The `-stats` option also specifies the format for the data in the DMS log file. Refer to Table 2–2 on page 2-7 for detailed information on the available `MsgBroker -stats` options.

DMS Format Options - Standard and Pretty

When DMS metrics are saved using `MsgBroker -stats`, the display options are standard format and pretty format. The pretty format allows easy viewing and uses indenting to illustrate the DMS directory hierarchy. The standard format is easy to parse with scripts and prints the full DMS pathnames on every line. Both formats use XML-like tags to delimit the metrics and the timestamps.

Each DMS metric save starts with a DMSDUMP tag as shown below (there may be more than one DMSDUMP saved in a DMS log file):

```
<DMSDUMP version = 2.0>
```

The DMSDUMP tag is always followed by a timestamp, of the form:

<timestamp>*timestamp formatted-timestamp*</timestamp>

Where:

timestamp represents milliseconds since the epoch (Jan 1, 1970)

formatted-timestamp is formatted and readable version of the same timestamp.

The *timestamp* is included for quick ordering and sorting.

Dumps end with a closing DMSDUMP tag of the form:

```
</DMSDUMP>
```

DMS Pretty Format

The following lines contain describe the pretty format data:

```
jspy_uptime.value: 0 secs
testNoun
 testState.value: 55 msecs
 testEvent.count: 3 ops
 testPhase.time: 0 msecs
JDMS
 JVM
  jvmTotalMem.max: 6144.0 kbytes
```

Each line represents either a DMS Noun or Metric. Refer to Example 6–1 for more sample pretty format output.

Noun Format Nouns are single strings on their own. Nouns never contain control characters or whitespace.

Metric Format Metrics have the form:

metricname: *value* [*units*]

metricname fields never contain control characters or whitespace.

value field contains the metric's value. If the metric is a String metric, then this field may contain whitespace. Otherwise it is an integer or floating point number with no whitespace.

units field is optional.

The pretty format indenting indicates containment, so in this case `testState.value` is a child of `testNoun`, and its siblings include `testEvent.count` and `testPhase.time`.

DMS Standard Format

The following lines contain sample standard format data:

```
/jspy_uptime.value: 0
/testNoun/testState.value: 55
/testNoun/testEvent.count: 3
/testNoun/testPhase.time: 0
/JDMS/JVM/jvmTotalMem.max: 6144.0
/JDMS/JVM/jvmTotalMem.min: 6144.0
```

Each line represents a metric. Units are not included. Names for Nouns and Metrics have no control characters or whitespace. Refer to Example 6–2 for more sample pretty format output.

Noun Format With standard format, noun names must be deduced from the metric names. Containment is indicated with the path separator '/' characters.

DMS-defined metrics

In addition to the Oracle Message Broker metrics, a DMS dump may contain any of the following pre-defined metrics:

```
/jspy_uptime.value: -- number of seconds since the process started
/JDMS/JVM/jvmTotalMem.max -- maximum observed JVM heap size
/JDMS/JVM/jvmTotalMem.min -- minimum observed JVM heap size
/JDMS/JVM/jvmTotalMem.value -- current JVM heap size
/JDMS/JVM/jvmFreeMem.max -- maximum observed amount of free JVM heap space
/JDMS/JVM/jvmFreeMem.min -- minimum observed amount of free JVM heap space
/JDMS/JVM/jvmFreeMem.value -- current amount of free JVM heap space
/JDMS/Measurement/nodes.max -- maximum number of metrics+nouns allocated by DMS
/JDMS/Measurement/nodes.value -- # nouns + # metrics allocated by DMS
/JDMS/Measurement/lastID.value -- last noun/metric identifier allocated
/JDMS/Measurement/nounCreate.count -- number of DMS nouns created
/JDMS/Measurement/nounDestroy.count -- number of DMS nouns destroyed
/JDMS/Measurement/sensorCreate.count -- number of DMS sensors created
/JDMS/Measurement/sensorDestroy.count -- number of DMS sensors destroyed
/JDMS/Measurement/sampleVal.count -- # metric values served by DMS so far
/JDMS/Log/archiveCount.count -- number of rollovers for the DMS log
/JDMS/Log/bytesWritten.value -- bytes written by DMS logging
/JDMS/Log/init.count -- number of times that DMS log was initialized
/JDMS/Log/attribute_changed.count -- count of changes to DMS logging attributes
```

Example 6–1 Sample Dump Using Pretty Format

```
<DMSDUMP version = 2.0>
<timestamp>958084102187 (Thu May 11 15:28:22 PDT 2000)</timestamp>

 jspy_uptime.value: 0 secs
 testNoun
  testState.value: 55 msecs
  testEvent.count: 3 ops
 testPhase.time: 0 msecs

 JDMS
  JVM
```

```
    jvmTotalMem.max: 6144.0 kbytes
    jvmTotalMem.min: 6144.0 kbytes
    jvmTotalMem.value: 6144 kbytes
    jvmFreeMem.max: 5665.0 kbytes
    jvmFreeMem.min: 5306.0 kbytes
    jvmFreeMem.value: 5306 kbytes
   Measurement
    nodes.max: 28.0
    nodes.value: 28
    lastID.value: 27
    nounCreate.count: 2 ops
    nounDestroy.count: 0 ops
    sensorCreate.count: 8 ops
    sensorDestroy.count: 0 ops
    sampleVal.count: 150 ops
   Log
    archiveCount.count: 0 ops
    bytesWritten.value: 0 bytes
    init.count: 1 ops
    attribute_changed.count: 0 ops
</DMSDUMP>
```

Example 6–2 Sample DMS Output Using Standard Format

```
<DMSDUMP version = 2.0>
<timestamp>958084102111 (Thu May 11 15:28:22 PDT 2000)</timestamp>
/jspy_uptime.value: 0
/testNoun/testState.value: 55
/testNoun/testEvent.count: 3
/testNoun/testPhase.time: 0
/JDMS/JVM/jvmTotalMem.max: 6144.0
/JDMS/JVM/jvmTotalMem.min: 6144.0
/JDMS/JVM/jvmTotalMem.value: 6144
/JDMS/JVM/jvmFreeMem.max: 5665.0
/JDMS/JVM/jvmFreeMem.min: 5366.0
/JDMS/JVM/jvmFreeMem.value: 5366
/JDMS/Measurement/nodes.max: 28.0
/JDMS/Measurement/nodes.value: 28
/JDMS/Measurement/lastID.value: 27
/JDMS/Measurement/nounCreate.count: 2
/JDMS/Measurement/nounDestroy.count: 0
/JDMS/Measurement/sensorCreate.count: 8
/JDMS/Measurement/sensorDestroy.count: 0
/JDMS/Measurement/sampleVal.count: 106
/JDMS/Log/archiveCount.count: 0
```

```
/JDMS/Log/bytesWritten.value: 0
/JDMS/Log/init.count: 1
/JDMS/Log/attribute_changed.count: 0
</DMSDUMP>
```

AQ Driver Runtime Metrics

The AQ Driver optionally exports Oracle AQ specific statistics using DMS. Since DMS monitoring can consume processor and other resources, by default the AQ specific metric collection is disabled.

> **Note:** AQ Driver runtime metrics are always disabled when the Oracle Message Broker runs in Local Mode.

Table 6–6 shows the available AQ Driver runtime metrics.

Table 6–6 AQ Driver Runtime Metrics

AQ Driver Metric	Description
deferred_count	Number of messages with deferred delivery time in the future. Applies for queues and topics.
expired_count	Number of messages for which delivery deadline expired. Applies for queues and topics.
ready_count	Number of messages that can be received. Applies for queues, topics, and subscribers.

The AQ Driver runtime metrics are collected for JMS queues and topics (multi-consumer AQ queues and single-consumer AQ queues). AQ implements a multi-consumer queue with:

- A base table with one row per message.
- A subscriber table with one row per subscriber.
- An index table with one row per pair (message_id, subscriber_id)

The per-subscriber metric `ready_count` is computed by querying the index table. The per-topic metrics (`ready_count`, `expired_count`, `deferred_count`) are computed by querying the basetable. If the `ready_count` for a topic is 0, then the `ready_count` for all subscribers to that topic will be 0. If the `ready_count` for a topic is 10, the `ready_count` for any subscriber to that topic will be less than or equal to 10 (the `ready_count` for all subscriber can be 10).

It is possible for schemas to show no queues or topics when there are queues / topics in that schema. This occurs when the username / password used to establish the JDBC connection by the AQ driver does not have access to execute queries on the tables created by AQ within that schema.

AQ Driver Runtime Metric Administration

Table 6–7 shows the Java properties that control AQ Driver runtime metric collection. The Oracle Message Broker does not support LDAP Directory based administration for AQ Driver runtime metrics.

Table 6–7 AQ Driver Runtime Metric Properties

Property	Description
oracle.oas.mercury.dmsaq	To enable the AQ Driver specific DMS metrics, set the Java System property to true. The AQ DMS runtime metrics are disabled by default because the DBMS queries consume resources.
oracle.oas.mercury.dmsaqInterval	The default interval at which AQ Driver queries the DBMS to find new AQ queues, single-consumer and multi-consumer, for DMS monitoring is 120 seconds. Set the Java System property to an integer value to adjust the query interval. The accepted range for the DMS query interval is between 10 and 600 seconds. When an invalid value is specified, the default value is used. It determines the interval at which the AQ driver queries the DBMS to determine which AQ queues have been created. It also determines the interval at which the AQ driver updates the DMS resource hierarchy. The resource hierarchy must be updated when a subscriber is added / deleted and when AQ queues are added / deleted

Creating Destinations

Oracle Message Broker clients can use several methods to create destinations that are not JMS administered objects. A client uses these destinations to send or to publish messages without referencing the Oracle Message Broker administrative directory. Refer to the JMS specification, section 4.4.4, for a brief description of domain-specific destinations.

The methods Oracle Message Broker provides to create and use destinations are:

`QueueSession.createQueue` to create a queue

`TopicSession.createTopic` to create a topic

Defining Destination Strings

The `createQueue` and `createTopic` methods each take a `String` argument that defines a destination. To describe a destination, use the following syntax for the destination string:

tag=value[,*tag=value*]*

The order of tags is not significant, nor is the case. The destination string syntax does not allow whitespace, embedded commas, or embedded "=" signs.

The `createQueue` and `createTopic` methods each require two *tags*: name and driver. The name and driver tag are defined as follows:

name	The destination name
driver	The driver name. The driver name must be one of the following: `aq`, `aqlite`, `vol`, `mq`, `zyg`, or `rv`.

In addition, when the driver is `mq`, the following tags are also required:

queue	The MQ specific queue name
manager	The name of the queue manager

For example, the following code creates an MQ Series queue using `createQueue`:

```
QueueSession qs = ...;
Queue q = qs.createQueue("driver=mq,name=q1,manager=mgr1,queue=mqq");
```

The following code creates a Volatile queue using `createQueue`:

```
QueueSession qs = ...;
Queue q = qs.createQueue("driver=vol,name=volq1");
```

Using Client-Side Callouts

Oracle Message Broker provides a facility for creating and using client-side callouts. Client-side callouts define message transformations or other user defined processing that is applied when messages are produced or consumed. Callout methods can be written in Java or C/C++.

A Callout method defines the transformations or processing that a message consumer performs before receiving messages. Likewise, message producers can also use callout methods to perform transformations or processing before sending messages. Since callout methods execute on the client-side, the Oracle Message Broker knows nothing of the transformations or of the callout methods.

This section covers the following:

Defining Callout Methods

Oracle Message Broker supports both Java and C/C++ callout methods. Use the interfaces shown in this section to define callout methods. Example 6–3 shows a sample Java stub for a callout method.

Defining Java Callouts

To define a callout method in Java, implement the following Oracle Message Broker client-side interface (defined in oracle.oas.mercury.jmsClient.callout):

```
interface Callout
{
     public Message invoke(Message message) throws JMSException;
}
```

Example 6–3 Java Callout Sample Stub

```
import oracle.oas.mercury.jmsClient.callout.Callout;
class mapName implements Callout
{
public Message invoke(Message message)
{
/* Include invoke method code for MapName message transformation. */
return message;
}
```

To supply parameters to a callout method, a message producer sets the JMSType message header field to indicate the message type. A message producer can set this value to indicate the message type and the callout method can use this field as a flag for callout transformation processing.

C/C++ Callouts

To define a callout method in C/C++, implement the following Oracle Message Broker JNI. Use this JNI to create a dynamic library for each callout and include the following in the callout source:

```
#include <jni.h>
#ifdef __cplusplus
extern "C" {
#endif
JNIEXPORT void JNICALL Java_oracle_oas_mercury_jmsClient_callout_CMaps_invoke
(JNIEnv jenv*, jobject javathis, jobject message);
#ifdef __cplusplus
}
#endif
```

This is the signature for the native method. Implement the callout transformation with this signature and then create a dynamic library. The name of the dynamic library is the same as the name of the callout.

To supply parameters to a callout method, a message producer sets the JMSType message header field to indicate the message type. A message producer can set this value to indicate the message type and the callout method can use this field as a flag for callout transformation processing.

For additional information on callouts, refer to the README file in the sample directory $OMB_HOME/samples/client/java/callout.

Using Callouts in a Message Producer

To register a callout in a message producer, either a QueueSender or a TopicPublisher, use the following method:

```
void MercuryProducer.setCallout(String calloutName) throws CalloutException
```

For example:

```
((MercuryProducer)sender).setCallout(pcallout);
```

To unregister a callout, use the following method:

```
MercuryProducer.setCallout(null);
```

To find the name of any registered callouts, use the following method:

```
String MercuryProducer.getCallout()
```

This returns a registered callout or null if a callout is not registered.

A client can use the following method to handle exceptions and return the original message:

```
Message MercuryProducer.getMsg()
```

This returns the original message, as it was before the callout transformation was attempted. The `getMsg` method returns a null value if no message was supplied to the callout routine.

The `setCallout` method specifies a transformation method, or a handler method that hands off messages to the appropriate transformation method.

When `setCallout` is invoked with a name, for example *calloutName*, the callout manager checks to see if a Java class, *calloutName*, can be found. If the name is available, the class is dynamically loaded. If the class is not available, the callout manager attempts to load a dynamic library with the name *calloutName*. If this fails the callout manager throws a CalloutException and the callout is set to null. Subsequent messages do not use any callouts until `setCallout` successfully completes.

After a queue sender calls `send` or a topic publisher calls `publish`, the Oracle Message Broker JMS client-side code sends messages to the CalloutManager, which invokes the transformation methods registered with `setCallout(..)`, and returns the transformed message, which is then passed on to the Oracle Message Broker. If the callout fails, the callout manager throws a callout exception that contains the original message.

For additional information on callouts and samples showing callouts, refer to the README file in the sample directory $OMB_HOME/samples/client/java/callout.

Using Callouts in a Message Consumer

To register a callout in a message consumer, either a QueueReceiver or a TopicSubscriber, use the following method:

```
void setCallout(String calloutName) throws CalloutException
```

To unregister a callout, use the following method:

```
MercuryConsumer.setCallout(null);
```

To find the name of any registered callouts, use the following method:

```
String MercuryConsumer.getCallout()
```

This returns a registered callout or null if no callout is registered.

Handle exceptions using the following method:

```
Message CalloutException.getMsg()
```

This returns the original message, as it was before the callout transformation was attempted. The `getMsg` method returns a null value if no message was supplied to the callout routine.

The `setCallout` methods specifies a transformation method, or a handler method that hands off messages to the appropriate transformation method.

When `setCallout` is invoked with a name, for example *calloutName*, the callout manager checks to see if a Java class, *calloutName*, can be found. If the name is available, the class is dynamically loaded. If the class is not available, the callout manager attempts to load a dynamic library with the name *calloutName*. If this fails the callout manager throws a CalloutException and the callout is set to null. Subsequent messages do not use any callouts until `setCallout` successfully completes.

After a queue receiver calls `receive` or a topic subscriber calls `receive`, the Oracle Message Broker JMS client-side code sends messages to the callout manager, which invokes the transformation method registered with setCallout(..), and returns the transformed message, which is then passed on to the Oracle Message Broker. If the callout fails, the callout manager throws a callout exception that contains the original message.

For additional information on callouts, refer to the README file in the sample directory $OMB_HOME/samples/client/java/callout.

Using Properties to Indicate Callouts

When the message consumer or producer does not indicate a callout method by explicitly using `setCallout`, the application can use a Java property to specify a default callout method for either a producer or a consumer.

A message consumer can indicate a routine to be called upon receipt of messages by setting oracle.oas.mercury.callout.consumer. A message producer can indicate a method to be called when sending or publishing messages with the property oracle.oas.mercury.callout.producer.

For example the following property sets a callout method that specifies that the client `TestClient` should use the consumer callout method named *myMap* for callout transformations:

```
java -Doracle.oas.mercury.callout.consumer=myMap TestClient
```

Each message is sent to the user-indicated callout. By default, if the message consumer or producer does not explicitly specify a callout transformation, the Java property oracle.oas.mercury.callout.producer is checked. If this property is set, and the callout exists, that callout is used. Otherwise, no callout transformation is attempted.

To supply parameters to a callout method, a message producer sets the JMSType message header field to indicate the message type. A message producer can set this value to indicate the message type and the callout method can use this field as a flag for callout transformation processing.

For additional information on callouts, refer to the README file in the sample directory $OMB_HOME/samples/client/java/callout.

Sample Client Side Callout Programs

For additional information on callouts and examples showing callouts, refer to the README file in the sample directory $OMB_HOME/samples/client/java/callout.

Universal Connections and Universal Sessions

The Oracle Message Broker supports a JMS connection and session extension called Universal Connections and Universal Sessions. This extension allows a JMS connection to support both topic and queue connections with a single, Universal Connection. Likewise, a JMS session can support both topic and queue sessions with a single Universal Session.

This extension is implemented as follows, for connections and sessions:

- Instances of `javax.jms.TopicConnection` and `javax.jms.QueueConnection` returned from calls to Oracle Message Broker's ConnectionFactories implement the interface `oracle.oas.mercury.MercuryConnection`.
- The interface `oracle.oas.mercury.MercuryConnection` implements both `TopicConnection` and `QueueConnection`. Thus, you can cast a newly created connection, either topic or queue, to `MercuryConnection`.
- Instances of `oracle.oas.mercury.MercuryConnection` can create both `TopicSessions` and `QueueSessions`.
- Instances of `javax.jms.QueueSession` and `javax.jms.TopicSession` returned from calls to the Oracle Message Broker's version of TopicConnection, `QueueConnection`, or `MercuryConnection` can be cast to the type `oracle.oas.mercury.MercurySession`. `MercurySession` implements both `TopicSession` and `QueueSession`.

With this feature, you only need to create one connection, either a topic connection or a queue connection. The result can be cast to type `MercuryConnection`. The `MercuryConnection` implements both `javax.jms.TopicConnection` and `javax.jms.QueueConnection` and can be used wherever a `javax.jms.*Connection` is expected.

Likewise, you only need to create one session, either a topic session or a queue session. The result can be cast to type `MercurySession`. `MercurySession` implements both `javax.jms.TopicSession` and `javax.jms.QueueSession` and can be used wherever a `javax.jms.*Session` is expected.

Thus, using Universal Connections and Universal Sessions, access to a queue and a topic can be performed in the same transaction.

Receiving with a Message ID

The Oracle Message Broker provides a JMS extension to receive messages by specifying a message ID. This is facilitated by using the `receive` method for a message consumer. The receive returns a Message. Refer to the Oracle Message Broker Javadoc for the package `oracle.oas.mercury.MercuryConsumer` for a full description of `receive` (the Javadoc is available in the directory $OMB_HOME/doc/javadoc, or %OMB_HOME%\doc\javadoc on Windows NT).

Using AQ Rules for Message Selection

The Oracle Message Broker supports two types of message selectors when messages are stored using the AQ Driver, including:

- Standard Oracle Message Broker messages selectors (refer to "Using Message Selectors" on page 5-3 for more information).
- AQ Rules based messages selectors. AQ Rules based message selectors are passed to Oracle AQ when an Oracle Message Broker JMS durable subscription is created for a topic using the AQ Driver (an Oracle AQ multi-consumer queue). AQ Rules based message selectors use Oracle AQ features to select specific messages that are stored in an AQ multi-consumer queue (JMS topic).

> **Note:** The Oracle Message Broker only supports AQ Rules based messages selectors for messages stored using the AQ Driver.

The benefits of using AQ Rules include:

- Scalability – AQ Rules provide better performance when there are many subscribers for a single topic.
- Lower storage requirements – Using AQ Rules, the rules are evaluated on enqueue. This allows Oracle AQ to avoid storing a message when the message does not satisfy any of the specified selectors. The Oracle Message Broker evaluates standard message selectors on dequeue, so with standard message selectors, all messages are stored in the underlying message store.
- Durability – With AQ Rules based message selectors, the message selector is as durable as the subscription. The Oracle Message Broker records both the AQ Rule based message selector and the durable subscription in the underlying Oracle 8*i* database tables representing the queue.
- Better selectors – AQ Rules accepts a more general syntax for message selectors. AQ Rules can also access a message body, while standard Oracle Message Broker message selectors cannot use the message body.
- AQ rules can call stored procedures supplied by Oracle or created by an Oracle 8*i* Database Server user.

Creating AQ Rules Based Message Selectors

Oracle Message Broker standard message selectors are specified according to the syntax in the JMS specification (refer to "Using Message Selectors" on page 5-3 for more information). The AQ Rules engine accepts a more general message selectors syntax. Application developers create AQ Rules during the development of Oracle Message Broker applications.

To create AQ Rules based message selectors, application developers must understand the following:

- Oracle8 ADTs used to store Oracle Message Broker messages (see Appendix A, "Oracle AQ Driver ADTs").
- PL/SQL (see the Oracle 8*i* Database Server documentation).
- PL/SQL helper functions (see "PL/SQL Functions Supporting AQ Rules" on page 6-21). The Oracle Message Broker support for AQ Rules requires developers to access property values explicitly using function calls rather than implicitly as described in the JMS specification. The helper functions assist developers in accessing property values.
- Oracle AQ Rules. See the Oracle 8i Documentation Library for complete information on Oracle AQ and AQ Rules.

Message Selector Format

When Oracle Message Broker applications use AQ Rules based message selectors, the message selector must be a string. The string must be constructed so that it would be valid as the WHERE clause of a SQL statement. The message selector string is also restricted; it can only access columns in the underlying AQ queue table representing the JMS topic that the AQ Rule is to be applied to.

To determine the available columns that AQ Rules can be applied to, execute the following SQLPLUS command:

```
sqlplus> DESCRIBE queue_table
```

Where *queue_table* is the name of the queue table in which the JMS messages are stored (a queue table holds an AQ queue that stores a JMS topic in the Oracle 8*i* Database). When the AQ queue table is created, the Oracle Message Broker administrator specifies the Oracle8 ADT for the *queue_table*'s user_data column. The types for all other *queue_table* columns are fixed for each queue table. Table 7–2 and Table 7–3 show the available types for the Oracle Message Broker AQ based queue table user_data column.

Table 6–8 describes how the ADT specified for the underlying AQ queue table affects the parts of JMS messages that can be selected using an AQ Rules based message selector. Each entry in Table 6–8 lists the type of the AQ Driver and the parts of the JMS message that can be accessed using a message selector for the specified ADT. All queues that Oracle Message Broker accesses use one of the eight data types for the user_data column shown in Table 6–8.

Keep the following in mind for the Exposes field in Table 6–8.

- Exposes nothing means the message is stored as a stream of bytes serialized in Java. The message is stored in the Database Server as a blob and should not be accessed using PL/SQL or AQ Rules based message selectors.
- Exposes message body and properties means the type used to create the queue table is an Oracle8 JMS ADT (see Appendix A, "Oracle AQ Driver ADTs"). Message properties and the message body can be accessed using PL/SQL or AQ Rules based message selectors.

Table 6–8 JDBC AQ Driver Queue Message Options

Oracle8*i* ADT Used for Queue Table	Driver	Message Types Allowed	Exposes
OMBAQ_BYTES_MSG	JDBC AQ	bytes	Exposes message body and properties
OMBAQ_MAP_MSG	JDBC AQ	map	Exposes message body and properties
OMBAQ_OBJECT_MSG	JDBC AQ	object	Exposes message body and properties
OMBAQ_STREAM_MSG	JDBC AQ	stream	Exposes message body and properties
OMBAQ_TEXT_MSG	JDBC AQ	text	Exposes message body and properties
OMBAQ_SERIAL_MSG	JDBC AQ	All JMS Message Types	Exposes nothing
RAW	OCI AQ	Non-queriable	Exposes nothing
OMBAQ_MESSAGE_T	OCI AQ	Queriable	Exposes the message body. Does not expose the message properties.

PL/SQL Functions Supporting AQ Rules

This section provides tables showing the PL/SQL helper functions that the Oracle Message Broker provides for developers to use AQ Rules as message selectors. The helper functions are part of the aq.ombaq PL/SQL package that is included with the Oracle Message Broker installation. Each entry in Table 6–9, Table 6–10, Table 6–11, Table 6–12, and Table 6–13 shows a helper function, with its description, and an example showing how to use the function. The tables listing helper functions are specific to each message type ADT, as specified in Table 6–8.

An AQ Rule rule is specified as a boolean expression (one that evaluates to true or false) using syntax similar to the WHERE clause of a SQL query. This boolean expression can include conditions on:

1. The fixed columns from queue table. This allows an AQ Rule to access the fixed columns from a queue table as part of the AQ Rule, regardless of the type of the user_data column. For example, the following example uses the priority and corrid columns from a queue table:

   ```
   tab.priority > 2 AND tab.corrid = 'PAYMENT'
   ```

2. The JMS message body, as stored in the queue table. The helper functions can assist in providing access to the JMS message body, as stored in the queue table with Oracle Message Broker ADTs, for particular message types, as shown in Table 6–8.

3. The JMS message properties, as stored in the queue table. The helper functions can assist in providing access to the JMS message properties, as stored in the queue table with Oracle Message Broker ADTs, for particular message types, as shown in Table 6–8.

4. The message body and the message properties can also be explicitly accessed from the user_data column. For example, the following example accesses properties from the user_data column:

   ```
   tab.user_data.ombaq_property.name = 'X.X'
   ```

Using Helper Functions

When using the PL/SQL helper functions, the function name must be fully specified. For example, `aq.ombaq.str_text` is fully specified. However, the function, `str_text` is not fully specified.

The helper functions can only be used with queue table user_data columns of the following types:

- OMBAQ_MAP_MSG
- OMBAQ_STREAM_MSG
- OMBAQ_TEXT_MSG
- OMBAQ_BYTE_MSG
- OMBAQ_OBJECT_MSG

Some functions return an integer when they should return a boolean. Functions that return a boolean cannot be used with AQ Rules for an AQ subscriber. To overcome this limitation, these functions return 0 to indicate false and 1 to indicate true.

Number Property Helper Functions The num_prop functions search the varray in which properties are stored and return the property that matches the specified name if the property value is stored as a SQL Number.

The num_prop functions return null if there is no property element with a matching name or if there is an element that matches, but the value is stored as a SQL varchar2. Properties in a JMS message with the following types are stored as a SQL Number:

- boolean
- byte
- short
- integer
- long
- float - float values may lose precision when converted from a Java float to a SQL Number and then back to a Java float.

String Property Helper Functions The str_prop functions search the varray in which properties are stored and return the property that matches the specified name if the property is stored as a SQL varchar2.

The str_prop functions return null if there is no property element with a matching name or if there is an element that matches, but the value is stored as a SQL

number. Properties in a JMS message with the following types are stored as a SQL varchar2:

- double - doubles are stored as a string since SQL Number cannot store the valid range of a Java double.
- string

Type of Property Helper Functions The type_prop functions search the varray in which properties are stored and return a SQL integer that is the value of the ombaq_type field for the ombaq_property varray element. The value of the ombaq_type field determines whether str_prop or num_prop should be used to access the value of the property. The values for the types are defined in the package aq.ombaq. The type_ prop functions return:

- The value of the ombaq _type field if there is a property element that matches and the property element is valid. The symbolic values for the types are defined in the aq.ombaq PL/SQL package installed with Oracle Message Broker.
- Null otherwise

In Property Test Helper Functions The in_prop functions search the varray in which the properties are stored and return a SQL integer. The returned value is:

- 1 if there is a name that matches and the property element is valid.
- 0 otherwise.

Functions for aq.ombaq_bytes_msg Messages

Table 6–9 shows the AQ Rules message selector helper functions for messages stored in queue tables of type `OMBAQ_BYTES_MSG` ADT.

Table 6–9 OMBAQ_BYTES_MSG Helper Functions

Function	Description

num_prop	`function num_prop(msg in ombaq_bytes_msg, name in varchar2) return number;`
	
	This function returns the value of a property that would be stored as a SQL Number type.
	`aq.ombaq.num_prop(tab.user_data, 'shortProperty') = 13`
	This example selects messages that have a property stored as a SQL Number with a value of 13 and the name 'shortProperty'.

str_prop	`function str_prop(msg in ombaq_bytes_msg, name in varchar2) return varchar2;`
	
	This function returns the value of a property that would be stored as a SQL varchar2 type.
	`aq.ombaq.str_prop(tab.user_data, 'stringProperty') = 'foo'`
	This example select messages that have a property stored as a SQL varchar2, the value 'foo', and the name 'stringProperty'.

in_prop	`function in_prop(msg in ombaq_bytes_msg, name in varchar2) return integer;`
	
	This function returns 1 if there is a property element with the specified name and 0 otherwise.
	`aq.ombaq.in_prop(tab.user_data, 'doNotSelectProperty') == 0`
	This example selects messages that do not contain a property with the name 'doNotSelectProperty'.

type_prop	`function type_prop(msg in ombaq_bytes_msg, name in varchar2) return integer;`
	
	This returns the value of the ombaq_type field for the property element with the specified name and null otherwise. The symbolic values for the types are defined in the aq.ombaq PL/SQL package installed with Oracle Message Broker.
	`aq.ombaq.type_prop(tab.user_data, 'booleanProperty') = 1`
	This example selects messages that have a boolean property with the name 'booleanProperty'.

Functions for aq.ombaq_map_msg Messages

Table 6–10 shows the AQ Rules message selector helper functions for messages stored in queue tables with the `OMBAQ_MAP_MSG` ADT type.

The functions num_map, str_map, in_map, and type_map are similar to the functions num_prop, str_prop, in_prop, and type_prop. The difference is the set of values that are searched. Functions of the form *_map search the varray that contains the map elements from a map message (the body of a map message).

Functions of the form *_prop search the varray that contains the message properties.

Table 6–10 OMBAQ_MAP_MSG Helper Functions

Function	Description

num_map	`function num_map(msg in ombaq_map_msg, name in varchar2) return number;`
	
	This function returns the value of a map element if the value is stored as a SQL Number, the name of the map element matches the specified name, and the map element is valid. Map values with the following Java types from a JMS message are stored as a SQL Number: boolean, byte, short, integer, long, float, char.
	
	Note: float values may lose precision when converted from a Java float to a SQL Number and then back to a Java float.
	
	`aq.ombaq.num_map(tab.user_data, 'mapElemName') = 13`
	
	This example selects messages that have a map element stored as a SQL Number with a value of 13 and the name 'mapElemName'.

str_map	`function str_map(msg in ombaq_map_msg, name in varchar2) return varchar2;`
	
	This function returns the value of a map element if the value is stored as a SQL varchar2, the name of the map element matches the specified name, and the map element is valid. Map values with the following Java types from a JMS message are stored as a SQL varchar2: double, string, byte array.
	
	`aq.ombaq.str_map(tab.user_data, 'stringMapElem') = 'foo'`
	
	This example selects messages that have a map element stored as a SQL varchar2, the value 'foo', and the name 'stringMapElem'.

in_map	`function in_map(msg in ombaq_map_msg, name in varchar2) return integer;`
	
	This function returns 1 if there is a valid map element with a name field that matches the specified name. Otherwise, it returns 0.
	
	`aq.ombaq.in_map(tab.user_data, 'doNotSelectElem') == 0`
	
	This example selects messages that do not contain a map element with the name 'doNotSelectElem'.

Table 6–10 (Cont.) OMBAQ_MAP_MSG Helper Functions

Function	Description

type_map	`function type_map(msg in ombaq_map_msg, name in varchar2) return integer;`
	
	This function return the value of the ombaq_type field of a map element if there is a valid map element that matches the specified name. Otherwise, it returns null. The symbolic values for the types are defined in the aq.ombaq PL/SQL package installed with Oracle Message Broker.
	`aq.ombaq.type_map(tab.user_data, 'booleanMapElem') = 1`
	This example selects messages that have a boolean map element with the name 'booleanMapElem'.

num_prop	`function num_prop(msg in ombaq_map_msg, name in varchar2) return number;`
	
	This function returns the value of a property that would be stored as a SQL Number type.
	`aq.ombaq.num_prop(tab.user_data, 'shortProperty') = 13`
	This example selects messages that have a property stored as a SQL Number with a value of 13 and the name 'shortProperty'.

str_prop	`function str_prop(msg in ombaq_map_msg, name in varchar2) return varchar2;`
	
	This function returns the value of a property that would be stored as a SQL varchar2 type.
	`aq.ombaq.str_prop(tab.user_data, 'stringProperty') = 'foo'`
	This example selects messages that have a property stored as a SQL varchar2, the value 'foo', and the name 'stringProperty'.

in_prop	`function in_prop(msg in ombaq_map_msg, name in varchar2) return integer;`
	
	This function returns 1 if there is a property element with the specified name and 0 otherwise.
	`aq.ombaq.in_prop(tab.user_data, 'doNotSelectProperty') == 0`
	This example selects messages that do not contain a property with the name 'doNotSelectProperty'.

type_prop	`function type_prop(msg in ombaq_map_msg, name in varchar2) return integer;`
	
	This returns the value of the ombaq_type field for the property element with the specified name and null otherwise. The symbolic values for the types are defined in the aq.ombaq PL/SQL package installed with Oracle Message Broker.
	`aq.ombaq.type_prop(tab.user_data, 'booleanProperty') = 1`
	This example selects messages that have a boolean property with the name 'booleanProperty'.

Functions for aq.ombaq_object_msg Messages

Table 6–11 shows the AQ Rules message selector helper functions for messages stored in queue tables with the OMBAQ_OBJECT_MSG ADT type.

Table 6–11 OMBAQ_OBJECT_MSG Helper Functions

Function	Description

num_prop	`function num_prop(msg in ombaq_object_msg, name in varchar2) return number;`
	
	This function returns the value of a property that would be stored as a SQL Number type.
	`aq.ombaq.num_prop(tab.user_data, 'shortProperty') = 13`
	
	This example selects messages that have a property stored as a SQL Number with a value of 13 and the name 'shortProperty'.

str_prop	`function str_prop(msg in ombaq_object_msg, name in varchar2) return varchar2;`
	
	This function returns the value of a property that would be stored as a SQL varchar2 type.
	`aq.ombaq.str_prop(tab.user_data, 'stringProperty') = 'foo'`
	
	This example selects messages that have a property stored as a SQL varchar2, the value 'foo', and the name 'stringProperty'.

in_prop	`function in_prop(msg in ombaq_object_msg, name in varchar2) return integer;`
	
	This function returns 1 if there is a property element with the specified name and 0 otherwise.
	`aq.ombaq.in_prop(tab.user_data, 'doNotSelectProperty') == 0`
	
	This example selects messages that do not contain a property with the name 'doNotSelectProperty'.

type_prop	`function type_prop(msg in ombaq_object_msg, name in varchar2) return integer;`
	
	This returns the value of the ombaq_type field for the property element with the specified name and null otherwise. The symbolic values for the types are defined in the aq.ombaq PL/SQL package installed with Oracle Message Broker.
	`aq.ombaq.type_prop(tab.user_data, 'booleanProperty') = 1`
	
	This example selects messages that have a boolean property with the name 'booleanProperty'.

Functions for aq.ombaq_stream_msg Messages

Table 6–12 shows the AQ Rules message selector helper functions for messages stored in queue tables with the OMBAQ_STREAM_MSG ADT type.

The functions num_stream, str_stream, and type_stream are similar to the functions num_prop, str_prop, and type_prop. The difference is the set of values that are searched and the argument use to specify how elements are matched. Functions of the form *_stream search the varray that contains the stream elements from a stream message (the body of a map message). Functions of the form *_prop search the varray that contains the message properties. Also, the *_stream function accept an integer that represents the index within a stream rather than a name. Since stream elements do not contain a name field, the only way to specify a stream element is to specify its position within the stream varray. The first element in a varray is at index 1, not at index 0.

Table 6–12 OMBAQ_STREAM_MSG Helper Functions

Function	Description

num_stream	`function num_stream(msg in ombaq_stream_msg, stream_index in integer) return number;`
	
	This function returns the value of the stream element at index 'stream_index' within the varray if the value is stored as a SQL Number, and the stream element is valid. Stream values with the following Java types from a JMS message are stored as a SQL Number: boolean, byte, short, integer, long, float, char.
	
	Note: float values may lose precision when converted from a Java float to a SQL Number and then back to a Java float.
	
	`aq.ombaq.num_stream(tab.user_data,1) = -1`
	
	This example selects messages that have a stream element at index 1 within the stream stored as a SQL Number, the value -1, and the name 'numStreamElem'.

str_stream	`function str_stream(msg in ombaq_stream_msg, stream_index in integer) return varchar2;`
	
	This function returns the value of a stream element at index 'stream_index' within the varray if the value is stored as a SQL varchar2 and the map element is valid. Stream values with the following Java types from a JMS message are stored as a SQL varchar2: double, string, byte array.
	
	`aq.ombaq.str_stream(tab.user_data, 1) = 'foo'`
	
	This example selects messages that have a stream element at index 1 within the stream stored as a SQL varchar2, the value 'foo', and the name 'stringStreamElem'.

Table 6–12 (Cont.) OMBAQ_STREAM_MSG Helper Functions

Function	Description

type_stream	`function type_stream(msg in ombaq_stream_msg, stream_index in integer) return varchar2;`
	
	This function return the value of the ombaq_type field of a stream element at index 'stream_index' within the varray if the stream element is valid. Otherwise, it returns null.
	`aq.ombaq.type_stream(tab.user_data, 2) = 6`
	This example selects messages that have a stream element at index 2 within the stream with type float.

num_prop	`function num_prop(msg in ombaq_stream_msg, name in varchar2) return number;`
	
	This function returns the value of a property that would be stored as a SQL Number type.
	`aq.ombaq.num_prop(tab.user_data, 'shortProperty') = 13`
	This example selects messages that have a property stored as a SQL Number with a value of 13 and the name 'shortProperty'.

str_prop	`function str_prop(msg in ombaq_stream_msg, name in varchar2) return varchar2;`
	
	This function returns the value of a property that would be stored as a SQL varchar2 type.
	`aq.ombaq.str_prop(tab.user_data, 'stringProperty') = 'foo'`
	This example selects messages that have a property stored as a SQL varchar2, the value 'foo', and the name 'stringProperty'.

in_prop	`function in_prop(msg in ombaq_stream_msg, name in varchar2) return integer;`
	
	This function returns 1 if there is a property element with the specified name and 0 otherwise.
	`aq.ombaq.in_prop(tab.user_data, 'doNotSelectProperty') == 0`
	This example selects messages that do not contain a property with the name 'doNotSelectProperty'.

type_prop	`function type_prop(msg in ombaq_stream_msg, name in varchar2) return integer;`
	
	This returns the value of the ombaq_type field for the property element with the specified name and null otherwise.
	`aq.ombaq.type_prop(tab.user_data, 'booleanProperty') = 1`
	This example selects messages that have a boolean property with the name 'booleanProperty'.

Functions for aq.ombaq_text_msg Messages

Table 6–13 shows the AQ Rules message selector helper functions for messages stored in queue tables of type OMBAQ_TEXT_MSG.

Table 6–13 OMBAQ_TEXT_MSG Helper Functions

Function	Description

str_text	`function str_text(msg in ombaq_text_msg) return varchar2;`
	
	This function returns the body of a text message.
	It returns at most 4000 characters. If the body is null, it returns null.
	`aq.ombaq.str_text(tab.user_data) = 'textbody'`
	This example selects messages that have the text body 'textbody'. The result of str_text can be used as the argument to another function call.

size_text	`function size_text(msg in ombaq_text_msg) return integer;`
	
	This function returns the size of the body of a text message. It returns the following values:
	-1 to indicate null
	0 to indicate a zero length body
	> 0 to indicate a non-zero length body
	`aq.ombaq.size_text(tab.user_data) > 100`
	This example selects messages that have a text body with at least 100 characters.

num_prop	`function num_prop(msg in ombaq_text_msg, name in varchar2) return number;`
	
	This function returns the value of a number property with the given name. The returned value is a SQL Number type.
	`aq.ombaq.num_prop(tab.user_data, 'shortProperty') = 13`
	This example selects messages that have a property stored as a SQL Number with a value of 13 and the name 'shortProperty'.

Table 6–13 (Cont.) OMBAQ_TEXT_MSG Helper Functions

Function	Description

str_prop	`function str_prop(msg in ombaq_text_msg, name in varchar2) return varchar2;`
	
	This function returns the value of a string property that is stored as a SQL varchar2 type.
	`aq.ombaq.str_prop(tab.user_data, 'stringProperty') = 'foo'`
	This example selects messages that have a property stored as a SQL varchar2, the value 'foo', and the name 'stringProperty'.

in_prop	`function in_prop(msg in ombaq_text_msg, name in varchar2) return integer;`
	
	This function returns 1 if there is a property element with the specified name and 0 otherwise.
	`aq.ombaq.in_prop(tab.user_data, 'doNotSelectProperty') == 0`
	This example selects messages that do not contain a property with the name 'doNotSelectProperty'.

type_prop	`function type_prop(msg in ombaq_text_msg, name in varchar2) return integer;`
	
	This returns the value of the ombaq_type field for the property element with the specified name and null otherwise.
	`aq.ombaq.type_prop(tab.user_data, 'booleanProperty') = 1`
	This example select messages that have a boolean property with the name 'booleanProperty'.

Selectors for aq.ombaq_serial_msg

The following selectors can be used when the type of the user_data column of the queue table is aq.ombaq_serial_msg. There are no helper functions available for this message type. AQ Rules can use fixed columns from the queue table. For example,

```
tab.priority > 2 and tab.corrid = 'PAYMENT'
```

Selectors for raw

The following selectors can be used when the type of the user_data column of the queue table is aq.ombaq_serial_msg. There are no helper functions available for this message type. AQ Rules can use fixed columns from the queue table. For example,

```
tab.priority > 2 and tab.corrid = 'PAYMENT'
```

Selectors for aq.message_t

The following selectors can be used when the type of the 'user_data' column of the queue table is aq.ombaq_serial_msg. There are no helper functions available for this message type. AQ Rules can use fixed columns from the queue table. For example,

```
tab.priority > 2 and tab.corrid = 'PAYMENT'
```

Obtaining the JDBC Connection in Local Mode

Using the AQ Driver, in JDBC Mode, Oracle Message Broker clients can access the JDBC connection associated with a transacted session, and perform JDBC operations within the transaction. Only when running in local mode, the method, oracle.oas.mercury.MercurySession.getJdbcConnection() allows you to obtain the JDBC connection.

Since all Oracle Message Broker JMS sessions that the Oracle Message Broker creates are instances of MercurySession, any connection that uses the AQ Driver in JDBC mode, with the following requirements, use the method getJdbcConnection():

- The configuration must be set up so that the session uses the JDBC AQ Driver
- The Oracle Message Broker must be running in local mode
- The session must be transacted

If a message listener has been registered for the session, the JDBC connection can only be used when a registered message listener is active for the session. A message listener is active when the 'onMessage' method is executing or on the call stack for a method that is executing.

Note 1: The Oracle Message Broker client must not close the JDBC connection.

Note 2: The Oracle Message Broker client should not commit or rollback the JDBC connection. Instead, it should use javax.jms.Session.commit and javax.jms.Session.rollback.

Sample Code Using a JDBC Connection

The following example uses the getJdbcConnection method in a JMS client.

Example 6–4 Sample JDBC Connection Using getJdbcConnection()

```
private static void addMessage(MercurySession qs,
        QueueSender snd, int i)
        throws Exception
    {
        TextMessage msg = qs.createTextMessage();
        msg.setText("this is message: " + msg.getJMSMessageID());
                                        snd.send(msg);
        Connection        conn = null;
        PreparedStatement stmt = null;
        try
        {
            conn = qs.getJdbcConnection();
            String cmd = "insert into jdbc_tmp_tbl VALUES (?)";
            stmt = conn.prepareStatement(cmd);

            System.out.println("adding messageID: " +
msg.getJMSMessageID());
            stmt.setString(1, msg.getJMSMessageID());

            stmt.executeUpdate();
        }
        catch (Exception e)
        {
            if (stmt != null) { stmt.close(); }
            throw e;
        }

        if ((i % 2) == 0)
        {
            System.out.println("commit session");
            qs.commit();
        }
        else
        {
            System.out.println("rollback session");
            qs.rollback();
        }
    }
```

7

Message Servers and Drivers

Oracle Message Broker includes drivers that access message storage and distribution facilities. Drivers allow client programs to produce and consume messages and allow the Oracle Message Broker to store messages. Oracle Message Broker supports several drivers, each having different characteristics, and depending on the driver, supporting JMS point-to-point or Publish/Subscribe messaging.

This chapter covers the following:

Driver Configuration

Oracle Message Broker administration facilities store driver configuration information. For information on driver configuration using an LDAP Directory, see Chapter 4, "Administration". For information on driver configuration using lightweight configuration, without the LDAP Directory, see Chapter 13, "Lightweight Configuration".

Driver Features Summary

This section summarizes the features of the Oracle Message Broker Drivers in Table 7–1. Where references to the JMS specification are included, they are in the form [*JMS X.X*], where chapter and section number. For additional driver specific limitations and general Oracle Message Broker limitations, refer to the *Oracle Message Broker Release Notes*.

Table 7–1 JMS Features Supported by Oracle Message Broker Drivers

Driver	Persistent Delivery	Non persistent delivery	JMS Publish Subscribe	JMS Point to Point	Queue Browsers	Transacted Sessions	Durable Subscriber
Oracle AQ	✓	No	✓	✓	✓	✓	✓
AQ Lite	✓	No	No	✓	No	✓	No
Volatile	No	✓	✓	✓	✓	✓	✓ (Note 1)
Multicast	No	✓	✓	No	No	No	✓ (Note 1)
MQSeries	✓	✓	No	✓	✓	✓	No
TIBCO	No	✓	✓	No	No	No	✓ (Note 1)

Note 1: Durable Subscribers for the Volatile Driver, Mutlicast Driver, and the TIBCO Driver are only as durable as the process in which the Oracle Message Broker runs.

The JMS specification describes the features shown in the headings of Table 7–1 in the following sections: Persistent Delivery – [*JMS 4.7*], Non-Persistent Delivery [*JMS 4.7*], Publish / Subscribe [*JMS 6*], Point-to-Point [*JMS 5*], Queue Browsers [*JMS 5.1*], Transacted Sessions [*JMS 4.4.7*], Durable Subscriber [*JMS 6.3*].

Refer to the Java Message Service specification available from Javasoft at the following site for more information on JMS:

`http://www.javasoft.com/products/jms`

Oracle Advanced Queuing Driver

The Oracle Advanced Queuing Driver (Oracle AQ Driver) supports persistent delivery of JMS messages using the Oracle 8*i* Database Server Advanced Queueing (AQ) messaging infrastructure (see the *Oracle 8i Application Developers Guide* for more information on AQ).

See the section, "Driver Features Summary" on page 7-2 for a list of the features and limitations of the Oracle AQ Driver.

In addition, the features shown in Table 7–1, the AQ Driver supports Oracle specific extensions that expose functionality unique to Oracle AQ. These extensions include:

- AQ Rules based message selection
- AQ Driver specific runtime metrics

The AQ Driver maps JMS topics to AQ multi-consumer queues and JMS queues to AQ single-consumer queues. Thus, the AQ Driver supports both JMS topics and JMS queues using Oracle AQ queues. The format of the AQ message stored in Oracle AQ is configurable using the Oracle Message Broker administrative facilities. The AQ Driver converts JMS messages to and from AQ messages during enqueues and dequeues respectively.

This section covers the following Oracle Message Broker AQ Driver topics:

AQ Driver Connection Types

The AQ Driver uses one of two connection types to access Oracle AQ: JDBC connections or OCI connections. To support these connection types, the AQ Driver operates in one of two modes: JDBC Mode, or OCI Mode. The Oracle Message Broker administration facilities determine the AQ Driver Mode.

To specify OCI Mode for the OCI connections, or JDBC Mode for the JDBC connections, use the use_jdbc attribute for directory based administration, (see Table 4–9). Using Lightweight configuration, the property oracle.oas.mercury.driver.aq.useJdbc determines the AQ Driver Mode (see Table 13–5 for information on this property).

The benefits of using JDBC connections include:

- This configuration provides a rich set of data types (ADTs) for storing JMS messages. Using these ADTs, programmers can access the messages in AQ using the Oracle Message Broker with the AQ Driver or with other applications, such as PL/SQL.
- Portability - this configuration can use pure Java. Pure Java may be a better fit for most Oracle Message Broker applications, which are usually written in Java.
- Interoperability - the Oracle Message Broker is easier to use for application integration. Oracle Message Broker clients can share AQ queues with native Oracle applications.
- Message management - the message warehousing functionality of Oracle AQ is exposed to Oracle Message Broker clients. JMS messages stored in AQ can be can be queried using SQL.
- With an understanding of the native storage mechanism, PL/SQL applications can enqueue messages into a queue from which Oracle Message Broker JMS clients dequeue messages. PL/SQL applications can also dequeue messages that are enqueued by an Oracle Message Broker JMS client.
- Enables AQ rules - the AQ rules engine is available directly, since all the data in the JMS message that is stored in AQ is queriable.
- Easier debugging - when problems arise, the JDBC configuration is easier to debug, as compared with the OCI configuration which is implemented in C.
- A JDBC connection used to access Oracle AQ can be shared with the client if the Oracle Message Broker is running in Local Mode. This enables the client to access the Oracle Database Server in the same transaction as the Oracle Message Broker's Oracle AQ access.

The benefits of using OCI connections include:

- For some applications, the OCI configuration may provide a performance advantage for large messages. The actual performance characteristics for any given application depend on the size of the messages being sent, the number of messages sent, the tuning and configuration parameter settings, and many other factors.
- The OCI based AQ Driver uses a maximum timeout of 5 seconds for any blocking call, (other than create subscriber and destroy subscriber which can block due to deadlocks in the Database Server). The OCI based driver can be used with a Database Server in MTS mode, and it does not unnecessarily lock resources on the shared servers.

> **Note:** An Oracle Message Broker process can start the Oracle AQ Driver using only one AQ Driver Mode, either the OCI Mode or the JDBC Mode. For an active Oracle Message Broker to change the mode of its AQ Driver, the Oracle Message Broker must be shutdown and then restarted using the new mode.

AQ Messages

The AQ Driver supports several underlying AQ queue table types for storing JMS messages to destinations. The types of underlying AQ queue tables are referred to as Abstract Data Types (ADTs). A destination's configuration determines the ADT that the AQ Driver uses to store a message.

The administrator sets the configuration and assigns an ADT for a destination when the destination, queue or topic, is created. The AQ Driver sends JMS messages to a destination using the specified ADT. If the underlying AQ queue table does not support the ADT that the AQ Driver is sending, the Oracle Message Broker throws an exception.

The message types that the AQ Driver supports depends on the mode the AQ Driver is using. This section describes the message types for both AQ Driver modes.

- JDBC Mode Message Types
- OCI Mode Message Types

JDBC Mode Message Types

Table 7–2 shows the ADTs supported in JDBC mode. Oracle Message Broker supports different AQ queue table ADTs for each JMS message type, plus a generic AQ queue table ADT that supports all JMS messages. A queue created with any of the ADTs other than the generic type can only store one type of JMS message, either: map, stream, bytes, object, or text.

Developers use one of the AQ Driver's JDBC JMS ADTs when they want to do one or more of the following:

- Share queues between the Oracle Message Broker and PL/SQL applications and other Oracle8*i* applications.
- Use the AQ rules engine (see the *Oracle 8i Application Developers Guide* for more information).
- Use Oracle AQ message warehousing features (see the *Oracle 8i Application Developers Guide* for more information).

To access and use the underlying AQ messages using PL/SQL, or other Oracle tools, you need to understand how JMS messages are mapped into AQ queues. Refer to Appendix A, "Oracle AQ Driver ADTs" to for a complete description of the mapping between JMS messages and AQ queue tables.

Table 7–2 JDBC AQ Driver Queue Message Options

Queue Configuration	Description	Oracle8*i* ADT Used for Queue Table
bytes	The underlying AQ Queue only supports JMS bytes messages.	`OMBAQ_BYTES_MSG`
map	The underlying AQ Queue only supports JMS map messages.	`OMBAQ_MAP_MSG`
object	The underlying AQ Queue only supports JMS object messages.	`OMBAQ_OBJECT_MSG`
stream	The underlying AQ Queue only supports JMS stream messages.	`OMBAQ_STREAM_MSG`
text	The underlying AQ Queue only supports JMS text messages.	`OMBAQ_TEXT_MSG`
all	The underlying AQ Queue supports all JMS message types. With this type of AQ Queue, the message is serialized and stored as a stream of bytes. A queue that uses this type can store all message types, but is not queriable.	`OMBAQ_SERIAL_MSG`

Note: the queue configuration specified in the Queue Configuration column is specified using the queue or topic attribute `aq_adt`. See Table 4–17 and Table 4–18 for the list of valid values for the `aq_adt` attribute.

OCI Mode Message Types

Table 7–3 lists the types supported for the AQ Driver in OCI mode.To access and use the underlying AQ messages using PL/SQL, or other Oracle tools, you need to understand how JMS messages are mapped into AQ queues. Refer to Appendix A, "Oracle AQ Driver ADTs" to for a complete description of the mapping between JMS messages and AQ queue tables.

> **Note:** The AQ Driver in OCI mode does not support access to AQ queues created with any of the AQ Driver's JDBC queue configurations.

Table 7–3 OCI AQ Driver Queue Message Options

Queue Configuration	Description	Oracle8*i* ADT Used for Queue Table
raw	The Oracle Message Broker administrator should select raw (non-queriable) mode when there is no need to query the contents of stored messages using SQL. The JMS message is serialized and stored as a raw payload in an AQ message. Non-queriable mode supports all the JMS message types. For non-queriable messages, the maximum message size is restricted only by the Oracle AQ payload size.	`RAW`
queriable	Administrators should select queriable mode to support queriable JMS messages (using SQL). The contents of the JMS message are mapped onto an AQ Driver defined abstract datatype (ADT). Using this type, the JMS properties are not saved.	`OMBAQ_MESSAGE_T`

Note: the queue configuration specified in the Queue Configuration column is specified using the queue or topic attribute `aq_adt`. See Table 4–17 and Table 4–18 for the list of valid values for the `aq_adt` attribute.

AQ Message Priority

When sending messages using the AQ Driver, the JMSPriority field is mapped to an associated AQ message priority (for message types where the JMS properties are retained). Native AQ priorities have a PL/SQL type BINARY_INTEGER, and the priority can be negative. Using the native AQ message system, a priority with a smaller value indicates a higher priority. However, with JMS messages, the valid values for the JMSPriority header fields are one of: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The

values 0 - 4 are normal priority messages, and the values 5 - 9 are expedited priority messages (refer to *[JMS 3.4.10]* for more information).

The Oracle Message Broker converts the priority values as follows, when it is sending / publishing a message using the AQ Driver:

```
Priority stored in AQ = ( 10 - JMS_priority )
```

The Oracle Message Broker converts the priority value as follows, when it is receiving a message using the AQ Driver:

```
JMS_priority = ( 10 - AQ_priority )
```

Thus after a send and receive, the original JMS priority is returned.

JDBC Mode

The AQ Driver in the JDBC mode creates one JDBC connection per JMS session to provide operational access to AQ queues. The Oracle Message Broker also creates the administrative JDBC connections when the Oracle Message Broker is initialized.

The AQ Driver uses the service name stored using the administrative facilities for connecting using JDBC connections. The JDBC driver supports local and host naming, so the service name must be a host name or a local name (for more information, see the *Oracle SQL*Net/DCE Administrator's Guide*).

The AQ Driver creates a pool of JDBC connections to support administrative access (create / destroy subscriptions, create / destroy AQ queues).

The number of JDBC connections used for administrative purposes by the AQ Driver can be specified as a parameter using the Oracle Message Broker administrative facilities (maxSharedSessions). These administrative JDBC connections are used to create durable subscriptions, destroy durable subscriptions, create AQ queues, and destroy AQ queues.

The AQ Driver in JDBC mode supports both transacted and non-transacted JMS sessions.

Obtaining the JDBC Connection in Local Mode

Using the AQ Driver in JDBC Mode, with the Oracle Message Broker running in Local Mode, the JMS Client can obtain a JDBC connection associated with a transaction. For information on obtaining the JDBC connection, see "Obtaining the JDBC Connection in Local Mode" on page 6-32.

OCI Mode

The AQ Driver in the OCI mode uses OCI connections to provide operational access to AQ queues. One Database Server connection is created per JMS session. Using the OCI mode, the AQ Driver and the Oracle Message Broker AQ administrative tools use JDBC connections for administrative access to AQ queues. Oracle Message Broker creates the OCI and JDBC connections when the Oracle Message Broker is initialized.

The AQ Driver uses the service name stored in the directory to establish OCI and JDBC connections. The JDBC driver supports local and host naming, so the service name must be a host name or a local name (for more information, see the *Oracle SQL*Net/DCE Administrator's Guide*).

The AQ Driver supports both transacted and non-transacted JMS sessions. For non-transacted JMS sessions, operations are done in the context of local transactions. If the Oracle Message Broker is running in Local Mode, and it is killed using Control-C, or it dies for any reason, transactions are rolled back by the Database Server as soon as the Database Server notices that the client connections have failed.

The AQ Driver supports JMS queue browsers. A query is executed when the queue browser is created which returns the message IDs of available messages. Messages enqueued after the browser has been created will be returned if retention is enabled for the AQ queue. There is a limit on the number of messages that can be returned by the browser (for information on AQ limitations, refer to the *Oracle Message Broker Release Notes*).

AQ Message Persistence

The AQ Driver supports only JMS persistent delivery mode. When the Oracle Message Broker sends a message using the AQ Driver, it sets the header value for `JMSDeliveryMode` to the value `PERSISTENT`.

Availability (JDBC Mode Only)

When running in JDBC mode, the AQ Driver can recover from Database Server failures. If the Database Server fails, and is then restarted after the Oracle Message Broker has been started:

- New JMS sessions can be created.
- Existing JMS sessions receive a rollback exception but can be used.
- Administrative access is possible using Oracle Message Broker administrative facilities after the DBMS has been restarted.

AQ User Identities

The Oracle AQ Driver supports multiple Database Server schemas. Oracle Message Broker allows schemas other than the default, the 'aq' schema to be used to access the underlying AQ queues. When additional schemas are not needed, the AQ Driver supports the 'aq' user and schema as the default for accessing the Database Server.

The AQ Driver uses two identities for its connections to the Database Server:

1. The admin identity – performs administrative actions with the AQ Driver.

   Administrative actions include:

   - Create or destroy subscribers
   - Create or destroy queues and queue tables
   - Query the number of available messages in a single-consumer queue
   - Query the number of available messages for a subscriber to a multi-consumer queue.
   - Query the names of subscribers at broker startup to a multi-consumer queue.
   - Query the status of a queue at broker startup.

2. The user identity – performs operational actions with the AQ Driver.

   Operational actions include:

   - Enqueue or dequeue messages from an underlying AQ queue.

Admin Identities

When Oracle Message Broker is configured using the LDAP Directory, the administrative identity is set using the following AQ Server entry attributes:

```
aq_username
aq_password
```

When Oracle Message Broker is configured using Lightweight Configuration, the administrative identity is set with the following properties:

```
oracle.oas.mercury.driver.aq.adminUser
oracle.oas.mercury.driver.aq.adminPassword
```

User Identities

Oracle Message Broker clients can specify both a user identity and a password, or neither, when creating JMS connections with javax.jms.createTopicConnection or javax.jms.createQueueConnection.

When the user identity and the password are not supplied, the credentials used to create a Database Server connection to DBMS are those set as follows:

- When Oracle Message Broker is configured using the LDAP Directory, the user identity for operational access is set using the AQ Server entry attributes:

  ```
  aq_username
  aq_password
  ```

- When Oracle Message Broker is configured using Lightweight Configuration, the user identity is set with the properties:

  ```
  oracle.oas.mercury.driver.aq.adminUser
  oracle.oas.mercury.driver.aq.adminPassword
  ```

When the user identity and the password are supplied as arguments in the either the method javax.jms.createTopicConnection, or javax.jms.createQueueConnection, the credentials used to create a Database Server connection to the underlying DBMS are:

- When Oracle Message Broker is configured using the LDAP Directory, the password is used, as supplied in the argument. The user identity is set using the name component of the supplied username argument. The supplied username argument is a full DN for an LDAP Directory user.

  For example, if the username is supplied as an argument in the createTopicConnection or createQueueConnection method, and the LDAP

Directory is used for configuration, then the AQ driver uses the component, "william" as the username:

```
cn=william,cn=users,,cn=omb,cn=products,cn=oraclecontext,<initial-context>
```

- When Oracle Message Broker is configured using Lightweight Configuration, the username and password are used as exactly as supplied in the arguments to javax.jms.createTopicConnection, or javax.jms.createQueueConnection.

Using Different AQ Identities for AQ Administration

To perform administrative access on queues in a schema "foo" when the Oracle Message Broker aq_server username is not equal to "foo" (assume it is "bar" and the queue to be accessed is foo.q1):

1. Grant the role ombadmin to the user bar.
2. Grant select privileges on the queue table in which the queue foo.q1 has been created to the user bar.
3. Grant update privileges on the queue table in which the queue foo.q1 has been created to the user bar.
4. If foo.q1 is a multi-consumer queue, grant select on *table_name*.aq$_q1_s and *table_name*.aq$_q1_i to the user bar.

   Where *table_name* is the name of the queue table in which the queue foo exists.

Using Different AQ Identities for AQ Operational Access

To perform operational access on queues in schema "foo" when the Database Server connection is authenticated with a name other than "foo" (assume it is bar and the queue to be accessed is foo.q1):

1. Grant the role ombuser to the user bar.
2. Grant enqueue and dequeue privileges to bar on the queue foo.q1 using a call to dbms_aqadm.grant_queue_privelege.
3. Specify bar as the username and the correct password in the call to javax.jms.createTopicConnection or javax.jms.createQueueConnection.
4. Grant select privileges on the queue table in which the queuefoo.q1 has been created to the user bar.
5. Grant update privileges on the queue table in which the queue foo.q1 has been created to the user bar.

AQ Tuning and Configuration

An administrator should tune the Database Server if an Oracle Message Broker is performing a large number of administrative operations. However, metrics are not provided to measure the waiting time for AQ related administrative operations. An Oracle Message Broker that performs a large number of administrative operations is likely to generate a large number of DBMS deadlocks since creating and destroying subscriptions requires exclusive locks to resources for which operational access, enqueues and dequeues, requires shared locks. It is safer to perform administrative actions offline, when there is no operational access to the queues.

AQ Queue Configuration

Oracle Message Broker administrative utilities create AQ queues and AQ queue tables. These AQ queues and AQ queue tables have an administrative interface that is available using PL/SQL with the package dbms_aqadm. This PL/SQL package can be used to alter the message retention set for a queue, and to alter other AQ specific table and queue parameters. By default, the use the Oracle Message Broker sets the following values when it creates AQ queues:

```
max_retries = 2147483647 (java.lang.Integer.MAX_VALUE)
retry_delay = 0
retention_time = 0
```

To alter any of these values, use the procedure dbms_aqadm.alter_queue.

AQ Failure Recovery

JMS applications that use the AQ Driver should be able to recover from a Database Server restart without restarting the Oracle Message Broker. The application does need to create new JMS sessions after the Database Server restart.

When the Database Server fails, it may need to be restarted. Consult the Oracle8*i* Documentation to determine when the Database Server must be restarted.

When an Oracle Message Broker client encounters an unexpected Oracle error, the client may need create a new JMS session, which will create a new Database Server connection. The Oracle Message Broker does not have to be restarted when unexpected Database Server errors occur.

AQ Driver Restrictions

1. The AQ Driver throws an exception when a message is enqueued to a multi-consumer queue and there are no consumers registered for that queue. This means, the Oracle Message Broker throws an exception when messages are published to a topic that uses an AQ queue and no subscriptions have been created for that topic.
2. The AQ Driver only supports PERSISTENT JMS delivery mode *[JMS 4.7]*.
3. The AQ Driver, in OCI Mode, only supports JMS text messages when the `aq_adt` attribute for the queue or topic is set to the value queriable.
4. JMSProperties are not supported when the `aq_adt` attribute is set to `queriable`.
5. AQ Driver Transaction Limitation:

   Using a transacted JMS session, if dequeues are performed on a given topic using a given subscriber and an unsubscribe operation is then attempted before the session is committed, or rolled back, then the unsubscribe blocks until the dequeues are committed or rolled back.

   It is recommended that the dequeues that are performed using a given subscriber be committed, or rolled back, before unsubscribing that subscriber.

   Unsubscribe or subscribe should not be attempted when there are uncommitted sends or receives to the topic.
6. The AQ Driver only supports correlation IDs with less than 129 characters. When you call javax.jms.Message.setJMSCorrelation() with a value containing more than 128 characters for the correlation ID, the AQ Driver throws an exception.
7. The AQ Driver allows durable subscribers to topics. The AQ Driver does not support non-durable subscribers for topics.
8. If the AQ time manager process is never started, or if it dies, the following may occur.

   Dequeued messages will not be removed from the topic AQ table, even though the messages have been delivered. This could eventually cause the database to run out of tablespace.

Start the AQ time manager by adding aq_tm_processes=1 to the Database Server initialization file. If the AQ time manager is running, the following command should return a line showing an active process:

```
% ps -ef | grep ora_q
```

9. If an Oracle Message Broker running on a system uses the Oracle8*i* Database Server on another system to support Oracle AQ, the NLS_LANG environment variable should have the same value on both systems, or if the values are different, they should be compatible values.

   If the NLS_LANG environment variable is set on one system, and not set on another system, receives from topic subscribers from multi-consumer AQ queues may fail when there are messages available.

10. The JDBC based AQ Driver does not work when the AQ Driver attribute `thin_jdbc` is set to true. Do not set `thin_jdbc` to true when the attribute `use_jdbc` is also true.

Oracle AQ Lite Driver

The Oracle Message Broker Advanced Queueing (AQ) Lite Driver provides support for storing persistent messages to AQ Lite. AQ Lite is Oracle's Advanced Queuing product based on Oracle 8*i* Lite Release 4.0. The AQ Lite driver should not be confused with the Oracle Message Broker AQ Driver. These two drivers support substantially different Oracle AQ implementations (see "Oracle Advanced Queuing Driver" on page 7-3 for information on the AQ Driver.)

> **Note:** The Oracle Message Broker AQ Lite Driver is only supported on Windows NT and requires JDK 1.2.x.The Oracle AQ Lite Driver and the AQ Driver, either JDBC or OCI, are mutually exclusive. Only one of these drivers can be active in a given OMB Instance at a given time.

The AQ Lite Driver supports installations that do not demand the performance or scalabiity of the Oracle Message Broker's full AQ driver. The AQ Lite Driver is easier to maintain and consumes fewer system resources than the full AQ Driver. The AQ Lite Driver supports JMS in a smaller configuration, intended for small scale, or individual use.

> **Note:** The AQ Driver provides better overall performance, higher concurrency, and offers richer functionality than the AQ Lite Driver. The AQ Lite Driver implements a subset of the AQ Driver's functionality. Refer to "Oracle Advanced Queuing Driver" on page 7-3 for a description of the AQ Driver.

The AQ Lite Driver runs against one and only one instance of an Oracle Lite database and there can be only one AQ Lite Driver per OMB Instance.

> **Note:** An Oracle Message Broker using the AQ Lite Driver must be co-located with the Oracle8*i* Lite database instance assigned to the AQ Lite Driver.

Connection Management

AQ Lite provides administrative and operational access to its AQ queues using connections known as Java Access Class (JAC) connections. The AQ Lite Driver uses JAC connections to communicate with AQ Lite queues as well as the Oracle Lite database. JAC connections created on a system can only connect to Oracle Lite database instances on the same system. Therefore, an OMB instance using the AQ Lite Driver must be co-located with the Oracle Lite database instance assigned to the AQ Lite Driver.

Using the AQ Lite Driver JAC connections are not pooled. A JAC connection is created each time a JMS session is created, and the connection is associated with the JMS Session for the life of the session. A JAC connection is closed when its corresponding JMS Session is terminated.

Transacted JMS Sessions

The AQ Lite Driver supports both transacted and non-transacted JMS sessions. The AQ Lite Driver implicitly commits at the end of each operation on an AQ Lite queue for JAC connections with non transacted JMS Sessions.

AQ Lite Message Persistence

The AQ Lite Driver supports only JMS persistent delivery mode. When the Oracle Message Broker sends a message using the AQ Driver, it sets the header value for `JMSDeliveryMode` to the value `PERSISTENT`.

AQ Lite Message Mapping

AQ Lite only supports message payloads of the type RAW. Thus, the AQ Lite Driver serializes all JMS messages and stores them as a RAW message payload. The JMS Message ID, Correlation ID, Expiration, and Priority header fields have similar analogues in the AQ Lite message header. The AQ Lite Driver maps these JMS header fields to the AQ Lite message header when it stores a message, and maps the JMS message. Similarly, when the AQ Lite Driver maps messages from AQ Lite to JMS, it copies these AQ Lite message header fields to the corresponding JMS message header fields.

AQ Lite Driver Propagation

The AQ Lite Driver supports message propagation using the Oracle Message Broker propagation facilities (see Chapter 8, "Oracle Message Broker Propagation" for more information).

Oracle Volatile Driver

The Oracle Message Broker Volatile Driver supports lightweight and quick delivery of JMS messages using in-memory communication facilities. The Volatile Driver is useful when high throughput of messages is required. For example, an application using current stock price information, where speed is required and persistence is not required. The Volatile Driver is also useful for communicating in any application where the messaging system does not require that messages be logged to stable storage.

Note: The Volatile Driver only supports NON-PERSISTENT JMS delivery mode *[JMS 4.7]*.

The Volatile Driver stores messages using the same process and the same memory as the active JVM running the Oracle Message Broker.Messages sent using the Volatile Driver are only as persistent as the process in which the Oracle Message Broker runs.

Note: There is a restriction for the Oracle Message Broker when it is running in Local Mode. When sending messages using the Volatile Driver, both the message producer using a volatile destination and the message consumer for the destination must reside in the same JVM. Due to this restriction, Remote Mode is recommended for the Volatile Driver.

IBM MQSeries Driver

The IBM MQSeries Driver is based on IBM MQSeries V5.1. The MQSeries Driver supports the features, and has the limitations as described in the section, "Driver Features Summary" on page 7-2.

In addition, the features shown in Table 7–1, the MQSeries Driver supports the following:

- All JMS message types in JMS mode and only the JMS TextMessage and JMS BytesMessage types in Native mode.
- Native mode support for interoperability with MQSeries applications
- Maximum MQSeries message size of 4 MB. The actual size of a JMS Message body that can be sent using the MQSeries Driver depends on which Driver mode is used (native or JMS). For either mode, the maximum size is smaller than 4 MB, although it is not possible to specify the exact maximum message size, due to differences in the size of the header information that is sent, and due to other overhead associated with sending messages.

MQSeries Message Mapping

Clients using the MQSeries Driver need to distinguish between messages passing between JMS clients and native MQSeries applications. This distinction is required due to limitations in MQSeries, and the fact that JMS requires that JMS messages be delivered to JMS clients without changes. Also, an existing native MQSeries application that uses its own message formats needs to be able to exchange messages with a JMS based application; this implies that a JMS implementation must be able to pass native message formats. Due to this distinction, an Oracle Message Broker administrator needs to configure queues that use the MQSeries Driver to indicate whether the queue is used with a native MQSeries application.

In this section, the terms, *native message* and *native queue* are used for messages exchanged through a queue configured for native MQSeries applications, and the

terms, *JMS message* and *JMS queue* refer to messages exchanged through a queue configured for JMS clients.

Native Queues

For native queues, only the JMS TextMessage and JMS BytesMessage body types are supported.

Sending a JMS TextMessage to a native queue translates the message into an MQSeries message with the format `MQFMT_STRING`; the message buffer contains the characters of the Java String in the character set currently in use by the MQSeries queue manager.

The JMS BytesMessage body type can interoperate with a native MQSeries application that has defined its own message formats. MQSeries data conversion exits must be supplied for these message formats on the queue manager accessed by Oracle Message Broker (for more information on data conversion exits, see the *MQSeries Application Programming Guide*).

Table 7–4 shows the native message mapping.

Table 7–4 Native Queues Message Header Mapping

JMS Header	MQSeries Mapping
JMSMessageID	MQSeries messageID field
JMSCorrelationID	MQSeries correlationID field
JMSDeliveryMode	MQSeries persistence field
JMSExpiration	MQSeries expiry field
JMSPriority	MQSeries priority field
JMSType	For JMS `BytesMessage` body type, the `JMSType` header field is mapped to the MQSeries format field. MQSeries format field set to `MQFMT_STRING` for JMS `TextMessage` messages.

Table 7–4 (Cont.) Native Queues Message Header Mapping

JMS Header	MQSeries Mapping
JMSReplyTo	For native messages, the `JMSReplyTo` field must refer to another MQSeries queue. If the destination specified in the `JMSReplyTo` field is configured for this broker: ■ the `provider_queue_name` for that destination is used to set the MQSeries `replyToQueueName` field ■ the `replyToQueueManagerName` is not set and will be set by the MQSeries queue manager. If the `JMSReplyTo` destination is not configured for this broker: ■ both the `replyToQueueName` and the `replyToQueueManagerName` fields are set from the destination ■ destinations not configured for the broker are created either via `createQueue`() or upon receipt of a native message which has the `replyToQueueName` and `replyToQueueManagerName` fields set ■ ReplyTo queues that are not defined locally require the use of the `DefXmitQName` feature of MQSeries.

JMS Queues

For JMS queues, messages are serialized and sent as raw bytes and then deserialized upon receipt.

The MQSeries Driver maps a JMS message header to fields in the MQSeries message descriptor. Table 7–5 shows the mapping between JMS header fields and MQSeries header fields.

Table 7–5 JMS to MQSeries Message Header Mapping

JMS Header	MQSeries Mapping
JMSMessageID	MQSeries messageID field
JMSCorrelationID	not mapped
JMSDeliveryMode	MQSeries persistence field
JMSExpiration	MQSeries expiry field
JMSPriority	MQSeries priority field
JMSType	MQSeries format field set to MQFMT_NONE
JMSReplyTo	not mapped

Connections to MQSeries Queue Managers

Each JMS session has its own connection to a MQSeries queue manager. The connection is established when the session is created and closed when the session is closed.

Transaction Support

JMS transacted sessions are supported using the native MQSeries transaction support.

Multiple Queue Manager Support

Currently, the MQSeries Driver supports connections to only one MQSeries queue manager.

MQSeries Driver Configuration

Configuring the Oracle Message Broker with the MQSeries Driver involves performing both MQSeries queue administration using the native MQSeries administrative tools and Oracle Message Broker administration using the Oracle Message Broker administrative tools.

Note: Setup and configuration of MQSeries queues is solely the MQSeries administrator's task. Oracle Message Broker does not provide utilities for MQSeries administration.

Administrators use the MQSeries administrative tools to perform all MQSeries administrative tasks, such as creating queues in an MQSeries queue manager. For details on using the MQSeries administration tools, refer to:

`http://www.software.ibm.com/ts/mqseries/library/manuals/`

The administrator also has to use the Oracle Message Broker administrative tools to configure the Oracle Message Broker to use MQSeries. See "Oracle Message Broker Configuration" on page 4-12 for more information.

Message Persistence

The MQSeries Driver supports both the JMS non persistent delivery mode and persistent delivery mode. When the Oracle Message Broker sends a message using

the MQSeries Driver, the default value for the JMSDeliveryMode header is set to PERSISTENT.

MQSeries Driver Limitations

1. The MQSeries driver supports only the JMS point-to-point (PTP) model.
2. The Oracle Message Broker MQSeries Driver only supports connections to one MQSeries Queue Manager.
3. The Oracle Message Broker must be executing on the same system as the MQSeries queue manager.
4. The maximum MQSeries message size is 4 MB. For JMS mode, this implies that the entire JMS message in its serialized form must be less than 4 MB. In native mode, it is just the message body that is limited to 4 MB.
5. Use the native MQSeries administrative tools for all MQSeries administration. Use the Oracle Message Broker administrative utilities to define queues in the directory that are accessed using the MQSeries Driver.
6. The MQSeries Driver does not support the programmatic creation and deletion of queues.

Oracle Multicast Driver

The Oracle Message Broker Multicast Driver supports lightweight and quick delivery of JMS messages using multicast communication facilities. The Multicast Driver is based on the Oracle Application Server Multicast Communication libraries.

The Oracle Application Server Multicast communication protocol gathers processes into logical groups, and issues messages to these groups using an efficient, reliable communication protocol. The Oracle Multicast Protocol uses hardware multicast facilities and is scalable to any number of recipients within a Local Area Network (LAN). The Multicast Driver supports delivery of JMS messages using the Oracle Multicast Protocol.

You can set up a JMS Publish/Subscribe domain using Oracle Message Broker by distributing providers across participating sites within the domain. In this set up, an individual provider does not represent a performance bottleneck or a single point of failure for the domain. On a domain using the Oracle Message Broker and the Multicast Driver, when an individual provider is unavailable, all the remaining providers are able to publish, and subscribe to topics.

Understanding Multicast Driver Operation

Before using Oracle Message Broker with the Oracle Multicast Driver, it is important to understand the driver's functional behavior. Improper usage can result in unexpected system behavior and poor performance.

Oracle Message Broker's multicast facilities define the notion of **group**. A group is a virtual entity that gathers processes together. A group is defined by an IP multicast address and a port number, and contains:

- All Oracle Message Broker instances in a LAN that use the Oracle Multicast Driver with the same IP multicast address and port number.
- All the instances of the `mcastsrv` program in a LAN that also use the same IP multicast address and port number. Refer to "Starting and Stopping the Multicast Bootstrap Server" on page 7-25 for more information on `mcastsrv`.

The group membership is defined as the set of processes that are part of the group. The group membership changes dynamically, with processes being started, terminated, or failing. There are two important rules:

- At least one instance of the `mcastsrv` program must be running prior to starting any Oracle Message Broker instance that uses the Multicast Driver.
- At any point in time, when the group membership changes, the new membership of the group must contain a majority of processes of the previous membership.

The second rule means that only a strict minority of processes can be terminated or fail at the same time. At the same time means *before a new membership is recorded*, which is typically in the order of 1 second for a failure. This implies that, when there are only two processes in the group, a `mcastsrv` program and a single Oracle Message Broker, the failure of one of these prevents the other from functioning properly. To protect against this situation, you should start more than one instance of the `mcastsrv` program.

Distributed Topics

The Multicast Driver maps JMS topics to Oracle Multicast communication groups. These groups support distributed topics and permit simultaneous message communication to many destinations; the JMS client publishing a message does not know the identity, location, or number of these destinations. When the Oracle Message Broker sends messages using the Multicast Driver, communication of the messages occurs without relying on a single, centralized component.

The Multicast Driver manages the distribution of information across remote Oracle Message Brokers. The Multicast Driver does not store messages in a centralized location, each message is distributed to subscribers as it is published. Multiple instances of the Oracle Message Broker using the Multicast Driver are distributed on all participating sites. The multiple instances of the Multicast Driver communicate to manage the production and consumption of messages in an efficient, and reliable manner.

Messages

The Multicast Driver transmits JMS messages and does not interpret them. When a JMS client consumes a message using the Multicast Driver, the message is expected to have been produced by another JMS client using the Multicast Driver.

Message Persistence

The Multicast Driver only supports the non persistent delivery mode. When the Oracle Message Broker sends a message using the Multicast Driver, the `JMSDeliveryMode` header field is expected to have the value `NON_PERSISTENT`.

Multicast Server Configuration

This section covers information an administrator needs to know to configure the Oracle Message Broker's Multicast Driver environment.

Setting Multicast Server Options

To use the Multicast Driver, an administrator needs to configure the IP multicast address and port number pair. Set these values using the Oracle Message Broker administrative tools. If multiple Oracle Message Brokers using the Multicast Driver are set up as a single domain, they need to use the same IP multicast address and port number pair. When an Oracle Message Broker starts, it prints the IP multicast address and port number in the Oracle Message Broker log file.

Warning: Results are unpredictable if an application other than another Oracle Message Broker uses the same IP multicast address and port number pair.

Starting and Stopping the Multicast Bootstrap Server

To initialize a domain with the Oracle Message Broker using the Multicast Driver, an Oracle Multicast bootstrap server, the `mcastsrv` program, needs to be running. The bootstrap server is only required while there are fewer than two Oracle Message Brokers using the Multicast Driver. After two Oracle Message Brokers are running, the bootstrap server can fail without affecting the ability of additional brokers to start up, or the distribution of messages in the domain. This section shows the command to start the bootstrap server.

A network of Oracle Message Brokers using the Multicast Driver requires one instance of a bootstrap server for initialization. The bootstrap server needs to use the same IP multicast address and port number pair as the group of Oracle Message Brokers using the Multicast Driver.

Start the Oracle Multicast bootstrap server using the command:

```
% mcastsrv ip-address port-number [network]
```

Where *`ip-address`* and *`port-number`* specify the address and port used for multicast communication. The *`network`* parameter can be used to specify a IP address if the system has more than one network card (more than one IP address).

When the bootstrap server starts, it prints its IP multicast address and port number to the console.

The `mcastsrv` program shuts down automatically once all the Oracle Message Broker instances that use the Oracle Multicast Driver with the same IP address and port number as the `mcastsrv` program, have been stopped.

MultiCast Driver Performance

The Oracle Multicast Driver implements only minimal flow control; it is the application's responsibility to avoid saturating the network with messages. If an application sends messages as fast as possible, the Oracle Message Broker may fail because it runs out of memory (the Oracle Message Broker keeps a copy of the messages it sends until it is sure that they have been received by all consumers).

When an application tries to send too many messages using the Oracle Multicast Driver, the underlying communication mechanism may be unable to free its buffers.

A throughput of up to 300 messages per second is a possible maximum, depending on the characteristics of the network and depending on the systems running the Oracle Multicast Drivers. This throughout is calculated as the sum of the individual throughputs of all publishers using a single OMB instance. However, there is no

specific maximum, the throughput limit depends both on the speed of the consumers and the load on the network.

MultiCast Driver Limitations

1. The Multicast Driver supports only the JMS publish/subscribe model.
2. The Multicast Driver only supports NON-PERSISTENT JMS delivery mode *[JMS 4.7]*.
3. The Multicast Driver does not support transacted sessions *[JMS 4.4.7]*.
4. The maximum length for a topic name is 120 characters. Using the Oracle Message Broker administrative utilities, limit topic names to less than 120 characters.
5. If a JMS Client performs a very large number of publishes with no delay between each publish, or is running on a highly loaded system, errors may result. The workaround for this is to delay between each publish. As an indication of this problem, the following JMSException is written to the omblog file: `driver state is invalid.`
6. Using the Multicast Driver on a highly loaded system, it is possible that the driver will be deactivated. In this case, any subsequent invocation of the Multicast Driver driver generates an invalid driver exception. All other drivers in the Oracle Message Broker process are fully functional. In this case, to use the Multicast Driver again, the Oracle Message Broker must be restarted. When running in local mode, the JMS client that instantiated the Oracle Message Broker needs to be restarted.

TIB/Rendezvous Driver

The TIB® / Rendezvous(tm) (TIBCO) Driver is based on TIB/Rendezvous Release 5.x or TIB/Rendezvous Pro Release 5.x (when using the JDK 1.2 version of Oracle Message Broker, TIB/Rendezvous Pro Release 5.x is required). The TIBCO Driver provides lightweight and quick delivery of transient messages based on multicast communication facilities.

TIB/Rendezvous is a Message Oriented Middleware (MOM) product based on the Publish/Subscribe model. Applications on heterogeneous platforms communicate by exchanging data on specific subjects. An application can listen to one or more subjects using TIBCO's subject-based addressing technology. TIB/Rendezvous provides lightweight reliable broadcast communication facilities that scale across

multiple LANs. The TIBCO Driver allows the Oracle Message Broker to distribute JMS messages using TIB/Rendezvous.

An administrator can set up a JMS Publish/Subscribe domain using the Oracle Message Broker (provider) and the TIBCO Driver by distributing providers across participating sites within the domain. In this set up, an individual provider does not represent a performance bottleneck or a single point of failure for the domain. On a domain using the Oracle Message Broker and the TIBCO Driver, when an individual provider is unavailable, all the remaining providers are able to publish, and subscribe to topics.

Distributed Topics

The TIBCO Driver maps JMS topics to TIB/Rendezvous subjects. TIB/Rendezvous subjects support distributed topics and permit simultaneous communication of messages to several destinations; the JMS client publishing a message does not know the identity, location, or number of these destinations. Communication occurs without relying on a single, centralized component.

Messages

By default, the TIBCO Driver transmits JMS messages and does not interpret them. Messages received by a JMS client using the TIBCO Driver are expected to be produced by another JMS client using the TIBCO Driver. Optionally, the TIBCO Driver supports message mapping for converting JMS format messages to native TIB/Rendezvous format (refer to "TIB/Rendezvous Message Mapping" on page 7-27).

Message Persistence

The TIBCO Driver only supports the JMS non persistent delivery mode. When the Oracle Message Broker sends a message using the TIBCO Driver, the `JMSDeliveryMode` header field is expected to have the value `NON_PERSISTENT`.

TIB/Rendezvous Message Mapping

The Oracle Message Broker Administrator supports a configuration where the TIBCO Driver converts JMS messages to native TIB/Rendezvous format. When the TIBCO Driver sends messages in native format, it maps JMS messages to structured self-describing TIB/Rendezvous messages. The TIBCO Driver performs this conversion for all JMS message types, including: TextMessage, BytesMessage, MapMessage, StreamMessage, and ObjectMessage. The native mapping option

affects only topic publishers; topic subscribers can read both standard JMS and TIBCO Driver generated messages.

TIB/Rendezvous messages have a self-describing format composed of name/value pairs. A value is a basic type or another message. All the basic types valid in JMS messages have a corresponding type in the generated TIB/Rendezvous message, except the Java `char` type that is converted into a `string` containing a single character. Since native TIB/Rendezvous applications do not use this type, this limitation applies only when using the TIBCO Driver with the native mapping enabled to communicate messages between two JMS applications.

With the native mapping option enabled, the TIBCO Driver translates JMS header fields into a name/value pair with the name mapped to the header field name. Table 7–6 shows the native mapping for the JMS message header.

Table 7–6 TIBCO Driver Native Header Mapping

JMS Message Header	TIB/Rendezvous Native Message
JMSDestination: `Destination`	("JMSDestination", `String`)
JMSDeliveryMode: `int`	("JMSDeliveryMode", `int`)
JMSExpiration: `long`	("JMSException", `long`)
JMSPriority: `int`	("JMSPriority", `int`)
JMSMessageID: `String`	("JMSMessageID", `String`)
JMSTimestamp: `long`	("JMSTimestamp", `long`)
JMSCorrelationID: `String`	("JMSCorrelationID", `String`)
JMSReplyTo: `Destination`	("JMSReplyTo", `String`)
JMSType: `String`	("JMSType", `String`)
JMSRedelivered: `boolean`	("JMSRedelivered", `boolean`)

The TIBCO Driver translates JMS properties into an equivalent TIB/Rendezvous name/value pair stored in a nested message named `Properties`. Table 7–7 shows the native mapping for JMS properties.

Table 7–7 TIBCO Driver Native Properties Mapping

JMS Message Properties	TIB/Rendezvous Native Message
	("Properties", RvMsg)
("Name1", type1)	("Name1", type1)
("Name2", type2)	("Name2", type2)
("Name*n*", type*n*)	("Name*n*", type*n*)

The body of a JMS message is stored in a name / value pair named Body. The message has different values depending on the message type. A JMS message body is translated to include the message content, and an additional field, named BodyType that describes the type of the data in the body (Text, Bytes, Map, Stream, or Object). Table 7–8 shows the different mappings for the message body.

Table 7–8 JMS Message Body Translation for Native TIB/Rendezvous

JMS Message Type	Translation
TextMessage	String
BytesMessage	An opaque array of bytes
ObjectMessage	An opaque array of bytes
MapMessage	A nested TIB / Rendezvous message
StreamMessage	A nested TIB / Rendezvous message

Table 7–9 shows the translation values for a TextMessage. Table 7–10 shows the translation values for a BytesMessage. Table 7–11 shows the translation values for a MapMessage. Table 7–12 shows the translation values for a StreamMessage. Table 7–13 shows the translation values for the ObjectMessage type.

Table 7–9 TextMessage TIBCO Driver Native Body Mapping

JMS Message Body	TIB/Rendezvous Native Message
	("BodyType", "Text")
String	("Body", String)

Table 7–10 BytesMessage TIBCO Driver Native Body Mapping

JMS Message Body	TIB/Rendezvous Native Message
	("BodyType", "Bytes")
byte[]	("Body", RvOpaque)

Table 7–11 MapMessage TIBCO Driver Native Body Mapping

JMS Message Body	TIB/Rendezvous Native Message
	("BodyType", "Map")
	("Body", RvMsg)
("Name1", type1)	("Name1", type1)
("Name2", type2)	("Name2", type2)
("Name*n*", type*n*)	("Name*n*", type*n*)

Table 7–12 StreamMessage TIBCO Driver Native Body Mapping

JMS Message Body	TIB/Rendezvous Native Message
	("BodyType", "Stream")
	("Body", RvMsg)
type1	("1", type1)
type2	("2", type2)
type*n*	("*n*", type*n*)

Table 7–13 ObjectMessage TIBCO Driver Native Body Mapping

JMS Message Body	TIB/Rendezvous Native Message
	("BodyType","Object")
Object	("Body", RvOpaque)

Sessions

The number of TIB/Rendezvous sessions is limited by the maxPrivateConnections attribute of the ServerEntry. A TIB/Rendezvous session is created for each JMS session that has at least one producer or one consumer, and for each durable subscriber.

TIB/Rendezvous Installation and Administration

To use the Oracle Message Broker with the TIBCO Driver an administrator needs to install and configure TIB/Rendezvous Release 5.x or TIB/Rendezvous Pro Release 5.x. For information on TIB/Rendezvous installation and administration refer to the *TIB/Rendezvous Administrator's Guide.* For more information on TIB/Rendezvous see the following web site, http://www.rv.tibco.com/

Note: Setup and configuration of TIB/Rendezvous is solely the TIB/Rendezvous administrators task. Oracle Message Broker does not provide utilities for TIB/Rendezvous administration.

TIB/Rendezvous Driver Limitations

1. The TIB® /Rendezvous™ (TIBCO) Driver supports only the JMS publish/subscribe model.
2. The TIBCO Driver only supports NON-PERSISTENT JMS delivery mode *[JMS 4.7]*.
3. The TIBCO Driver does not support transacted sessions *[JMS 4.4.7]*.
4. The TIBCO Driver uses the `max_private_sessions` attribute to limit the number of sessions connected to the Rendezvous daemon. A session is created when a JMS session is created, and when a new durable subscriber is created. Therefore, creating a durable subscriber may cause an exception to be thrown because the Oracle Message Broker creates a Rendezvous session for the subscriber, even though it already maintains a valid JMS session. If an exception is thrown in this case, increase the value for the `max_private_sessions` attribute to increase the limit on the number of sessions connected to the Rendezvous daemon.
5. Using the JDK 1.2 version of Oracle Message Broker, TIB/Rendezvous Pro Release 5.x is required.

8

Oracle Message Broker Propagation

This chapter describes the Oracle Message Broker propagation manager. The Oracle Message Broker uses a propagation manager to transfer messages between JMS destinations, either within a single Oracle Message Broker, or on different Oracle Message Brokers running in a distributed environment. The propagation manager supports propagation between Oracle Message Brokers using either the IIOP protocol or the HTTP protocol.

This chapter covers the following:

Overview of Oracle Message Broker Propagation

In order to transfer messages from one JMS destination to another JMS destination, an entry for a propagation job needs to be created in the Oracle Message Broker administrative LDAP Directory. A **propagation job** defines a propagation source, a propagation target, and a propagation transport protocol. The Oracle Message Broker that dequeues messages from the source destination is the **sending broker**, while the Oracle Message Broker that enqueues messages into the target destination is the **receiving broker**. The sending broker and receiving broker for a propagation job can be the same broker if the source destination and the target destination of the propagation job are managed by the same Oracle Message Broker.

When a propagation job is activated, the sending broker works with the receiving broker to move messages from the source destination to the target destination. If both the source destination and the target destination are persistent, messages are guaranteed to be delivered from the source to the target reliably (in order, with no loss, and no duplicates). If either the source or the target of a propagation job is non-persistent, the sending broker and receiving broker make every effort to deliver messages reliably. However, non-persistent messages may be lost in the case of failures of either the sending or the receiving broker, or any of the underlying messaging systems.

Note: The propagation feature is only available when running the Oracle Message Broker in Remote Mode.

This section covers the following:

- Types of Propagation
- Propagation with Message Selectors

Types of Propagation

Oracle Message Broker message propagation can be used for the following purposes:

- Message Distribution – to transfer messages across sites.
- Message Server Conversion – to convert messages between message servers (different message servers use different Oracle Message Broker drivers).
- Domain Conversion – to convert messages between different JMS message domains (queue to topic or topic to queue).

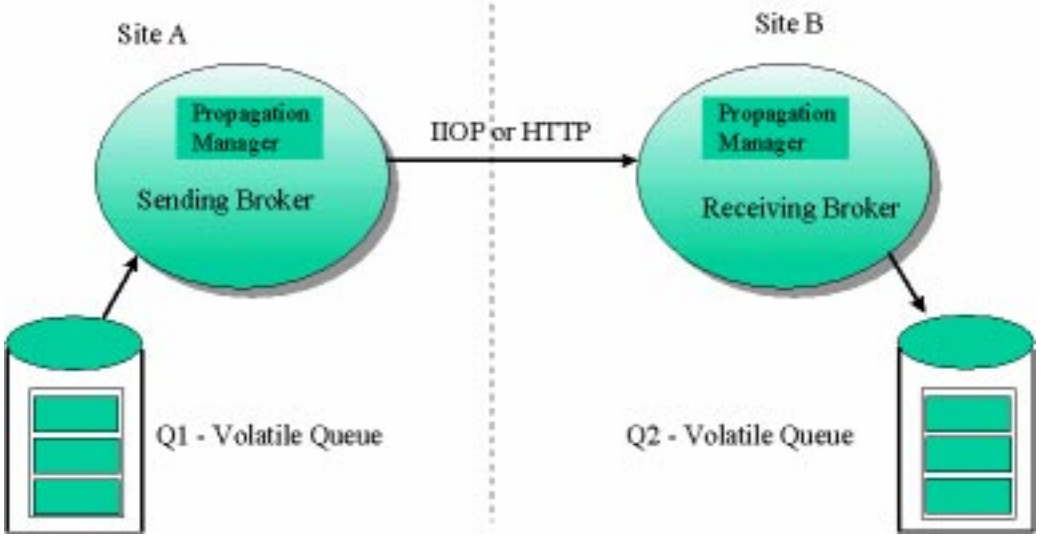
To perform one of these types of propagation, the Oracle Message Broker administrator configures a propagation job. The characteristics of the source and the target destinations determines the type of propagation that is performed, either: message distribution, message server conversion, or domain conversion.

Message Distribution

Using message distribution propagation, the Oracle Message Broker uses its propagation feature to transfer messages between destinations. To use this type of propagation, the administrator configures a propagation job and specifies the source queue (or topic) and the target queue (or topic).

For example, Figure 8–1 shows message distribution for Volatile Queue Q1 on site A, to Volatile Queue Q2 on site B.

Figure 8–1 Sample Message Distribution with Propagation



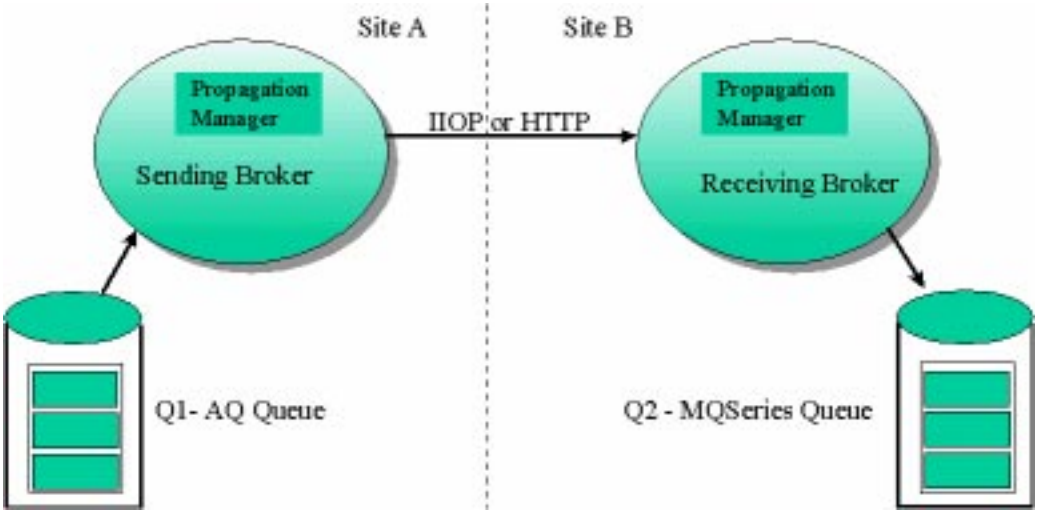
Message Server Conversion

Using message server conversion, the Oracle Message Broker transfers messages between different supported message servers. The Oracle Message Broker supports message servers using the available Oracle Message Broker drivers (for more information see Chapter 7, "Message Servers and Drivers").

The propagation manager supports message server conversion between the following drivers: Oracle Advanced Queuing (AQ) Driver, IBM MQSeries Driver, Oracle Volatile Driver, Oracle Multicast Driver, and the TIBCO/Rendezvous Driver.

Figure 8–2 shows message server conversion from AQ Queue Q1 on Site A to MQ Series Queue Q2 on Site B.

Figure 8–2 Sample Propagation Message Server Conversion

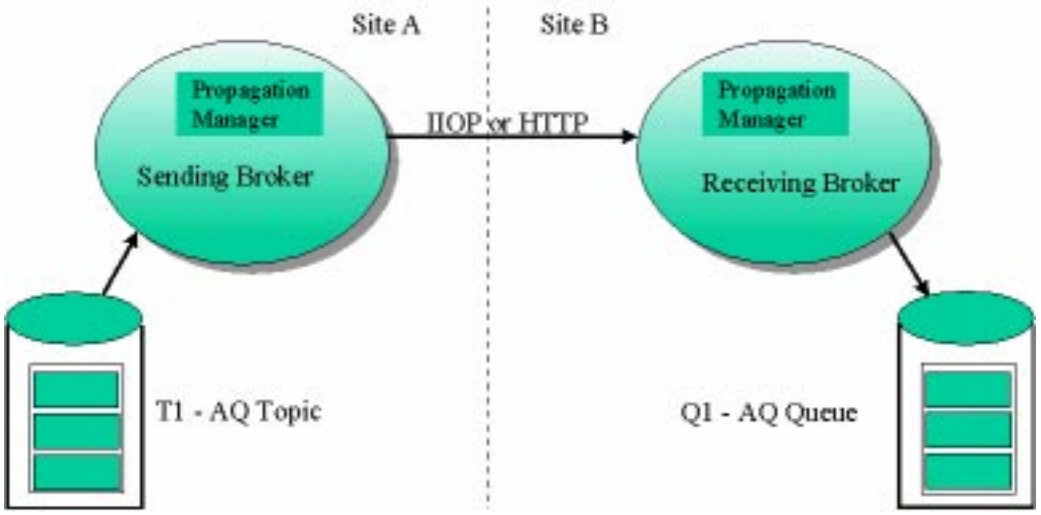


Domain Conversion

The domain conversion feature uses the Oracle Message Broker to transfer messages between supported JMS messaging domains. Domain conversion allows the Oracle Message Broker to send messages from a JMS queue to a JMS topic, or from a JMS topic to a JMS queue (in either direction).

For example, Figure 8–3 shows message propagation from AQ topic T1 to AQ queue Q1.

Figure 8–3 Sample Propagation Domain Conversion



Propagation with Message Selectors

Some applications in a distributed environment require messages to be distributed among messaging systems based on message properties. Oracle Message Broker propagation supports this feature by allowing the administrator to associate a message selector with a propagation job. Using this feature, only the messages that satisfy the message selector are propagated from the source to the target.

Note: Message selectors can be specified only for a propagation job whose source is a topic. Message selectors are not supported when the propagation source is a queue.

A message selector for a propagation job must be defined as a valid JMS message selector (for more information on the valid message selector format, see "Message Selector Format" on page 5-4). Refer to "Propagation Job Configuration" on page 8-16 for more information on specifying propagation job message selectors.

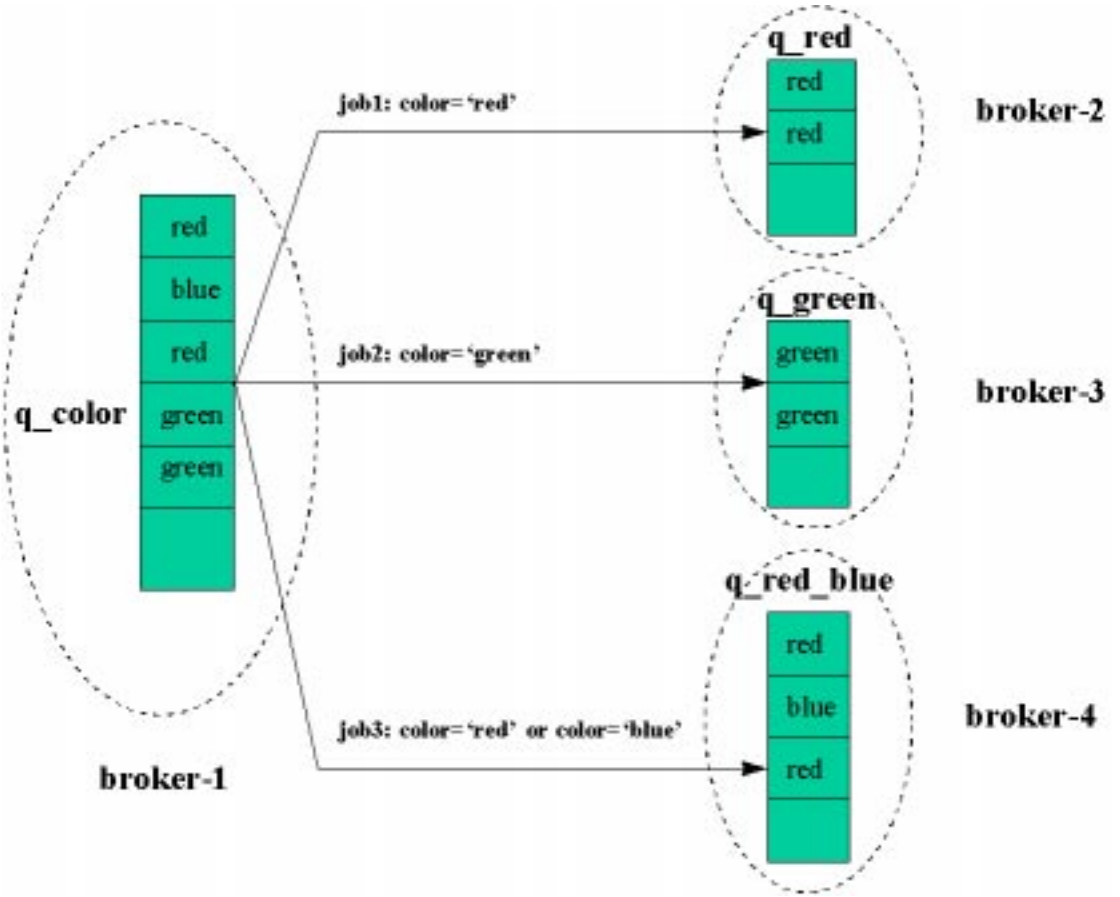
Figure 8–4 shows an example of this type of propagation, where the messages in the queue, q_color on broker-1, are distributed to many destinations on other Oracle Message Brokers (queue q_red on broker-2, queue q_green on broker-3 and queue q_red_blue on broker-4). Messages are selected for distribution based on message

properties using selectors. For example, the colors red, blue, and green, stored in a JMS properties could be selected for propagation to the specified color queues on the target Oracle Message Brokers.

By accessing the queue q_red, an application on broker-2 receives all and only the "red" messages that were propagated from the queue q-color on broker 1.

To distribute messages as shown in Figure 8–4, the source q_color must be a topic, and three propagation jobs need to have the color message selectors defined, as shown in the figure.

Figure 8–4 One-to-many Propagation using Message Selectors



Propagation Transport Protocols

The Oracle Message Broker propagation manager allows propagation between queues and topics using two transport protocols:

- IIOP Propagation – messages are propagated between the sending broker and the receiving broker using the IIOP protocol.
- HTTP Propagation – messages are propagated between the sending broker and the receiving broker using the HTTP protocol.

IIOP Propagation

Using IIOP propagation, messages are propagated between the sending broker and the receiving broker using the CORBA IIOP protocol. This is the default transport protocol.

Local Propagation

When the propagation manager finds a single broker is used as both the sending broker and the receiving broker, the propagation manager uses local procedure calls rather than IIOP.

HTTP Propagation

The Oracle Message Broker HTTP propagation manager supports two types of HTTP propagation:

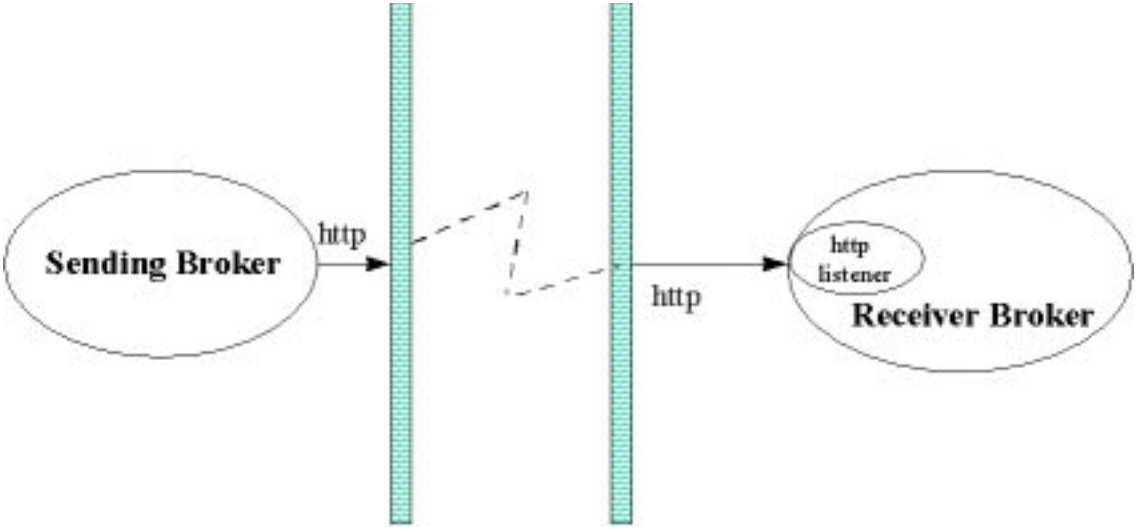
- Direct HTTP Propagation
- Servlet Based Propagation

This section describes the two HTTP propagation types.

Direct HTTP Propagation

Oracle Message Broker can be configured to use Oracle Message Broker's built-in HTTP listeners. If a sending broker can directly access the HTTP built-in listeners on a receiving broker and create HTTP connections, direct HTTP propagation can be setup between the sending broker and the receiving broker (see Figure 8–5). Compared with servlet based propagation, direct HTTP propagation is easier to setup and usually has better performance characteristics.

Figure 8–5 HTTP Direct Propagation



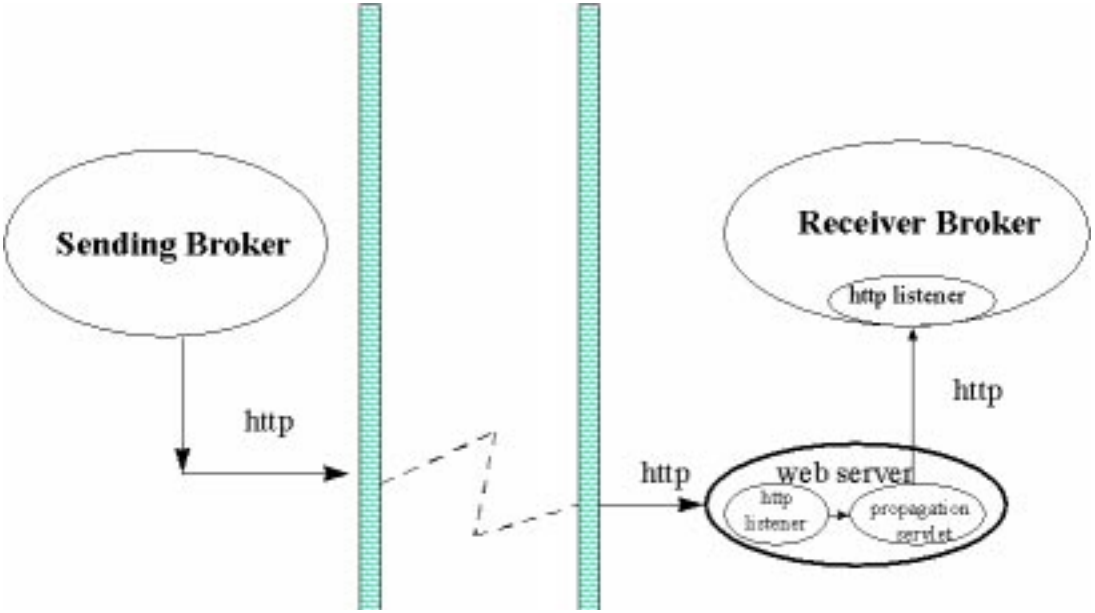
Servlet Based Propagation

A built-in HTTP listener running in the receiving broker behind a firewall may not be directly accessible to a sending broker outside the firewall. Some organizations require, for security reasons, that all HTTP traffic pass through centralized web servers. In this case, an Oracle Message Broker HTTP propagation servlet needs to be deployed on the web server to forward messages from the web server to the receiving broker (See Figure 8–6). Using servlet based propagation, Oracle Message Broker propagation jobs that send messages to the receiving broker are configured as though the messages are sent to the servlet. The HTTP propagation servlet then transfers the messages to the receiving broker.

An HTTP propagation servlet communicates with one and only one Oracle Message Broker built-in HTTP listener of a receiving broker. If there are more than one receiving brokers behind the same firewall, multiple propagation servlets must be configured (each using a different virtual path).

The configuration parameters of the HTTP propagation servlet include a host name, the HTTP listener port number, SSL level, and wallet information for the associated receiving broker (for more information, see "HTTP Propagation Servlet Configuration" on page 8-21.)

Figure 8–6 HTTP Propagation Using a Servlet



Administration and Configuration

Propagation configuration tasks differ depending on whether the administrator is configuring the sending broker or the receiving broker (a broker may be both a sending broker and a receiving broker). To set up and configure the Oracle Message Broker for propagation and to create propagation jobs, the Oracle Message Broker administrator needs to perform one or more of the following tasks:

- Sending Broker Configuration
- Receiving Broker Configuration
- Propagation Job Configuration
- HTTP Propagation Servlet Configuration

Note: Some details of propagation configuration are different for the HTTP and IIOP protocols. This section covers details specific to both of the supported protocols: IIOP or HTTP.

Sending Broker Configuration

Configuring and setting up a sending broker for propagation includes the following tasks:

- Configuring the Message Broker Entry (for the Sender)
- Configuring Persistent Message Servers and Drivers
- Configuring Remote Directories (for IIOP Propagation Only)
- Configuring Remote HTTP Listeners (for HTTP Propagation Only)

Configuring the Message Broker Entry (for the Sender)

To configure the `msg_broker` entry for propagation on the sending broker, use the Oracle Message Broker administrative utilities. Table 4–8 shows the `msg_broker` entry attributes, including the sending broker related attributes:

- `propagation_send_threads`
- `propagation_http_handlers`

> **Note:** The administrator only needs to perform this step once, no matter how many propagation jobs are created.

Setting Propagation Send Threads By default, the `propagation_send_threads` attribute is set to one (1). The administrator can change the default value to tune the Oracle Message Broker to handle propagation more efficiently, based on the available resources and on the desired number of concurrent active propagation jobs.

Setting Propagation HTTP Handlers (for HTTP Propagation only) When an Oracle Message Broker participates in HTTP propagation as a sending broker, the administrator needs to configure the `propagation_http_handlers` attribute (see Table 4–8). This specifies the number of HTTP propagation handler threads that are used to process acknowledgments and flow control requests from receiving brokers.

Do not use the administration utilities to modify the `propagation_http_handlers` attribute when the Oracle Message Broker is active (this is a `msg_broker` entry attribute). If attempted, the updated value is added to the directory, but the existing, old value is used until the Oracle Message Broker is restarted.

Configuring Persistent Message Servers and Drivers

If a broker is expected to process propagation jobs that use persistent destinations as source, the associated persistent message server needs to be configured to support propagation. The message servers that support persistent messages are:

- Oracle AQ Driver
- IBM MQSeries Driver

> **Note:** If the propagation job involves a non-persistent queue or topic, the message server entry does not need to be modified to support propagation. This includes the Oracle Volatile Driver, the Oracle Multicast Driver, and the TIBCO Driver.

Using persistent queues or topics with propagation, the administrator needs to perform the following actions to setup the message server and the driver:

1. Create Propagation Logging Queue For Sending
2. Configure Message Server Logging Queue Attribute
3. Configure Driver Propagation Send Sessions Attribute

> **Note:** The administrator only needs to perform the steps in this section for propagation jobs with a destination using the Oracle AQ Driver or the IBM MQSeries Driver.

Create Propagation Logging Queue For Sending The propagation manager uses the propagation logging queue to log the state of propagation jobs. The propagation manager uses logging information for propagation job recovery in case of failure. This propagation configuration step requires the following actions:

a. For MQSeries or Oracle AQ, create the sending log queue entry. The sending log queue entry is created like any other queue using the Oracle Message Broker administration utilities. For AQ queues, this step should also create the native AQ queue.

b. For MQSeries, use the native MQSeries administration tools to create the native queue.

The sending log queue and the receiving log queue can be the same. Using separate queues for the logging queues should provide better performance.

For propagation with an AQ source, the associated sending log queue must be configured using the following specification for the aq_adt attribute.

- OCI Mode: When the AQ Driver is configured in OCI Mode, the logging queue must set the `aq_adt` to the type raw. See Table 4–17 and Table 4–18 for information on the `aq_adt` attribute.
- JDBC Mode: When the AQ Driver is configured in JDBC Mode, the logging queue must set the `aq_adt` to the type stream. See Table 4–17 and Table 4–18 for information on the `aq_adt` attribute.

Configure Message Server Logging Queue Attribute After creating the required sending log queue, set the `prop_send_log_queue` in the message server entry to the DN of the sending log queue.

> **Note:** You only need to set both of the `prop_send_log_queue` and the `prop_recv_log_queue` attributes only if the server is used for both the sending and receiving.

Configure Driver Propagation Send Sessions Attribute For the driver associated with the propagation source destination, set the driver's `propagation_send_sessions` attribute to the desired number of sessions on the driver. This value determines the degree of concurrency for the propagation sending process. When setting this value, note the following restriction:

```
propagation_recv_sessions + propagation_send_sessions < max_private_sessions
```

Configuring Remote Directories (for IIOP Propagation Only)

A Remote Directories entry represents a foreign, or remote, LDAP Server. A remote directory entry provides information about how to access a remote LDAP Server to resolve a propagation target DN in the propagation job entry.

A remote directories entry only needs to be specified when an Oracle Message Broker is involved in IIOP propagation as a sender. See Table 4–19 for a description of the Remote Directories entry attributes.

Configuring Remote HTTP Listeners (for HTTP Propagation Only)

A Remote HTTP Listener entry represents a foreign, or remote, Oracle Message Broker built-in HTTP listener, or a propagation servlet.

A Remote HTTP Listener entry only needs to be specified when an Oracle Message Broker is involved in HTTP propagation as a sender. See Table 4–20 for a description of the remote HTTP listener entry attributes.

Note: The attributes remote_wallet_location and remote_wallet_password specify a “local” wallet information on the sending site

---

Receiving Broker Configuration

Configuring and setting up a receiving broker for propagation includes the following tasks:

- Configuring the Message Broker Entry (for the Receiver)
- Configuring Persistent Message Servers and Drivers
- Configuring the HTTP Listener (for HTTP Propagation Only)

Configuring the Message Broker Entry (for the Receiver)

To configure the `msg_broker` entry for propagation on the receiving broker, use the Oracle Message Broker administrative utilities. Table 4–8 shows the `msg_broker` entry attributes, including the receiving broker related attributes:

- `propagation_recv_threads`
- `propagation_http_handlers`

Note: The administrator only needs to perform this step once, no matter how many propagation jobs are created.

Setting Propagation Receive Threads By default, the `propagation_recv_threads` attribute is set to one (1). The administrator can change the default value to tune the Oracle Message Broker to handle propagation more efficiently, based on the available resources and on the desired number of concurrent active propagation jobs.

Setting Propagation HTTP Handlers (for HTTP Propagation only) When an Oracle Message Broker participates in HTTP propagation as a receiving broker, the administrator needs to configure the `propagation_http_handlers` attribute (see Table 4–8). This specifies the number of HTTP propagation handler threads that are used to process propagation requests from sending brokers.

Do not use the administration utilities to modify the `propagation_http_handlers` attribute when the Oracle Message Broker is active (this is a `msg_broker` entry attribute). If attempted, the updated value is added to the directory, but the existing, old value is used until the Oracle Message Broker is restarted.

Configuring Persistent Message Servers and Drivers

If a broker is expected to process propagation jobs that use persistent destinations as targets, the associated persistent message server needs to be configured to support propagation. The message servers that support persistent messages are:

- Oracle AQ Driver
- IBM MQSeries Driver

> **Note:** If the propagation job involves a non-persistent queue or topic, the message server entry does not need to be modified to support propagation. This includes the Oracle Volatile Driver, the Oracle Multicast Driver, and the TIBCO Driver.

Using persistent queues or topics with propagation, the administrator needs to perform the following actions to setup the message server, and the driver:

1. Create Propagation Logging Queue for Receiving
2. Configure Message Server Logging Queue Attribute
3. Configure Driver Propagation Receive Sessions Attribute

> **Note:** The administrator only needs to perform the steps in this section for propagation jobs with a destination using the Oracle AQ Driver, or the IBM MQSeries Driver.

Create Propagation Logging Queue for Receiving The propagation manager uses the propagation logging queue to log the state of propagation jobs. The propagation manager uses logging information for propagation job recovery in case of failure. This propagation configuration step requires the following actions:

a. For MQSeries or Oracle AQ, create the receiving log queue entry. The receiving log queue entry is created like any other queue using the Oracle Message Broker administration utilities. For AQ queues, this step should also create the native AQ queue.

b. For an MQSeries queue, use the native MQSeries administration tools to create the queue manager.

The sending log queue and the receiving log queue can be the same. Using separate queues for the logging queues should provide better performance.

For propagation with an AQ target, the associated receiving log queue must be configured using the following specification for the `aq_adt` attribute.

- ACI Mode: When the AQ Driver is configured in OCI Mode, the logging queue for the associated source or target must set the `aq_adt` to the type raw. See Table 4–17 and Table 4–18 for information on the `aq_adt` attribute.
- JDBC Mode: When the AQ Driver is configured in JDBC Mode, the logging queue for the associated source or target must set the `aq_adt` to the type stream. See Table 4–17 and Table 4–18 for information on the `aq_adt` attribute.

Configure Message Server Logging Queue Attribute After creating the required receiving log queue, set the `prop_receive_log_queue` in the message server entry to the DN of the receiving log queue.

Note: You only need to set both of the `prop_send_log_queue` and the `prop_recv_log_queue` attributes only if the server is used for both the sending and receiving.

Configure Driver Propagation Receive Sessions Attribute For the driver associated with the propagation target destination, set the driver's `propagation_recv_sessions` attribute to the desired number of sessions on the driver. This value determines the degree of concurrency for the propagation receiving process. When setting this value, keep the following restriction in mind:

```
propagation_recv_sessions + propagation_send_sessions < max_private_sessions + 4
```

Configuring the HTTP Listener (for HTTP Propagation Only)

For a receiving broker participating in HTTP propagation, HTTP listeners must be configured by creating `prop_http` entries. A receiving broker can have multiple HTTP listeners listening on different ports. After creating a `prop_http` entry, set its attributes. Refer to Table 4–15 for details on the attributes.

Note: The HTTP Listener entry only needs to be created and configured on the receiving broker.

Propagation Job Configuration

The propagation job entry specifies configuration information for the propagation job, such as the source, the target, and the transport protocol. The propagation job entry must be created in the same OMB Instance as its source. This section provides further information on the propagation job attributes shown in Table 4–21.

Note: A propagation job entry is only defined on the sending side.

This section describes the following propagation job attributes:

- Propagation Source
- Propagation Target
- Remote DN
- Propagation Message Selector
- Activation State
- Propagation Username
- Propagation Password
- Propagation Timeout
- Create Timestamp
- Valid Status

Propagation Source

The propagation source holds the DN of a queue or topic from which messages are propagated.

Note the following limitations when defining a propagation source:

1. A queue can be the source destination of at most one propagation job, otherwise unrecoverable propagation failures may occur. It is the administrator's responsibility to define only one propagation job per queue as the propagation source.

   If a topic is specified as the source for a propagation job, the topic can be used as the source for other propagation jobs.

2. Applications should not receive messages from propagation source queues. Receiving messages from the propagation source queue may result in unrecoverable propagation failures. Applications can receive messages from propagation source topics.

Propagation Target

The propagation target holds the DN of a queue or topic to which messages are propagated. This DN is a location on an Oracle Message Broker LDAP Directory. It is not necessarily the local LDAP Directory that stores the propagation source. The target location is determined by the combination of the `remote_dn` attribute and the `propagation_target` attribute (see "Remote DN" on page 8-18).

When the propagation target is an AQ queue or an AQ topic, the administrator must make sure the types of all messages in the source destination are compatible with the target destination. Otherwise, propagation processing may fail. For example, consider a propagation job that uses a volatile queue as the propagation source, and an AQ queue that is configured with the `aq_adt` attribute type text for the propagation target. If the source queue contains a message of JMS object type, then the propagation job fails when the object type message is propagated to the target, since the types of messages that the AQ queue specified `aq_adt` attribute set to text is limited to text messages (see Table 8–1 for limitations on propagation using AQ destinations as the source and AQ destinations as the target).

See "Propagation Limitations" on page 8-28 for information on how to avoid creating looped propagation jobs.

When the propagation source and target both use the AQ Driver, they must use compatible types. To configure AQ destination types, set the `aq_adt` attribute for the topic or the queue. Table 8–1 lists compatible types for AQ propagation.

Table 8–1 Propagation Using the AQ Driver

Source ADT	Supported Destination ADT
all	all, raw
bytes	all, bytes, raw
map	all, map, raw
object	all, object, raw
queriable	all, queriable, raw
raw	all, raw
stream	all, stream, raw
text	all, text, raw

Remote DN

The remote DN attribute determines the transport protocol for the propagation job and allows the propagation manager to find the receiving broker and resolves the target destination name. The remote DN attribute value is set to null, a remote directory entry DN, or a remote HTTP entry DN.

The `remote_dn` attribute should contain one of the values shown in Table 8–2.

Table 8–2 Remote DN Values

Value	Description
null	The value of the `remote_dn` attribute is set to null if the propagation target DN shared the LDAP Server as the propagation source DN. If the sending broker and the receiving broker use different OMB instances, then IIOP protocol is used for propagating messages from the source to the target. If the sending broker and the receiving broker are the same broker, then local procedure calls are used to propagate messages from the source to the target.
Remote Directory entry DN	When the `remote_dn` attribute is set to the DN of a Remote Directory entry, the information in the remote directory entry associated with the DN is used to access a remote LDAP Server in order to fetch the propagation target destination entry and the receiving broker entry. This implies using IIOP for the propagation job.
Remote HTTP entry DN	When the `remote_dn` attribute is set to the DN of a Remote HTTP entry, the remote HTTP entry specifies information about how to access the remote HTTP handler. This implies using HTTP for the propagation job.

Propagation Message Selector

The `propagation_msg_selector` attribute sets the message selector for the propagation job.

> **Note:** Message selectors can be specified only for propagation jobs whose source destination is a topic. Message selectors are not supported when the propagation source destination is a queue.

A message selector for a propagation job must be a string that is a valid JMS message selector (for more information on the valid message selector format, see "Message Selector Format" on page 5-4.

The propagation manager processes each propagation job independently. Therefore, a message in a source topic can be propagated to multiple target destinations if its properties satisfy more than one message selector for propagation jobs that use the same topic as propagation source. When messages are only intended to propagate from the source topic to a single target destination, it is the administrator's responsibility to specify exclusive message selectors, among all propagation jobs that use the same topic as source.

Propagation Message Selector Notes

1. Message selectors cannot be updated. To change a message selector, the associated propagation job entry must be deleted and re-created, using the new message selector.
2. Deleting a propagation job with a topic as a source results in removing remaining messages. Thus, changing a message selector by deleting a propagation job entry could result in lost messages.
3. When defining message selectors for propagation jobs using `AdminUtil`, administrators should pay attention to the special characters, such as ", ', and $, so that the selector to be parsed correctly by `AdminUtil` (quotes must precede some special characters). If the message selectors are defined using the GUI interface provided with `ombadmin`, this is not an issue.
4. Since AQ queues that use the `aq_adt` attribute, and set its value to queriable do not support JMS properties, propagation message selectors should not be specified for propagation jobs with `aq_adt` set to queriable (for the propagation source).

Activation State

The `activation_state` attribute specifies whether the propagation job is activated or deactivated. When a propagation job is deactivated, the propagation manager stops sending messages for the propagation job from the sending broker to the receiving broker. See "Activating and Deactivating a Propagation Jobs" on page 8-26 for more information.

Propagation Username

The `propagation_username` attribute is used along with the `propagation_password` attribute for authentication and authorization at the receiving site.

Propagation Password

The `propagation_password` attribute is used along with the `propagation_username` attribute for authentication and authorization at the receiving site.

Propagation Timeout

The propagation manager at the sending site expects an acknowledgment from the receiving site for each propagation request that it sends. This occurs before the sending site commits operations associated with a request. If the acknowledgment does not arrive within the time specified by the propagation timeout attribute, the propagation manager assumes a failure has occurred with the associated propagation job. The propagation manager then stops the propagation job and automatically tries to recover and continue the propagation job after a delay.

When messages in the following format are shown in the Oracle Message Broker log file, this signifies a potential propagation problem:

```
Deactivating propagation job [job id] because its requests have timed-out.
```

The propagation problem could be caused by any of the following:

- Very large messages are being propagated
- The receiving broker or message server is busy, or not running
- Communication lines are congested
- The propagation timeout value is set to a value that is too small

If the message servers, and the Oracle Message Broker, and the network, and all other components are running properly, then the propagation requests may require more time for processing. In this case, the propagation timeout value should be increased.

Create Timestamp

The `create_timestamp` attribute is reserved for internal use only. This attribute stores the creation time of the propagation job entry.

Valid Status

The `valid_status` attribute is used internally by the propagation manager as a flag to indicate if a propagation job entry is valid. An invalid propagation job is caused by an unsuccessful create or delete of a propagation job entry. If a propagation entry is invalid the value of this attribute is set to false. The propagation manager does not process propagation jobs with `valid_status` in the false state.

To avoid accumulating messages for an invalid propagation job, users should remove invalid propagation jobs.

HTTP Propagation Servlet Configuration

The HTTP propagation servlet requires that the Oracle Message Broker be installed on the system where the servlet runs. After the Oracle Message Broker is installed, the propagation servlet file (PropHttpServlet.class) needs be copied from $OMB_HOME/servlets/propagation/PropHttpServlet.class to the location where the servlet is hosted in the web server (on Windows NT systems, %OMB_HOME%\servlets\propagation\PropHttpServlet.class).

When running with JDK 1.2, the propagation HTTP servlet requires the following three jar files in the classpath:

```
$OMB_HOME/classess/mercury.jar
$ORACLE_HOME/jlib/jssl-1_2.jar
$ORACLE_HOME/jlib/javax-ssl-1_2.jar
```

When running with JDK 1.1.8, the propagation HTTP servlet requires the following three jar files in the classpath:

```
$OMB_HOME/classess/mercury.jar
$ORACLE_HOME/jlib/jssl-1_1.jar
$ORACLE_HOME/jlib/javax-ssl-1_1.jar
```

Also, $ORACLE_HOME/lib needs to be included in the LD_LIBRARY_PATH environment variable.

JServ Sample Servlet Configuration

Here is an example for Jserv on Solaris, using JDK1.2, with the following environment variables set to the following values:

```
ORACLE_HOME = /u/oracle/product/oracle/8.1.6
OMB_HOME = /u/oracle/product/oralce/8.1.6/omb/2.0
```

In this sample configuration, to enable the servlet, add the following lines in the file jserv.properties:

```
wrapper.classpath=/u/oracle/product/oracle/8.1.6/omb/2.0/mercury.jar
wrapper.classpath=/u/oracle/product/oracle/8.1.6/jlib/jssl-1_2.jar
wrapper.classpath=/u/oracle/product/oracle/8.1.6/jlib/javax-ssl-1_2.jar
wrapper.env=LD_LIBRARY_PATH=/u/oracle/product/oracle/8.1.6/lib
```

The initial parameters for the HTTP propagation servlet specify the Oracle Message Broker built-in HTTP listener of a receiving broker that the servlet sends messages to. Table 8–3 shows the HTTP servlet parameters.

For example, using Jserv, set these parameters as follows:

```
servlet.PropHttpServlet.initArgs=host=recvhost,port=80,ssl=1,wallet=/wallets/w1
passwd=welcome
```

Table 8–3 HTTP Propagation Servlet Parameters

Name	Description	Default
host	Host name of the machine that the receiving broker runs on.	mandatory
port	Port number that the receiving HTTP listener listens to.	80 if ssl=0 443 if ssl>0
ssl	SSL level of the receiving HTTP listener uses.	0
wallet	Full path name of wallet file used by the servlet.	
passwd	Password applied to the wallet file used by the servlet.	
proxyHost	Host name of the proxy server for the servlet.	
proxyPort	Port number of the proxy server for the servlet.	

You can access the servlet through a browser to check if the HTTP propagation servlet is installed and setup correctly. The servlet tries to connect to the receiving broker to obtain the propagation version number, upon receiving a request from the

browser. The receiving broker and its HTTP listener must be running for this test to run successfully.

Propagation Security

The Oracle message Broker supports the SSL protocol to secure propagation connections. Table 8–4 shows the SSL levels that an Oracle Message Broker administrator can specify for propagation.

Table 8–4 SSL Levels Supported for Propagation

SSL Level	Description
0	This specifies no authentication and no encryption on both the sending and the receiving sides. This is the default SSL level.
1	This level specifies only encryption on both sending and receiving sides.
2	This level specifies encryption and receiving side authentication.
3	This level specifies authentication and encryption on both the sending and the receiving sides.

Keep the following points in mind when configuring SSL for propagation:

- For SSL connections, the server determines whether the client certificate is required. With propagation, in terms of SSL, the server is the receiving broker and the client is the sending broker.
- If the sending broker uses SSL level 2, and receiving broker uses SSL level 3, then the receiving broker requests the sending broker to send its certificate. Since on the sending broker, a wallet is set that contains a valid certificate, the certificate from the wallet on the sending broker is sent to the receiving broker. There is no way to control this. In this case, the SSL level established is level 3.
- If the sending broker uses SSL level 3, and receiving broker uses SSL level 2, the receiving broker never requests the sending broker to send its certificate. In this case, the SSL level established is level 2.

In summary, if the sending broker uses SSL level 3 and receiving broker uses SSL level 2, the SSL connection is established using SSL level 2. If the sending broker uses SSL level 2 and the receiving broker uses SSL level 3, the SSL connection is established at SSL level 3 (see "Propagation Limitations" on page 8-28 for information additional limitations).

IIOP propagation Security

If a receiving broker is SSL enabled, it requires that connections from sending brokers be secured using SSL. See "Oracle Message Broker SSL Options" on page 12-26 for information on enabling SSL. On the propagation sending side, users specify SSL information, including the SSL level, wallet location, and wallet password, in the remote directories entry associated with propagation jobs that send messages to the SSL enabled receiving broker (see Table 4–19 for details on the remote directories entry).

To prevent unauthorized propagation, the administrator can protect the receiving broker. When starting a propagation job, the sending broker retrieves information from the receiving broker's directory entry to allow the sending broker to contact the receiving broker. The receiving side protects the LDAP Directory entries using LDAP Directory access control, which allows only authorized users with valid passwords access to the directory entries. At the sending side, a valid username and password must be provided in the remote directories entry for a propagation job to access a protected LDAP Directory (see Table 4–19 for details on the remote directories entry). Correct SSL information, including the SSL level, wallet location and wallet password must be included in the remote directories entry if the LDAP Directory is SSL enabled.

HTTP Propagation Security

The Oracle message Broker supports the SSL protocol to secure HTTPS propagation. This section covers the following:

- Enabling SSL on the Receiving Broker
- Enabling SSL on the Sending Broker
- Enabling SSL for Servlet Based Propagation

Enabling SSL on the Receiving Broker

Enable the HTTP Propagation security features using attributes in the `prop_http` entry under the `msg_broker` entry on the receiving broker (see Table 4–15 for the list of attributes). To enable SSL for a receiving broker, configure the built-in HTTP listener with a SSL level using a value greater than 0. When the SSL level is set to a value greater than 1, you need to include the proper wallet information (the wallet location and password).

Enabling SSL on the Sending Broker

Enable the HTTP Propagation security features on the sending broker using attributes in the RemoteHTTPListener entry associated with a propagation job (see Table 4–20 for the list of propagation job attributes). In order for a sending broker to process a propagation job whose receiving broker is SSL enabled, the associated remote HTTP listener entry of the propagation job must be set properly. The sending broker and the receiving broker negotiate SSL options, using the attributes specified in the remote HTTP listener entry on the sending side and the built-in HTTP listener SSL settings on the receiving side (see Table 4–15 for the list of `prop_http` attributes).

Enabling SSL for Servlet Based Propagation

For servlet based HTTP propagation, there are two HTTP connections (or two HTTPS connections, see Figure 8–6). In order to secure the HTTP connection between the sending broker and the HTTP propagation servlet, the web server must be SSL enabled, and the remote HTTP listener entry at the sending side that represents the propagation servlet must be set properly with respect to the web server's SSL configuration. That is, the SSL level in the remote HTTP listener entry must be set properly to correspond with the level in the web server, and the wallet must be valid with common trust points with that of the web server. In order to secure the HTTP connection between the servlet and the receiving broker, the receiving broker must be SSL enabled and the servlet must set its initial parameters (SSL, wallet, passwd) properly with respect to the SSL setting of the receiving broker.

Propagation Control

This section covers the following:

- Creating and Deleting Propagation Jobs
- Activating and Deactivating a Propagation Jobs
- Error Handling and Recovery

Note: The propagation manager is automatically disabled when the Oracle Message Broker is running in Local Mode. Thus, the Oracle Message Broker never activates propagation when it is running in Local Mode. Refer to the section, "Running in Local Mode" on page 5-8 for details on running in Local Mode.

Creating and Deleting Propagation Jobs

Propagation jobs can be created and deleted with the Oracle Message Broker running or shutdown. When a propagation job is created, if the source destination of the job is a topic, the propagation manager creates a durable subscriber on the topic, regardless of whether the Oracle Message Broker is running or whether the propagation job is activated. The durable subscriber does not have corresponding entries in the LDAP directory (as a user-created durable subscriber would). The durable subscriber is deleted when the associated propagation job is deleted.

When deleting a propagation job, first deactivate the propagation job. You are not allowed to delete an activated propagation job.

Before deleting a propagation job, first check the message log to make sure that there are no uncommitted propagation requests for the propagation job (see "Activating and Deactivating a Propagation Jobs" on page 8-26 for details). Deleting a propagation job with uncommitted propagation requests may leave messages in the source destination and the target destination, since the propagation manager only removes propagation jobs from the source destination as the last step before the propagation manager's actions are committed.

In order to avoid uncommitted requests, you can activate and deactivate the propagation job. Repeat this process until there are no more log messages indicating that there are uncommitted propagation requests for the propagation job.

It is possible that creating or deleting a propagation job may fail. This could leave the propagation job in invalid state. A propagation job in invalid state may cause unnecessary message accumulation if the job has a topic as its propagation source. Propagation jobs in invalid state should be deleted.

Activating and Deactivating a Propagation Jobs

The Oracle Message Broker administrator activates and deactivates a propagation job by setting the attribute `activation_state` in the propagation job entry. To activate a propagation job, set the attribute to true. To deactivate a propagation job, set the attribute to false (see Table 4–21 for a list of the propagation job attributes).

Activating a propagation job means the propagation manager can start processing the propagation job, and the sending broker starts sending messages to the receiving broker. Deactivating a propagation job means stopping to process the propagation job, the sending broker stops sending messages to the receiving broker. For a propagation job with a topic as propagation source, deactivating the propagation job stops only sending, not subscribing. Messages published to the

topic when the job is deactivated will be sent to the receiving broker once the job is activated again.

When a propagation job is being deactivated, the propagation manager attempts, for up to ten(10) seconds to complete uncommitted propagation requests for the propagation job. If the propagation manager cannot complete the requests, for example if the receiving broker is down, it writes a message to the Oracle Message Broker log file, in the format:

`There are [ number ] uncommitted propagation requests for [job id].`

Followed by a message similar to the following deactivated message:

`propagation job [ job id ] has been deactivated.`

> **Note:** If uncommitted requests exist and a propagation job is deleted, the uncommitted requests will never be completed. This may cause duplicated messages if a user later tries to recreate the same propagation job.

Error Handling and Recovery

When starting the Oracle Message Broker, administrators should pay a close attention to the Oracle Message Broker message log file. Check to see that propagation manager and any propagation jobs start successfully. For example, check to see if HTTP listeners start successfully for a receiving broker, and if propagation jobs are activated successfully for a sending broker.

When the propagation manager encounters failures, it stops the related propagation job and writes log messages into the Oracle Message Broker log file. A log message in the log file for the sending broker, in the following form indicates that some failures occurred when processing the propagation job:

`propagation job [ job id ] has been stopped.`

The propagation manager attempts to restart and recover failed propagation jobs once every two minutes, until it successfully recovers the jobs or the propagation job is deactivated.

Propagation Limitations

This section lists several limitations that apply for propagation, and for configuring propagation.

1. replyTo Limitation – If a client receives a message that was propagated from a remote queue by a foreign Oracle Message Broker, and the client tries to reply using the `replyTo` destination found in the message header, the following restriction applies:

   When the `replyTo` destination is not a local destination for the client, the Oracle Message Broker throws the exception:
   `java.jms.InvalidDestinationException`

2. Looped Propagation Jobs Limitation – looped propagation jobs, where there is a chain of propagation jobs, with a target in the chain pointing back to a source, are not detected by Oracle Message Broker. When and if propagation jobs are setup in a loop, messages will be transferred in an infinite loop. This behavior is also true for a single propagation job where the source and the target are the same destination. It is the Oracle Message Broker Administrator's responsibility to avoid creating looped propagation jobs.

3. Propagation is not supported for Oracle Message Brokers running in local mode.

4. Propagation is not supported for Oracle Message Brokers using lightweight configuration.

5. Propagation logging queues restriction. The propagation logging queues are used internally by the propagation manager. Client applications should never use these queues. Using either the propagation sending log queue or the propagation receiving log queue may result in unrecoverable propagation failures.

6. If an administrator specifies a mismatch between the security SSL setting for a propagation job on the sending broker and the receiving broker, the mismatch may cause propagation threads to hang when the propagation job is processed. This can occur when the SSL level is set to 1, 2, or 3, on the sending broker and the security SSL level for the associated HTTP listener on the receiving broker is set to 0 (non-SSL).

7. HTTP propagation does not support HTTP request redirect.

9

Oracle Message Broker C++ API

This chapter describes the Oracle Message Broker C++ API. Using the C++ API, clients written in C++ can use the Oracle Message Broker services and interoperate with clients written in Java.

The sample programs for the examples shown in this chapter are available in the directory, $OMB_HOME/samples/client/cpp.

This chapter covers the following:

Introduction

The Oracle Message Broker provides a C++ API by defining a set of C++ classes that *clone* the JMS Java classes and interfaces (these are defined in the `javax.jms` package for Java). The C++ API uses the same names and follows the same class hierarchy as the JMS Java API counterparts. If you are a C++ programmer familiar with the JMS specification you do not need to learn a new API to use the Oracle Message Broker C++ API. However, you need to learn several conversion rules between Java and C++. This chapter covers the C++ API, and shows the conversion rules for working with the C++ API and the Oracle Message Broker.

System Requirements

To compile applications that use the Oracle Message Broker C++ API, you need to use an ISO C++ compiler (ISO/IEC FDIS 14882). In addition, the Oracle Message Broker C++ API requires support for the `long long` type (this type is not

mandated by the ISO standard, but most compilers provide it). Some pre-ISO compilers may work, if they support namespaces, runtime type information, and the C++ Standard Template Library (STL).

Limitations

The Oracle Message Broker C++ API implements most of the features of the Java API, except:

- Only the `TextMessage` and `BytesMessage` classes are implemented. In `BytesMessage`, only the `reset()`, `readByte()`, `readUnsignedByte()`, `readBytes()`, `writeByte()`, and `writeBytes()` methods are implemented. Functions which are not implemented throw a `NotImplementedException` exception.
- Queues and topics cannot be fetched directly from the directory by a C++ application. They must be created using the `createQueue()` and `createTopic()` methods declared on sessions. The parameter to these methods is the distinguished name (dn) of the queue or topic in the directory. The Oracle Message Broker will use this name to fetch itself the queue or topic from the directory.

Major Differences between the Java and C++ APIs

Declaration

The Oracle Message Broker C++ API is declared in the scope of the "jms" C++ namespace. All the class declarations are imported by including the `jmscpp.hh` header file. An additional implementation-specific header file, `ImplDepFactory.hh`, must be included to obtain initial references to the Oracle Message Broker instance; once these initial references are obtained, only standard JMS constructs are used.

Types

While Java's basic types have well-specified sizes, the C++ specification is less restrictive concerning the size of built-in types. The Oracle Message Broker C++ API maps Java basic types to C++ basic types that have at least the same size, according to the ISO standard. Table 9–1 shows the mapping.

Table 9–1 Mapping Between Java and C++ Types

Java	C++
`short`	`short`
`int`	`long`
`long`	`long long`
`float`	`float`
`double`	`double`
`byte`	`unsigned char`
`string`	`wstring(basic_string<wchar_t>)`
`byte[]`	`vector<unsigned char>`

Memory Management

While Java has built-in garbage collection, the C++ programmer must explicitly allocate and deallocate memory. The rules for C++ memory management are as follows:

- Parameters passed by reference to a Oracle Message Broker C++ method are never deallocated by an Oracle Message Broker C++ library; they must be explicitly deallocated by the caller.
- Parameters passed by value by an Oracle Message Broker C++ method are copied during invocation and the caller is responsible for deallocating the original value; the Oracle Message Broker C++ library manages the life cycle of the copy.
- Parameters returned by reference by an Oracle Message Broker C++ method must be deallocated explicitly by the caller, even if they have been allocated by the Oracle Message Broker C++ library.
- Parameters returned by value by an Oracle Message Broker C++ method are copied during invocation and the caller is responsible for the returned copy's life cycle.

Sample Application

This section presents a sample Oracle Message Broker C++ application composed of two programs: a queue sender and a queue receiver. Most of the code for the sender and the receiver is similar. The code sections that differ are marked as sender or receiver specific.

Start the sender with arguments for the queue name and a text message. Start the receiver with a queue name. The queue name for the Oracle Message Broker C++ API is the dn of the queues entry in the directory. For example:

```
% sender cn=myQueue,cn=Queues,cn=MyOMB,cn=OMB,cn=Products,cn=OracleContext,
ou=oas,o=oracle,c=us "Hello World!"
% receiver cn=myQueue,cn=Queues,cn=MyOMB,cn=OMB,cn=Products, cn=OracleContext,
ou=oas,o=oracle,c=us
```

General Declarations

Both the sender and the receiver code include the Oracle Message Broker C++ headers, and define a utility function for converting a C string into a wide string and printing a wide string to the console. This common code is as follows:

```
#include <jmscpp.hh>
#include <ImplDepFactory.hh>

wstring to_wstring(char *str)
{
  string s(str);
  wstring ws(s.length(), ' ');
  copy(s.begin(), s.end(), ws.begin());
  return ws;
}

ostream& operator <<(ostream& os, const wstring& ws)
{
  copy(ws.begin(), ws.end(), ostream_iterator<char>(os));
  return os;
}
```

Initialization

The sender and the receiver first obtain a reference to an Oracle Message Broker specific `ImplDepFactory` object that creates a JMS connection factory. There are two different constructors for the `ImplDepFactory` object:

```
ImplDepFactory(const wstring& ior_file, const wstring& driver_name,
               bool unused, const wstring& cid, long priority,
               long tx_timeout);
```

Where the arguments are as follows:

ior_file	The name of a file that contains the Oracle Message Broker IOR. This IOR can be obtained from the `msg_broker` entry in the directory or from the Oracle Message Broker log file.
driver_name	The driver name, this can be one of: vol, aq, mq, mcast, rv.
unused	An unused boolean value.
cid	Arbitrary name identifying of the client
priority	Default priority of messages
tx_timeout	Default transaction timeout

```
ImplDepFactory(const wstring& provider_ior, const wstring& driver_name,
               const wstring& cid, long priority, long tx_timeout);
```

Where the arguments are as follows:

provider_ior	The Oracle Message Broker IOR. This IOR can be obtained from the `msg_broker` entry in the directory or from the Oracle Message Broker log file.
driver_name	The driver name, this can be one of: vol, aq, mq, mcast, rv.
cid	Arbitrary name identifying of the client
priority	Default priority of messages
tx_timeout	Default transaction timeout

C++ applications can fetch the Oracle Message Broker IOR directly from the directory using OID's LDAP C libraries. A sample program that performs this task is included with the C++ samples.

The sender and the receiver next create a queue connection factory, a queue connection, a queue session, a queue, and finally start the connection.

The following code shows these steps.

```
int main(int argc, char **argv)
{
  try {
    ImplDepFactory idf(L"JMSProvider", L"vol", true, L"client", 4, 0);
    QueueConnectionFactory *cf = idf.createQueueConnectionFactory();

    // Create connection and session
    QueueConnection *conn = cf->createQueueConnection();
    QueueSession *sess = conn->createQueueSession(false,
          Session::IMMEDIATE_ACKNOWLEDGE);
    Queue *queue = sess->createQueue(to_wstring(argv[1]));

    // Start connection
    conn->start();
    }
}
```

Sending Messages (Sender Specific)

The queue sender creates a sender, constructs a text message, and sends the text message using the sender. The message and the sender are then explicitly deallocated.

```
    // Sender-specific code
    QueueSender *sender = sess->createSender(queue);
    // Create and send message
    Message *msg = sess->createTextMessage(to_wstring(argv[2]));
    sender->send(msg);
    sender->close();
    delete msg;
    delete sender;
```

Receiving Messages (Receiver Specific)

The queue receiver creates a receiver, waits for a text message, prints the text message, closes the receiver, and deallocates the message and the receiver.

```
    // Receiver-specific code
    QueueReceiver *receiver = sess->createReceiver(queue);
    // Receive message
    Message *msg = receiver->receive();
```

```
    TextMessage *tm = dynamic_cast<TextMessage*>(msg);
    if(tm != NULL)
      cout << "Received message: " << tm->getText() << endl;
    receiver->close();
    delete msg;
    delete receiver;
```

Cleanup

Finally, the sender and the receiver close open sessions and connections, and deallocate all Oracle Message Broker C++ objects.

```
    conn->stop();
    sess->close();
    conn->close();
    delete queue;
    delete sess;
    delete conn;
    delete cf;
  } catch(JMSException e) {
    cerr << "Unexpected exception: " << e.getMessage() << endl;
  }
  return 0;
}
```

10

Logging and Troubleshooting

This chapter covers the format and location of the Oracle Message Broker log file. This chapter has the following sections:

Working with Log Files

The Oracle Message Broker stores status and error messages in Oracle Message Broker log files. A log file includes information that is helpful for diagnosing problems and for tracking the operation of the Oracle Message Broker. Log files allow Oracle Message Broker administration and support personnel to diagnose problems with the system.

By default, the log file is named omblog with the appended values as shown below, or the name is specified using the string defined in the Java property oracle.oas.mercury.logName.

omblog-*hostname*-*time*

where:

hostname is the name of the system where the Oracle Message Broker is running.

time is the value of the current time, in milliseconds, when the omblog file is created.

Logging Directory

When running in remote mode, the Oracle Message Broker creates log files in the directory $OMB_HOME/logs if $OMB_HOME is defined and $OMB_HOME/logs is writable (on Windows NT systems, the directory is %OMB_HOME%\logs if %OMB_HOME% is defined and %OMB_HOME%\logs is writable). Otherwise, log files are created in the current directory.

When running in local mode, the Oracle Message Broker creates the log file in one of the following two directories:

- The current directory
- The directory that is specified using the Java property oracle.oas.mercury.logDirectory. The directory specified with the oracle.oas.mercury.logDirectory property must be writable.

When running in Local Mode, the Oracle Message Broker appends information to the log file, if the specified log file already exists. It does not overwrite existing logging information.

The Oracle Message Broker administrator is responsible for deleting unused log files.

DMS Metric Log Files

Use the `MsgBroker` command with the `-stats` option to save the collected Dynamic Monitoring Service metrics to the DMS log file. `MsgBroker -stats` saves the DMS log file to a file with the same name as the associated Oracle Message Broker log (omblog) file, prepended with “dms-”. Refer to "Working with Log Files" on page 10-1 for information on the omblog file name and the directory where the file is written.

The `-stats` option includes a parameter that specifies whether information is appended to the DMS log file, or if an existing log file is replaced. The `-stats` option also specifies the format for the data in the DMS log file. Refer to Table 2–2 on page 2-7 for detailed information on the available `MsgBroker -stats` options.

Refer to "Collecting Runtime Metrics" on page 6-3 for information on the format of the information saved to the DMS log file.

Logging Security Exceptions

The Java property oracle.oas.mercury.sec.trace set the level of logging for conditional tracing of security exceptions (SecException). If this system property is set, either on the command line or programmatically, then all authentication / authorization exceptions are traced in the Oracle Message Broker log file. The value of the property is not considered, but only if it is set, or not.

If this property is not set, then only security exceptions (SecException) arising out of internal errors, invalid configurations, or client attempts to unsuccessfully spoof JMS connections are traced in the Oracle Message Broker log file.

Problems and Common Solutions

This section describes some common problems, and workarounds for these problems. The section is divided as follows:

MQ Series Driver Problems

`MQSeries driver: instantiation failure - can't find MQSeries java binding`

The CLASSPATH does not include the MQSeries java support. Assuming a normal MQSeries 5.1 installation, the following must be added to the CLASSPATH:

```
/opt/mqm/java/lib /opt/mqm/java/lib/com.ibm.mqbind.jar
/opt/mqm/java/lib/com.ibm.mq.jar
```

Note: Oracle Message Broker must be running on the same system as the MQSeries queue manager.

The log queue specified in the MQSeries server entry for Oracle Message Broker, must have been separately created in the queue manager using MQSeries administration tools.

Runtime Exceptions

Security Exceptions

When an incorrect exported wallet password is given, the Java SSL throws the following exceptions:

1. Incorrect wallet password specified at the server:
   - Exception seen at the server side:

     ```
     Exception: javax.net.ssl.SSLException: SSL handshake failed:
     SSLBadParameterErr
     ```

   - Exception seen at the client side:

     ```
     javax.net.ssl.SSLException: SSL handshake failed:
     SSLConnectionClosedGraceful
     ```

2. Incorrect wallet password specified at the client:
   - Exception seen at the server side:

     ```
     Exception: javax.net.ssl.SSLException: SSL handshake failed:
     SSLConnectionClosedGraceful
     ```

   - Exception seen at the client side:

     ```
     javax.net.ssl.SSLException: SSL handshake failed: SSLUnknownErr
     ```

LDAP Directory Naming Exceptions

When an entry specified as a DN does not exist in the directory, javax.naming reports the following exception:

```
javax.naming.NameNotFoundException
```

Administration Problems When Running ombadmin

When starting the utility `ombadmin`, the following exception indicates your environment may not be appropriately initialized. To initialize your environment, use the ombenv scripts found in the bin directory (refer to "Working with the Administration Utilities" on page 2-2 for more information). Also this exception occurs if the DISPLAY variable is not set correctly on Solaris.

```
Can't find class oracle.oas.admin.gui.ombadmin
```

11

Administration GUI

This chapter describes the features of the Oracle Message Broker Manager. The Oracle Message Broker Manager creates and manages configuration information stored in the Oracle Message Broker's LDAP Directory. This chapter covers the following Oracle Message Broker Manager topics:

- Terminology
- Starting Oracle Message Broker Manager
- Connecting to a Directory Server
- Navigating Oracle Message Broker Manager
- Disconnecting from a Directory Server
- Performing Administration Tasks

> **Note:** Oracle Message Broker Manager allows you to work with configuration information that is stored as directory entries in an LDAP Directory. The directory entries are OMB entries. OMB entries are entries that are created with the Oracle Message Broker administrative tools. An LDAP Directory also contains non-OMB entries. For non-OMB entries, Oracle Message Broker Manager allows you to view and delete the entries, but not to modify the entries.

For a list of the terms used in this chapter, see "Terminology" on page 11-2.

Terminology

Throughout this chapter, we use the following terminology:

DN	Refers to an LDAP distinguished name
RDN	Refers to an LDAP relative distinguished name

Starting Oracle Message Broker Manager

The Oracle Message Broker administration command, `ombadmin` starts the Oracle Message Broker Manager. To start the Oracle Message Broker Manager, follow the instructions for your platform:

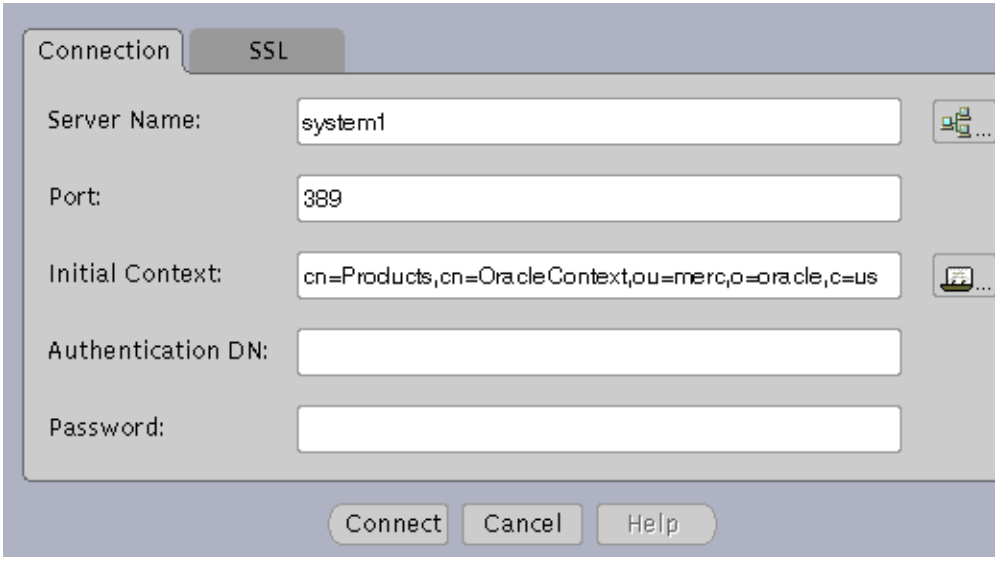
Platform	Instructions
Windows NT	You can either: ■ Type at the Run command: `ombadmin` or ■ Type at a DOS command prompt: `ombadmin`
Sun Solaris	Type at the system prompt: `ombadmin`

The executable is in the $OMB_HOME/bin directory (or %OMB_HOME%\bin on Windows NT systems).

Connecting to a Directory Server

When you start Oracle Message Broker Manager, you need to connect to an LDAP Directory server. The LDAP Connection dialog box (Figure 11–1) prompts you for the server name, port number, initial context, authentication DN, and password for the LDAP server. The program supplies default values for these fields using the OMB_LP environment variable. If you want to change the default values, make your selections in the dialog box.

Figure 11–1 LDAP Connection Dialog Box



In each field of the LDAP Connection dialog, type the information specific to the LDAP Directory you want to use.

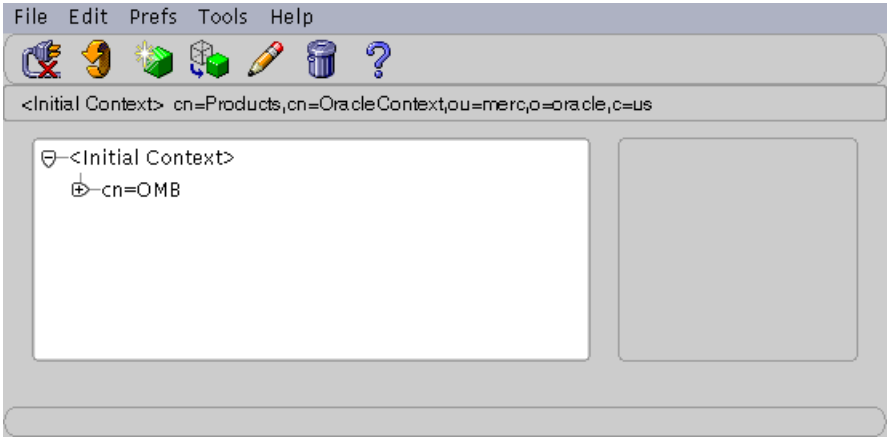
Field	Description
Server Name	The name of the server running the LDAP Directory that you want to view or modify. The icon to the right of the Server Name field presents the Select Directory Server window that allows you to select a server name and port combination. Using this window, you can also save or edit entries on the list.
Port	The port number for the server running the LDAP Directory.
Initial Context	The initial context for the Oracle Message Broker configuration entries stored in the LDAP Directory. This is the DN of the LDAP entry that serves as the base entry. Leave this field blank to connect to the root context of the directory. The icon to the right of the Initial Context field presents the Quick List containing saved DNs that can be selected for the Initial Context field. Using this window, you can also save or edit Quick List entries

Field	Description
Authentication DN	The DN of a user entry for LDAP Directory authentication. For information on LDAP authentication and recommended security roles, see "LDAP Server Security" on page 12-5.
Password	The password for the user associated with the specified Authentication DN. For information on LDAP authentication and recommended security roles, see "LDAP Server Security" on page 12-5.

When you select the Connect button, a connection to the specified LDAP Directory is created. If you select the Cancel button, the Oracle Message Broker Manager screen comes up and a connection is not attempted. From the Manager screen, you can connect and disconnect to LDAP servers, view entries, create new entries, delete entries, and modify entries.

Figure 11–2 shows the Oracle Message Broker Manager opening window.

Figure 11–2 Oracle Message Broker Manager Opening Window



Navigating Oracle Message Broker Manager

The Oracle Message Broker Manager window serves as the main window for the application. From this window, menu options and toolbar buttons allow you to perform tasks such as:

- Connecting to a directory server
- Viewing Oracle Message Broker entries
- Creating and Modifying Oracle Message Broker entries
- Deleting Oracle Message Broker entries

The Navigator pane (left side of the double window interface) contains a tree structure to browse entries that are relative to the initial context of the current server connection. Entries above the initial context are not accessible. To change the initial context or the server, you need to disconnect and then connect.

When Oracle Message Broker Manager first opens, the Navigator pane shows only one tree item, "<Initial Context>." By clicking the plus sign(+) next to the tree item, subcomponents of that tree item appear. Tree items that have plus signs in front of them may have their own sub-tree items. The plus sign becomes a minus sign (-) when the entry is expanded. You can expand and contract the tree by clicking the plus signs and minus signs.

For example, if you click the plus sign next to <Initial Context> in the opening window Navigator pane, the tree expands to show the top level entries in the initial context to which you are connected, as shown in Figure 11–2.

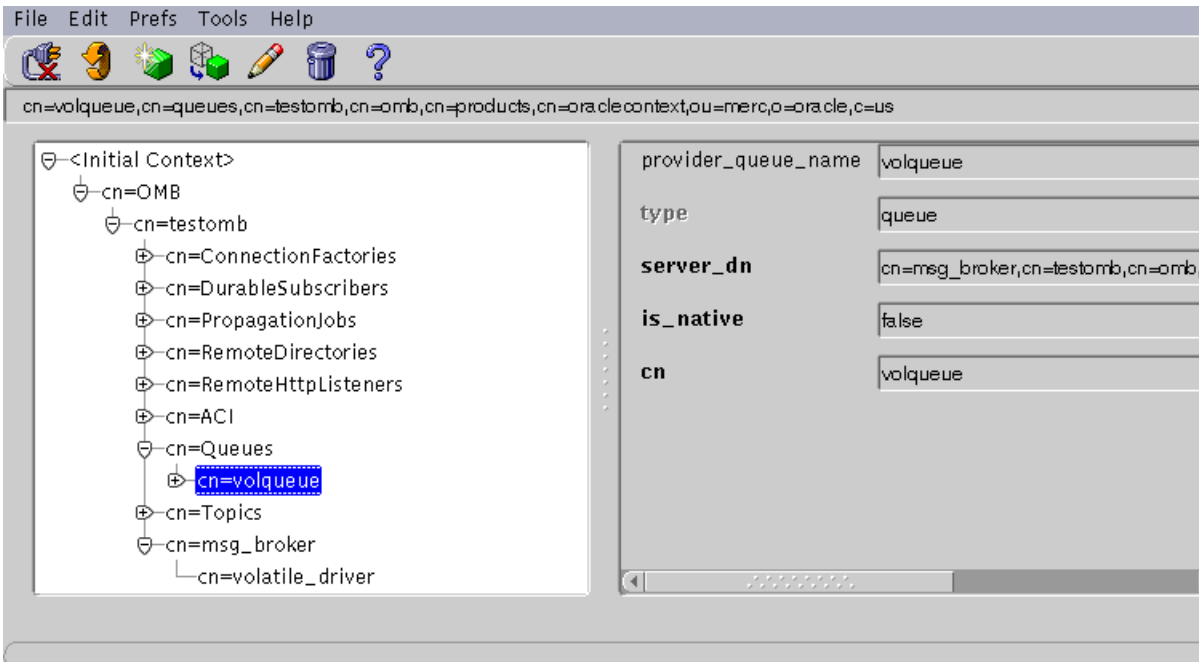
You can navigate around Oracle Message Broker Manager using one or a combination of the following options.

- Select menu items from the menus across the top of the window.
- Use buttons on the toolbar across the top of the left pane to perform such actions as create, create-like, and delete.
- Click the tree items in the Navigator pane. For example, as shown in Figure 11–3, clicking the Navigator tree item testomb causes the tree item to be highlighted and the entry's attributes to be displayed in the right panel. The attributes shown can be modified by selecting Show Assigned Attributes Only from the Prefs menu.

Attribute names in Oracle Message Broker Manager are shown in a bold, normal, or grayed out font.

- Bold names indicate mandatory attributes, where a value is required when the attribute is created (see **cn** in Figure 11–3).
- Names using a regular font indicate optional attributes (these can be modified after creation).
- Grayed out names indicate read only attributes that cannot be changed using the Oracle Message Broker Manager.

Figure 11–3 Manager Entry View



Oracle Message Broker Manager Menu Bar

Table 11–1 lists the menus you can access by using the menu bar, and describes the items in each menu. Menu items may become enabled or disabled depending on the item you are displaying in Oracle Message Broker Manager.

Note: Many menu options are disabled if a server connection is not currently established.

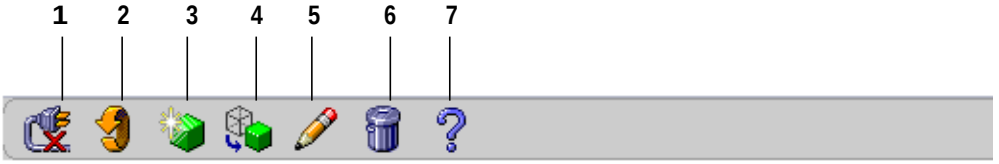
Table 11–1 Menu Bar Items

Menu	Menu Items
File	Create – Adds an entry. Dialogs guide you through the creation process. Create Like – Adds a new entry by using the entry selected in the Navigator pane as a template. Connect – Establishes a new server connection. Once a connection is established you must disconnect before connecting to another server. Disconnect – Disconnects the current server connection. Exit – Exits Oracle Message Broker Manager.
Edit	Modify – Displays a dialog which allows you to modify the attributes for the currently selected entry. This option only allows Oracle Message Broker entries to be modified. Delete – Deletes the currently selected entry. Quick List – Save information displayed by the Oracle Message Broker Manager in a quick list that is saved between invocations of the Manager. Refresh – Refreshes the information displayed in the Manager window.
Prefs	Show Assigned Attributes Only – This checkbox, when selected, tells the Oracle Message Broker Manager to only display attributes that currently have an assigned value. When not selected, all attributes are displayed. Confirm Delete – This checkbox, when selected, enables the confirmation dialog when entries are deleted. When this is not selected, there is no confirmation for deletes. The default value is to confirm for deletes.
Tools	Provides options to run the available OMB Wizards that create entries using a series of dialogs. For more information, see "Using the Configuration Wizards to Add Entries" on page 11-15.
Help	Contents – Displays the Contents tab page of the Help Navigator. Hide – Hides the help window, if it is currently displayed. Index – Displays the help index. Search for Help On...– Displays the Help Search dialog box which you use to search for words in the online help guide. About – Displays Oracle Message Broker Manager version information.

Oracle Message Broker Manager Toolbar

Figure 11–4 and the accompanying table illustrate and describe the Oracle Message Broker Manager toolbar. Buttons become enabled or disabled depending on the item you are displaying in the Oracle Message Broker Manager and the current state, connected or disconnected.

Figure 11–4 Oracle Message Broker Manager Toolbar



Button	Purpose
1	Connect / Disconnect – Connects or disconnects to or from a directory server.
2	Refresh – Updates data for objects stored in memory to reflect changes in the directory.
3	Create – Adds a new entry
4	Create Like – Adds a new entry by using another entry as a template
5	Modify – Modifies an entry
6	Delete – Deletes an entry. This can also delete subentries when an entry contains subentries
7	Help – Displays the Help Navigator

Disconnecting from a Directory Server

To disconnect from a directory server using Oracle Message Broker Manager, go to the File menu and select Disconnect or select the disconnect icon in the toolbar. Also, when you exit Oracle Message Broker Manager, the connection is closed.

Performing Administration Tasks

This section gives step-by-step instructions for several common Oracle Message Broker Manager tasks, including:

- Viewing Entries
- Adding Entries
- Deleting Entries
- Modifying Entries
- Using the Configuration Wizards to Add Entries

Viewing Entries

In the Navigator pane, expand the <Initial Context> to show the top level entries in the directory. The initial context, (top) of the tree is listed first, and then the second level and so forth, moving from left to right. The subtree lists the RDN of each entry in hierarchical order. To see the lower level entries within any subtree, click the plus (+) sign to the left of the parent entry. Once you are at the entry you want to view, select the entry to view its attributes.

The right panel displays the attributes for a selected entry. This panel can show all available attributes, or only the attributes which have an assigned value (to change this option, use the Prefs menu item, “Show Assigned Attributes Only”).

Attribute names in Oracle Message Broker Manager are shown in a bold, normal, or grayed out font. Bold names indicate mandatory attributes, where a value is required. Names using a regular font indicate optional attributes. Grayed out names indicate read only attributes that cannot be changed using the Oracle Message Broker Manager.

Adding Entries

You can use Oracle Message Broker Manager to add entries as described in the following sections:

- Adding A New Entry
- Adding an Entry by Copying an Existing Entry

Adding A New Entry

To add entries with Oracle Message Broker Manager, you must have an active connection to a directory server.

To add a new entry:

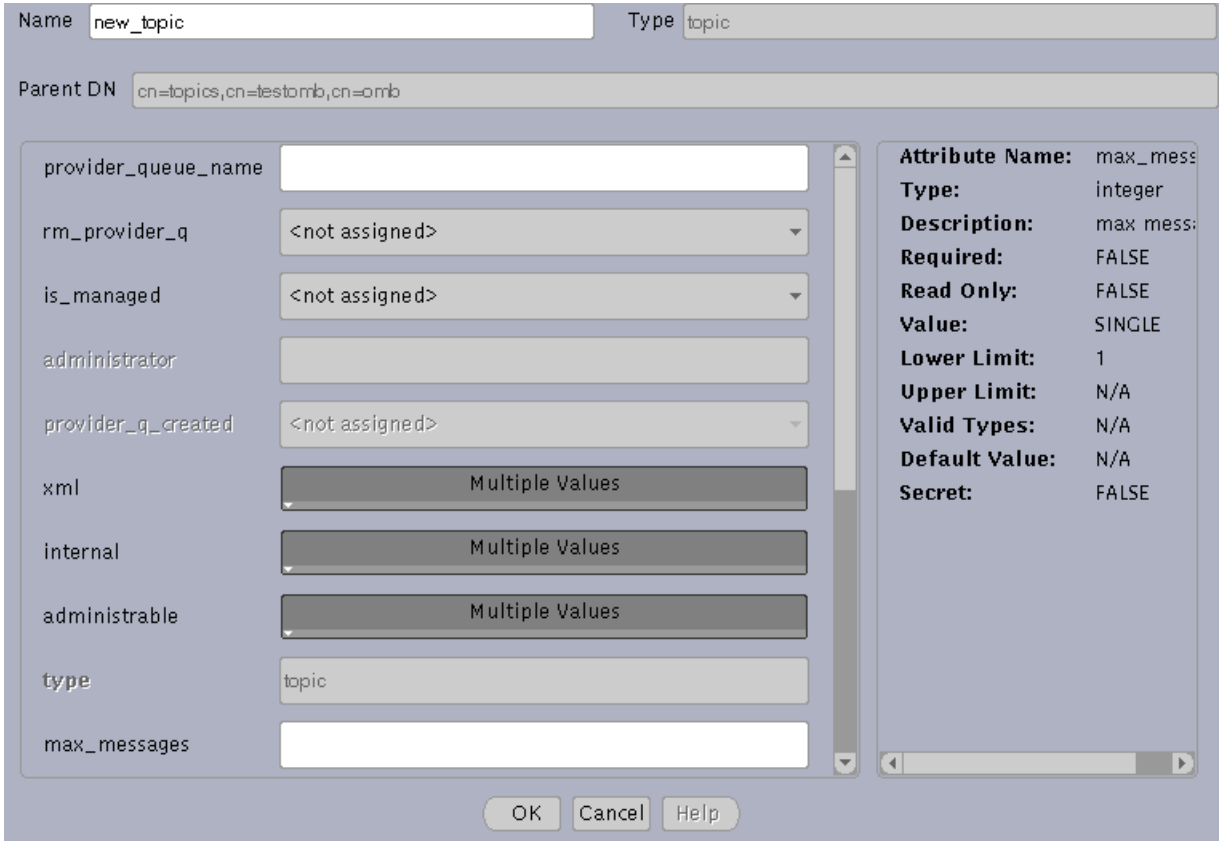
1. Either click the Create button on the toolbar or select Create from the File menu. The New Entry dialog box appears (Figure 11–5).

Figure 11–5 Adding a New Entry

2. Select the desired type from the Type list.
3. Select the parent for the new entry from the Parent list. The Parent list contains all valid parents within the initial context that may contain the type of entry you are adding. This list shows <Initial Context> if the only available parent is the entry corresponding to the initial context. This list shows nothing if no entries currently exist that can serve as a parent for the selected entry type. The parent entry must be created before a child entry.
4. Enter the name for the entry you are adding in the Name field and press OK. The name for some entry types is fixed. For entries with fixed names, you cannot modify the Name field and the field is grayed out.

5. After you press OK in the New Entry dialog, the New Entry Attributes dialog appears (see Figure 11–6). This dialog allows you to view all the attributes for the new entry, and to supply values for attributes that can be modified.
6. Using the New Entry Attributes dialog, enter the desired attribute values and press OK to add the entry, or Cancel to cancel the operation.

Figure 11–6 Setting New Entry Attributes



The left panel in Figure 11–6 shows the attributes and provides fields to enter values or menus containing values for each attribute.

The right panel shows the meta data for the attribute with the current focus. Moving the mouse over an attribute in the left panel changes the focus to that

attribute. Meta data lists information such as limits for the data allowed for an attribute, whether a default value is available, and other attribute information.

Notes for Adding a New Entry

1. There is a fixed hierarchy that governs the parent and child relationship between OMB entry types. The Parent list presented in the New Entry dialog box contains only the entries that can serve as the parent of the entry type that is being added (see Figure 11–5). If the parent does not exist, it must be created prior to creating an entry. For example, a queue_container entry must exist before a queue entry can be created.
2. The omb_instance type corresponds to the cn=OMB entry. This entry is created by the Install process, along with the other entries making up the cn=OMB, cn=Products, cn=OracleContext DN. The omb_instance_container type is the topmost entry that can be created using Oracle Message Broker Manager.
3. An attribute of type Distinguished Name requires a string with an LDAP distinguished name syntax. Specify the full DN for such an attribute; a DN relative to the initial context is not sufficient. For example, a topic or queue entry has a server_dn attribute of type Distinguished Name, which must be the full DN of an existing msg_broker, aq_server, aqlite_server, mq_server, rv_server, or mcast_server entry. Obtain the full DN for an entry from the Oracle Message Broker Manager window which displays the currently selected entry's DN in a text field along the top of the window.

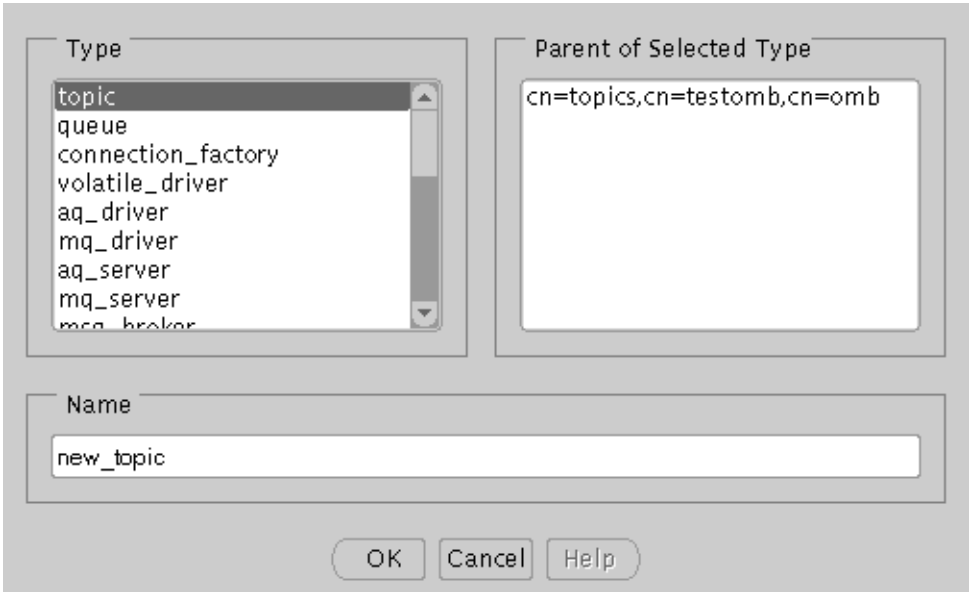
Adding an Entry by Copying an Existing Entry

You can use Oracle Message Broker Manager to create a new entry by copying from an existing entry. When you copy from an existing entry, you need to select the entry that you want to use as the template for the new entry, and you need to select the name and the parent for the new entry.

To add an entry by copying an existing entry:

1. In the Navigator pane, expand the <Initial Context>. Once you have reached the entry that you want to copy, select the entry.
2. Either click the Create Like button on the toolbar or select Create Like from the File menu. The New Entry dialog box appears (see Figure 11–7). Modify the attributes that you want to change in the new entry.
3. The value in the Type field matches the type of the item you selected. Select the parent for the new entry from the available choices in the Parent list.
4. Finally, type the name for the entry you want to add and press OK.

Figure 11–7 New Entry Dialog Box



5. After you press OK in the New Entry dialog, the New Entry Attributes dialog appears (see Figure 11–6). All the values from the selected entry are copied in their respective fields to the new entry. Change the fields you need to tailor your new entry.

   Keep the following in mind when editing the attributes for the new entry:

   - Carefully look over the new attributes in the New Entry Attributes dialog. If any entries are not correct for the new entry, change them. In particular, if you are using create like with different OMB Instances as parents, make sure that the new entry uses DNs that point to entries within its OMB Instance, and not to DNs that point to entries within the parent of the selected entry.
   - If you are using create like to add a new queue or a topic, make sure that the DN specified for the server_dn attribute that the queue or topic points to in the copied entry exists within its parent's OMB Instance.
   - If you are using create like to add a new queue or topic, make sure that the value of provider_queue_name attribute is correct. If the value for this attribute is not correct, results are unpredictable.

6. Finally, press OK to add the entry, or Cancel to cancel the operation.

Deleting Entries

You can use Oracle Message Broker Manager to delete directory entries. You can delete OMB entries, which are entries created with the Oracle Message Broker administrative tools. You can also delete non-OMB entries. The confirm delete preference is available for deleting entries. When selected, this preference enables the confirmation dialog for each delete operation. When Confirm Delete is not selected, there is no confirmation for delete operations.

Warning: Deleting entries removes the entries from the directory. When you delete entries, be certain that you are not selecting entries that you, or other users need. It may be very difficult to recover deleted entries.

To delete an entry:

1. In the Navigator pane, expand the Initial Context to show entries in the directory. Select the entry that you want to delete.
2. Either click the Delete button on the toolbar or select Delete from the Edit menu.
3. Delete presents a confirmation dialog. Press Yes to delete the entry, or No to cancel the operation.

Modifying Entries

You can use Oracle Message Broker Manager to modify the attributes of an existing entry. Non-OMB entries cannot be modified using Oracle Message Broker Manager. If you need to modify a non-OMB entry, you need to use another tool, for example `oidadmin`, or `ldapmodify`.

Note: Always use Oracle Message Broker administration tools to create, modify, and delete OMB entries. Do not use other tools, such as `oidadmin` or `ldapmodify` to create, modify, or delete OMB entries.

To modify an entry:

1. In the Navigator pane, expand the <Initial Context> to show entries in the directory. Once you have reached the entry that you want to modify, select the entry.
2. Either click the Modify button on the toolbar or select Modify from the Edit menu. The Edit Entry Attributes dialog appears. The Edit Entry Attributes dialog looks and behaves similar to the New Entry Attributes dialog (see Figure 11–6).
3. Edit the attribute values that you want to modify, and press OK to modify the entry or Cancel to cancel the operation.

Using the Configuration Wizards to Add Entries

You can use Oracle Message Broker Manager to create entries and subentries by selecting a configuration wizard from the Tools menu. The tools menu provides the following wizards:

- OMB Instance Configuration Wizard – this wizard helps you configure an OMB Instance. An OMB Instance contains all of the directory entries required to run the Oracle Message Broker.
- Queue Wizard – this wizard helps you create one or more queues.
- Topic Wizard – this wizard helps you create one or more topics.
- Propagation Configuration Wizard – this wizard helps you to configure the entries required to setup the Oracle Message Broker for propagation.
- Propagation Job Wizard – this wizard helps you create and configure propagation jobs.

To run a wizard, do the following:

1. Select, the desired wizard from the Tools menu. For example, if you select the OMB Instance Configuration Wizard, this wizard appears (see Figure 11–8).
2. Follow the instructions provided with the wizard to create and configure the entries associated with the wizard.

Figure 11–8 New OMB Instance Wizard Dialog

12

Security

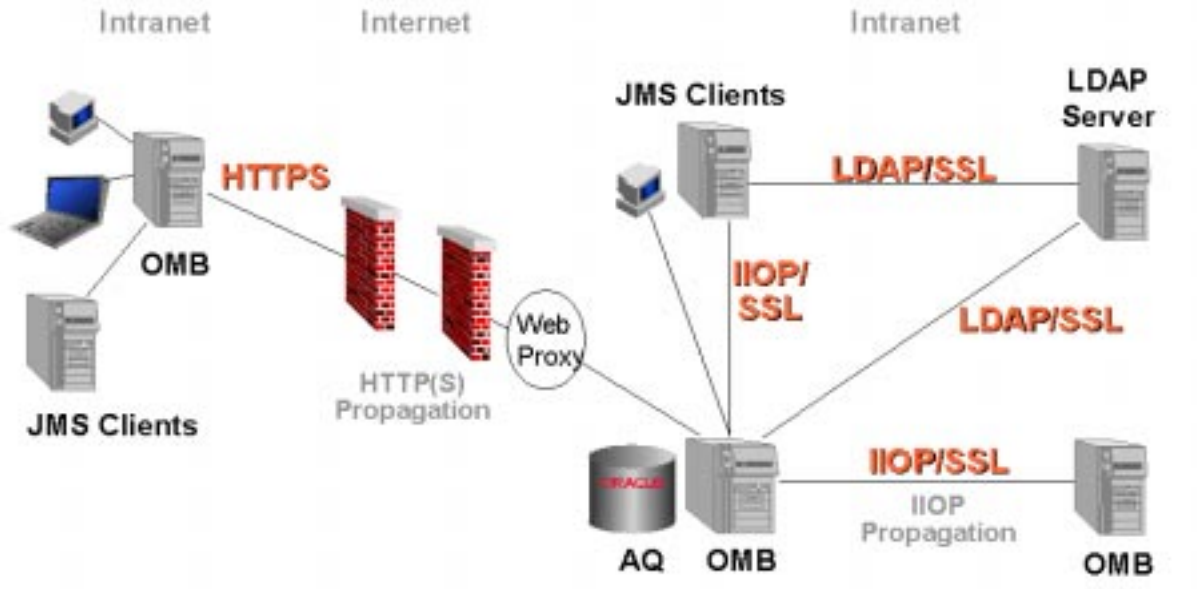
The Oracle Message Broker security features are integrated with the Oracle Message Broker. To ensure a secure system, it is essential that the Oracle Message Broker administrator understand the security requirements for the installation, and the security features available with the Oracle Message Broker.

This chapter covers the following:

- Features and Assumptions
- Security Components
- LDAP Directory Server Security Administration
- Oracle Message Broker Security Administration
- Provider Security Administration

Figure 12–1 shows the components of a sample Oracle Message Broker deployment.

Figure 12–1 Sample Oracle Message Broker Deployment



Features and Assumptions

To implement its security features, the Oracle Message Broker supports security in the following areas:

- LDAP Server Security – including mechanisms for protecting the LDAP Directory information from unauthorized access. Also, Secure Sockets Layer (SSL) is used for securing the network connections to the LDAP server.
- Oracle Message Broker and JMS Client Security – including SSL for securing network connections and authentication and authorization. This security area secures the connection between a JMS Client and an active Oracle Message Broker, and between Oracle Message Brokers when using propagation. This security area also includes the Oracle Message Broker Security Service to control access to Oracle Message Broker destinations (queues and topics).
- Oracle Message Broker Provider Security – the Oracle Message Broker drivers support several different providers, called message servers, for storing messages. The Oracle Message Broker supports provider defined security features specific to particular message servers.

In addition, the Oracle Message Broker supports the following security features:

- An encryption facility to store passwords in the LDAP Directory.
- Security for all network connections. This provides support for data integrity, privacy, certificate based authentication, and non-repudiation.
- Customization of the security features that a particular installation utilizes.
- Allowing administrators to specify a username and password in connection factories and using this information as credentials while accessing an underlying provider (driver).

SSL Overview

Secure Sockets Layer (SSL) is an industry standard protocol for securing network connections. SSL provides:

- Encrypted connections (privacy)
- Authentication - server-side or server-side and client-side
- Integrity (a third party cannot modify data)
- Non repudiation

SSL provides authentication through the exchange of certificates that are verified by trusted certificate authorities. A certificate ensures that an entity's identity information is correct. Additional information on SSL is available at the following web site,

http://home.netscape.com/eng/ssl3/ssl-toc.html

Programming and Administration Control and Assumptions

In order to ensure that an Oracle Message Broker installation is as secure as possible, in this chapter we assume that an administrator understands the following:

- The LDAP Server and its administration
- SSL
- JNDI
- Provider security (Oracle Message Broker drivers)

Furthermore, to ensure Oracle Message Broker security, the application programmer is responsible for making the correct JNDI and JMS API calls.

Administration Summary

The following components can be configured independent of each other:

- LDAP server authentication / authorization – users / groups configuration and access control information setup for the LDAP server is performed using the LDAP server configuration tools. If using OiD, `ldapsearch` / `ldapmodify` and / or `oidadmin` can be used for this purpose.
- Enabling SSL for the LDAP server SSL configuration is performed using the LDAP server configuration tools. If using OiD, `ldapsearch` / `ldapmodify` and / or `oidadmin` can be used for this purpose.
- Enabling SSL for the ORB – Enabling SSL for the ORB is done through ORB tools. For SSL connections, Oracle Message Broker uses Oracle Wallets. The administrator needs to use the Oracle Wallet Manager to manage wallets.
- Configuring message provider specific security.
- Enabling proxy and SSL for HTTP propagation.
- Protecting Oracle Message Broker destinations.

Administration Tasks

It is the administrator's responsibility to setup security. By default, on installation security is not enabled. The administrator needs to perform the following optional tasks to secure Oracle Message Broker:

- Enabling SSL for the ORB
- Enabling SSL for the LDAP Directory directory
- Controlling access to the LDAP Directory entries
- Using provider / driver specific security features
- Enabling proxy and SSL for HTTP propagation
- Creating Oracle Message Broker users, groups, and ACLs.

Error Reporting

All fatal errors and malicious attempts to breach Oracle Message Broker security are logged in the Oracle Message Broker log file. Security related non-fatal errors are logged in the Oracle Message Broker log file if the Java property oracle.oas.mercury.sec.trace is defined.

The Oracle Message Broker never logs passwords in exception messages.

Security Components

This section covers the following:

LDAP Server Security

Using LDAP based configuration the Oracle Message Broker stores its configuration information in the LDAP Directory. By controlling read access to LDAP Directory entries containing destinations, connection factories, and the `msg_broker` entry, the Oracle Message Broker administrator can control access for the JMS Client to the Oracle Message Broker and to its destinations.

Note: This chapter does not cover security with Lightweight Configuration security.

If a JMS client cannot read the msg_broker entry in an OMB Instance stored in an entry in the LDAP Directory, then it cannot connect to the Oracle Message Broker. A connection factory is required to connect to the Oracle Message Broker.

If a JMS client does not have read access to a particular Oracle Message Broker destination entry (topic or queue), it is not aware of the destination and therefore will not be able to request access to it from the broker.

Access limitations for information in the directory consists of authentication and authorization. When an LDAP client connects to an LDAP server, the client uses a username / password to authenticate itself. The server then evaluates the access control information (ACI) for the directory entry / attribute requested and based on this evaluation allows the client access to the information or denies it.

This section covers the following areas of LDAP Server security:

- LDAP Directory Authentication
- LDAP Directory Authorization
- LDAP Directory Secure Sockets Layer Connections
- Security Roles

LDAP Directory Authentication

Authentication is the process by which the LDAP Server establishes the true identity of the user connecting to the LDAP Directory (for information on setting up user entries, refer to "LDAP Directory Server Security Administration" on page 12-19).

Directory authentication occurs when an LDAP session is established by connecting to a directory. Every LDAP session has an associated user identity. This user identity is also referred to as the **authorization ID**. To ensure that LDAP Directory users' and clients' identities are correctly known, the Oracle Message Broker provides two options for connecting to the LDAP Directory: anonymous and simple authentication. Table 12–1 describes these authentication options.

Table 12–1 Directory Authentication Options in Oracle Message Broker

Authentication	Description
Anonymous	If the LDAP Directory is available to everyone, then you can allow users to log in to the directory anonymously. In this case, when using Oracle Message Broker commands, administrators simply leave the user name and password fields blank when they are prompted (or specify blank values using the command line options, or use the -noauth option). Each anonymous user then exercises whatever privileges are specified for anonymous users.
Simple	In this case, the client identifies itself to the server by means of a bind DN and a password. The values are sent in the clear over the network (unless SSL is enabled, in which case the information sent over the network can be encrypted, depending on the SSL level specified). The LDAP server verifies that the DN and password sent by the client match the DN and password stored in the LDAP Directory.

The following Oracle Message Broker components use LDAP Authentication when they contact the LDAP Directory:

- The `MsgBroker` command at startup – the Oracle Message Broker uses authentication to validate the user and password for the Oracle Message Broker whenever, during it execution, it accesses the directory.
- JMS Client programs that access the LDAP Directory to obtain Oracle Message Broker administration information (through JNDI) – these programs use authentication to validate the user and password for LDAP Directory access.
- The administration utilities, including `AdminUtil`, `AdminDirCheck`, `ombadmin`, `LDAPSchema`, and `InitDir`. These utilities use authentication to validate the user and password for their LDAP Directory access.

  See the following references for more information on providing options for LDAP Authentication:

`AdminUtil`	"Command-line Administration Utility - AdminUtil" on page 4-41
`AdminDirCheck`	"Checking Directory Entries with AdminDirCheck" on page 4-64
JMS Client Programs	"Enabling Propagation Security" on page 12-27
`InitDir`	*Oracle Message Broker Installation Guide*
`LDAPSchema`	*Oracle Message Broker Installation Guide*
`MsgBroker`	"Starting and Stopping the Oracle Message Broker" on page 2-6
`ombadmin`	"Starting Oracle Message Broker Manager" on page 11-2

LDAP Directory Authorization

Authorization is the process of ensuring that a user reads or updates only the information for which that user has privileges. When directory operations are attempted within a directory session, the directory server ensures that the user-identified by the authorization ID associated with the session has the requisite permissions to perform those operations. Otherwise, the operation is disallowed. Through this mechanism, the directory server enforces authorization policies in order to protect the directory data from unauthorized operations. This mechanism is called access control.

Access Control Information (ACI) is the directory metadata that captures the administrative policies relating to access control.

ACI is stored in the directory as user-modifiable operational attributes. Typically, a list of these ACI attribute values, called an Access Control List (ACL), is associated with directory objects. The attribute values on that list govern the access policies for those directory objects.

ACIs are represented and stored as text strings in the directory. These strings must conform to a well defined format. Each valid value of an ACI attribute represents a distinct access control policy. These individual policy components are referred to as ACI Directives or ACI Items and their format is called the ACI Directive format.

Note: The following information on ACLs is specific to the Oracle Internet Directory implementation of LDAP. This may not apply for other LDAP implementations.

ACLs are stored as special operational attributes of an entry. ACLs are inherited by an entry from its parents and can be overridden.

LDAP Directory Secure Sockets Layer Connections

The Oracle Message Broker supports SSL for connections to the directory from any of its components (for an overview of SSL, see "SSL Overview" on page 12-3). Each of the Oracle Message Broker commands that access the directory support options to specify an SSL connection mode. JMS Clients need to set properties to specify an SSL connection mode. Note that JMS Clients can have two SSL connections, one from the JMS Client to the Oracle Message Broker, and another from the JMS Client to the LDAP Directory. SSL can be enabled for none, one, or both of these JMS Client connections (for details on JMS Client properties, refer to "Oracle Message Broker SSL Options" on page 12-26).

Table 12–2 lists the supported SSL connection modes. In the descriptions in Table 12–2, client refers to an LDAP client.

Note: The Oracle Message Broker also supports a connection mode where SSL is disabled. In this mode, the SSL encryption / decryption and SSL authentication are both disabled.

Table 12–2 SSL Connection Modes

SSL Connection Mode	Description
No authentication	Neither the client nor the LDAP server authenticates itself. No certificates are sent or exchanged. In this case, only SSL encryption / decryption is used. In this configuration, there is no requirement for certificate management.
One-way authentication	Only the LDAP Server authenticates itself to the client. The LDAP Server sends the client a certificate verifying that the server is authentic.
Two-way authentication	Both client and the LDAP Server authenticate themselves to each other. Both the client and the LDAP Server send certificates to each other.

There are performance considerations to keep in mind when using an LDAP Directory to lookup configuration information. When the Oracle Message Broker operates using a remote LDAP Directory, each access to the directory involves network activity. When SSL is enabled, data is encrypted for transmission over the network. This data encryption, and subsequent decryption can have an impact on performance. When SSL security is required, and performance is an issue, an SSL hardware accelerator should be considered.

Security Roles

To limit directory access, and the potential for erroneous or malicious directory modification, different types of Oracle Message Broker users can be set up to have different security roles for accessing the LDAP Directory. Each role should be given a different set of access permissions. The access, and modification of directory entries is enforced by the LDAP Directory's access control mechanism.

Table 12–3 shows the recommended security roles that can be setup in the LDAP Directory to support the Oracle Message Broker.

Table 12–3 Security Roles

Role	Purpose
Administrator	Creates, modifies, and deletes OMB Instances, drivers, queues, topics, servers, and other Oracle Message Broker administrative entries. The Oracle Message Broker administrator needs to have authorization and permissions that allows for adding, deleting, and searching entries underneath the following entry: cn=OMB,cn=Products,cn=OracleContext,....
MsgBroker Instance User	Accesses all the entries underneath a single OMB Instance. A MsgBroker Instance User creates, updates, and deletes entries underneath an OMB Instance. When the Oracle Message Broker is running, any changes that are made to the LDAP Directory within the active OMB Instance using either `AdminUtil` or `ombadmin` are channeled through the Oracle Message Broker. This allows the MsgBroker Instance User to act as the administrator for the active, OMB Instance. MsgBroker Instance User authentication is set with the user DN and password specified when the Oracle Message Broker is started using the `MsgBroker` command. In Local Mode, the MsgBroker Instance User authentication is set in the call that starts the Oracle Message Broker. The MsgBroker's authentication should have access permissions similar to the Oracle Message Broker Administrator, but permissions for a MsgBroker user could be limited to a single OMB Instance.
JMS Client User	A JMS Client user does not need to create, delete, or modify OMB Instance entries. JMS Client users require read and search permissions for an Oracle Message Broker's msg_broker entry within the OMB Instance, and for queues, topics, and connection factories. When the Oracle Message Broker is running in Local Mode, the JMS Client user requires the same permissions as the MsgBroker Instance User.

Protecting Credentials

Oracle Message Broker encrypts all passwords that it stores in the LDAP Directory. The passwords that are encrypted include: AQServerEntry password, Oracle Wallet password, and passwords used for Oracle Message Broker propagation. This allows administrators to disable LDAP authentication/authorization without compromising the passwords used by Oracle Message Broker.

Oracle Message Broker Security

This section covers the Oracle Message Broker security facilities that allow you to protect Oracle Message Broker connections and Oracle Message Broker data. This section covers the following security areas:

- Oracle Message Broker Connections to Directory Server
- JMS Client Connections to Directory Server
- C++ Client Connections to Directory Server
- JMS Client Connections to Oracle Message Broker
- Propagation Security
- Security Service - Application Level Authentication and Authorization

Oracle Message Broker Connections to Directory Server

The Oracle Message Broker and its administration utilities support SSL for their connections with the LDAP Directory Server. See the section, "LDAP Directory Secure Sockets Layer Connections" on page 12-8 for a description of the SSL options available for these directory connections.

JMS Client Connections to Directory Server

JMS Clients support SSL for their connections with the LDAP Directory Server. See the section, "LDAP Directory Secure Sockets Layer Connections" on page 12-8 for a description of the options available for these SSL connections.

C++ Client Connections to Directory Server

C++ Clients support SSL for their connections with the LDAP Directory server. See the section, "LDAP Directory Secure Sockets Layer Connections" on page 12-8 for a description of the options available for these SSL connections.

JMS Client Connections to Oracle Message Broker

When a JMS Client is connecting to the Oracle Message Broker it can use SSL in one of three authentication modes, as shown in Table 12–4.

Table 12–4 JMS Client SSL Connection Modes

SSL Connection Mode	Description
No authentication	Neither the client nor the Oracle Message Broker authenticates itself. No certificates are sent or exchanged. In this case, only SSL encryption / decryption is used. In this case, there is no requirement for certificate management.
One-way authentication	Only the Oracle Message Broker authenticates itself to the client. The Oracle Message Broker sends the client a certificate verifying that the server is authentic.
Two-way authentication	Both client and the Oracle Message Broker authenticate themselves to each other. Both the client and the Oracle Message Broker send certificates to each other.

Note: For one-way or two-way authentication, both the JMS Client and the Oracle Message Broker has to manage certificates using the Oracle Wallet Manager.

Propagation Security

The server-to-server interaction, using Oracle Message Broker propagation, is made secure using IIOP / SSL or HTTP / SSL. A receiving broker authenticates and authorizes a sending broker, if it is configured to do so, while accepting requests for propagation. For details on setting up security for propagation, refer to "Propagation Security" on page 8-23.

Security Service - Application Level Authentication and Authorization

The Oracle Message Broker security service support authentication and authorization of Oracle Message Broker clients. The security service allows the Oracle Message Broker to provide an additional level of control for its operations, including controlling whether a particular user can access and use JMS destinations. This level of security uses the LDAP Directory for obtaining names and passwords and storing authentication and authorization information.

Oracle Message Broker uses the security service to provide a finer grained access control mechanism. The LDAP security provides an all-or-nothing access for Oracle Message Broker users. The security service provides more control. For example, the security service allows Oracle Message Broker to distinguish between subscribing to a topic and publishing to a topic, or sending a message to a queue and receiving a message from a queue.

The security service allows Oracle Message Broker administrator to associate access control lists (ACL)s with destinations. The security service uses the LDAP Directory to store its configuration information. The Oracle Message Broker management tools, `AdminUtil` and `ombadmin`, are available to manage the security service. The security service configuration information consists of users, groups and ACLs. For details on working with these security service components, refer to "Using the Oracle Message Broker Security Service" on page 12-27.

Provider Security

This section describes the Oracle Message Broker provider facilities that allow you to protect messages stored using an Oracle Message Broker driver. Administrators need to manage security features of underlying providers (drivers) using provider specific facilities.

This section covers the following security areas:

- Provider Security Summary
- AQ Driver Security Features
- Provider Security Limitations

Provider Security Summary

The Oracle Message Broker acts a client of the underlying message provider. If the message provider is on a separate machine, the data sent between the Oracle Message Broker and the underlying message provider is sent over the network. Depending on the security requirements, it may be necessary to protect the privacy and integrity of this data. In addition, the underlying message provider may have its own authentication / authorization mechanisms. Table 12–5 shows Oracle Message Broker support for underlying message provider security.

Table 12–5 Message Provider Security in Oracle Message Broker

Provider	Security Available
AQ Driver	Oracle Message Broker uses the username and password specified in the connection factory entry stored in the LDAP Directory. That is the credentials used by the JMS Client to create a JMS connection are used to access AQ. If the username and password are not defined in the connection factory entry then the username and password specified in the JMS call that created the connection are used. The Oracle Message Broker process uses the value of the distinguished attribute as the username. For example, if the username is, cn=bjensen,cn=users then the username used for the AQ connection would be 'bjensen'. Both the AQ OCI driver and AQ JDBC driver can be configured for data privacy and integrity using the Oracle 8*i* advanced security option. This is independent of Oracle Message Broker. See the Oracle 8*i* administration guide for more information.
Oracle Mcast Driver	Does not provide any message provider specific mechanism for authentication / authorization or for data privacy / integrity.
Tibco Driver	Does not provide any message provider specific mechanism for authentication / authorization.
Volatile Driver	Does not provide any message provider specific mechanism for authentication / authorization or for data privacy / integrity.
MQ Series Driver	Authenticates users using the underlying operating system users / groups. The active Oracle Message Broker and MQ Series must run on the same system. In addition, the user running the Oracle Message Broker process should have permissions to access all the MQ Series queues used by the Oracle Message Broker. The JMS Client credentials are not used to access MQ Series.

AQ Driver Security Features

The AQ Driver authenticates Database Server connections with credentials provided by the following choices:

1. Using LDAP Directory Configuration Options - a user name and password can be provided using the administrative facilities. These credentials are used to authenticate the JDBC connections created to support administrative operations. These credentials are also used to authenticate JDBC connections created for each user's JMS session when the user does not specify a user name and password in the connection methods, as shown in option 2 (below). When credentials are stored in the LDAP Directory, the passwords are always encrypted.
2. Invocation Methods in Oracle Message Broker JMS Clients - a user name and password can be supplied when the Oracle Message Broker JMS Clients create a topic connection or a queue connection. These credentials authenticate the JDBC connection when a JMS session is created within the topic or queue connection.

3. Using Lightweight Configuration Properties, the user name and password are used to authenticate Database Server (AQ Driver) connections.

However the credentials are provided, the user name and password may be used to authenticate access to the LDAP Directory, to make access control decisions for the Oracle Message Broker, and to provide access control to the AQ Driver for message store access.

When the Oracle Message Broker is started with lightweight configuration. The user name and password used by the AQ Driver must either be stored in a file or specified on the command line. The Oracle Message Broker administrator should understand the security risks of both options.

AQ security functionality can be configured external to the Oracle Message Broker by modifying the sqlnet.ora and tnsnames.ora file.

Provider Security Limitations

Several Oracle Message Broker drivers have resources that Oracle Message Broker security facilities cannot protect, including the following:

Oracle Mcast Driver – the IP Address and port number

Oracle AQ Driver – Database Server access using PL/SQL, OCI, or JDBC

MQSeries Driver – Native access to MQSeries Queues

Oracle Message Broker security facilities and access control do not and cannot provide access control for the underlying providers, beyond what the providers have in place for access control.

Security Priority

Oracle Message Broker has three, optional levels of authentication/authorization for JMS Client to Oracle Message Broker interactions.

1. LDAP Directory Authentication and Authorization: A JMS Client must be able to read the LDAP Directory to get access to topics, queues, connection factories, or the Oracle Message Broker. All Oracle Message Broker configuration data is stored in an LDAP Directory (unless Oracle Message Broker is run using Lightweight Configuration). A JMS client must be able to read information from the LDAP Server to access destinations managed by the Oracle Message Broker. Access to this information can be controlled through the use of LDAP authentication and authorization mechanism.
2. Oracle Message Broker Security Service ACLs.

3. Provider Access Control. Once the JMS client has access to the information in the LDAP Directory, it can connect to the Oracle Message Broker. If an Oracle Message Broker destination is mapped to a message provider (such as AQ), then operations on the Oracle Message Broker destination get mapped to operations on the message provider destination. The broker uses the client's credentials to request access from the message provider.

The Oracle Message Broker as a whole uses the following order for assuring security:

1. LDAP Directory Level Authentication and Authorization
2. Oracle Message Broker Security Service ACLs
3. Provider-Level Access Control (for example AQ authentication / authorization)

If security access fails for any reason at a higher level, access is completely denied for the lower level.

Network Security Overview

Oracle Message Broker components can use three protocols for communicating between different components: HTTP, LDAP, and IIOP. All these three protocols can be layered over SSL. SSL provides encryption and optional server / client side authentication. SSL provides Oracle Message Broker with privacy of data over the network, data integrity, and certificate based SSL authentication.

Figure 12–2 shows all the components of an Oracle Message Broker deployment.

Figure 12–2 Overview of Network Security

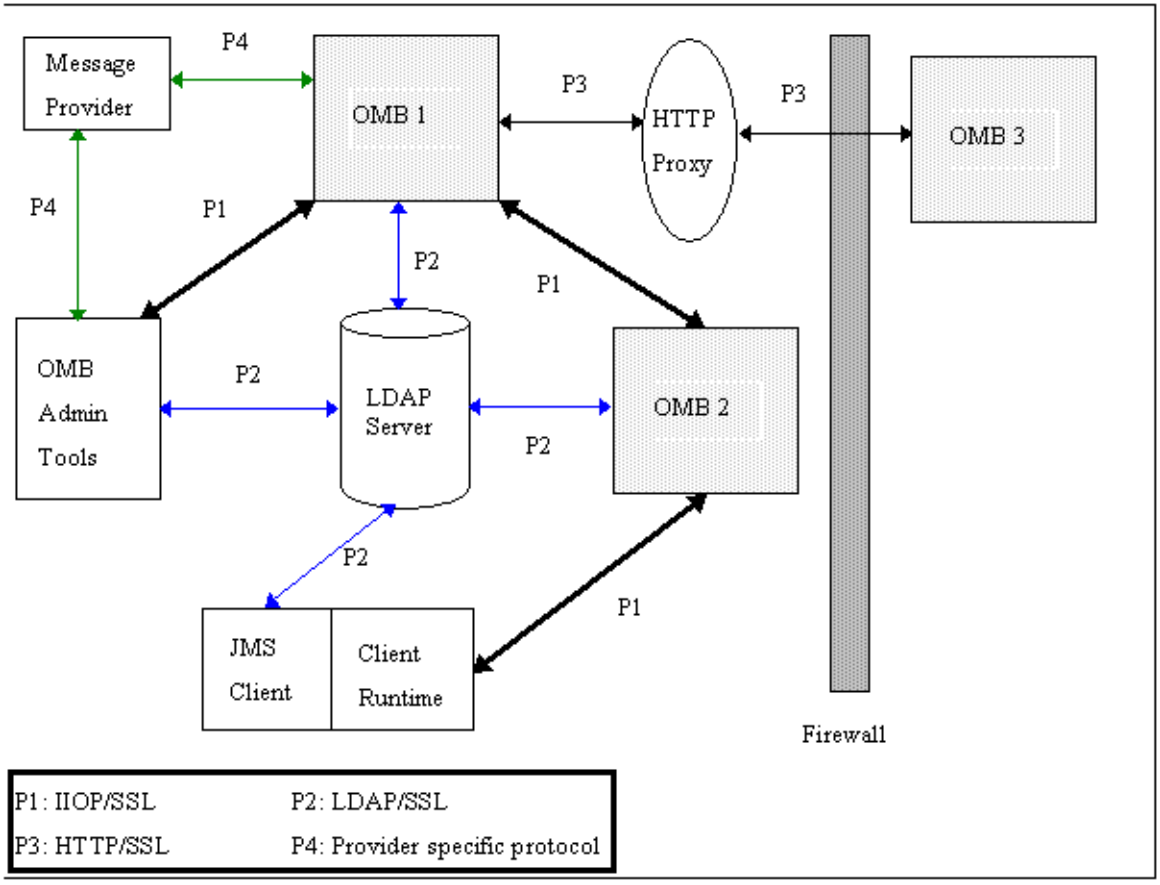


Figure 12–2 shows components linked with the following four labels:

1. P1 - Connections between client side ORBs and server side ORBs. Oracle Message Broker uses the ORB for communicating between the Oracle Message Broker and JMS Clients. The ORB uses IIOP protocol for communication between the client and the server. The ORB allows one to run IIOP over SSL. For more information, refer to "Oracle Message Broker SSL Options" on page 12-26.
2. P2 - Connections between the LDAP Directory Server and LDAP Clients. LDAP clients are any of the following: the Oracle Message Broker, JMS Clients, Oracle Message Broker administration tools. These connections use LDAP over SSL to

the LDAP Directory Server. An SSL socket factory is provided for this purpose. For more information, refer to "Enabling SSL and Authentication for the LDAP Directory" on page 12-20.

3. P3 - Connections between Oracle Message Brokers for propagation using HTTP. Oracle Message Broker supports propagation over HTTP and HTTP over SSL (HTTPS). Oracle Message Broker propagation also supports HTTP proxies. For more information, refer to "Enabling Propagation Security" on page 12-27.
4. P4 - Connections between Oracle Message Brokers and underlying message providers (such as Oracle AQ). Oracle Message Broker supports security provided by the underlying message provider. For more information, refer to "Provider Security Administration" on page 12-33.

Supported Cipher Suites

When you use the Oracle Message Broker, SSL is preconfigured to support a default set of cipher suites. The set of cipher suites supported depends on the SSL level associated with a connection.

A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

To establish an SSL connection between a client and a server, the client and the server must have at least one common cipher suite. During the SSL handshake the client and the server agree on the cipher suite to use for the connection.

The Oracle Message Broker supports prioritized cipher suites as shown in Table 12–6 and Table 12–7. These cipher suites support SSL connections to the LDAP Directory, and for propagation between Oracle Message Brokers using HTTPS.

Table 12–6 Authenticated SSL Connections (SSL level 2 and SSL level 3).

Cipher Suite	Priority Level
SSL_RSA_WITH_3DES_EDE_CBC_SHA	1
SSL_RSA_WITH_RC4_128_SHA	2
SSL_RSA_WITH_RC4_128_MD5	3
SSL_RSA_WITH_DES_CBC_SHA	4
SSL_RSA_EXPORT_WITH_RC4_40_MD5	5

Table 12–6 (Cont.) Authenticated SSL Connections (SSL level 2 and SSL level 3).

Cipher Suite	Priority Level
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	6

Table 12–7 Un-authenticated SSL Connection (SSL Level 1)

Cipher Suite	Priority Level
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	1
SSL_DH_anon_WITH_RC4_128_MD5	2
SSL_DH_anon_WITH_DES_CBC_SHA	3
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	4
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	5

Note: All the cipher suites shown in Table 12–6 and Table 12–7 provide encryption and data integrity.

LDAP Directory Server Security Administration

This section covers the following security administration areas:

- Creating LDAP Users and Working with Access Control Lists
- Enabling SSL and Authentication for the LDAP Directory

Creating LDAP Users and Working with Access Control Lists

To create LDAP users and groups, and to manage LDAP Directory access control lists for the users and the groups, use the administrative tools supplied with your directory, or use the LDAP command line utilities (`ldapadd, ldapmodify`, or other commands that modify directory entries).

Before you create users and groups in your LDAP Directory, refer to the section, "LDAP Server Security" on page 12-5 for a description of the types of users you may want to create (security roles), and the permissions required to control access to the Oracle Message Broker.

Note: This section describes Oracle Message Broker sample scripts that set up LDAP authentication. Oracle Message Broker administrators should modify these scripts, depending on the security policies needed for their LDAP Directory.

Oracle Internet Directory Authentication Sample Scripts

The Oracle Message Broker provides sample ldif scripts for setting up directory authentication in $OMB_HOME/samples/admin/security/ldap/oid (%OMB_HOME%\samples\admin\security\ldap\oid on Windows NT). Refer to the README file for instructions on running the authentication setup scripts.

Netscape Directory Server Authentication Sample Scripts (Windows NT Only)

The Oracle Message Broker provides sample ldif scripts for setting up directory authentication in $OMB_HOME/samples/admin/security/ldap/netscape (%OMB_HOME%\samples\admin\security\ldap\netscape on Windows NT). Refer to the README file for instructions on running the authentication setup scripts.

Enabling SSL and Authentication for the LDAP Directory

When you use Oracle Message Broker components that access the LDAP Directory, they internally set properties that control simple authentication and SSL, either based on the values supplied in prompts to the user, or based on values supplied with command line arguments. JMS Client programs that access the LDAP Directory and connect with the Oracle Message Broker need to explicitly set these properties.

JMS clients use JNDI to access the configuration information in the LDAP Directory, specifically the connection factories and destinations, therefore a JMS client needs to set JNDI properties to their appropriate values to enable authentication and to provide credentials.Table 12–8 shows the Java properties must be set to enable LDAP authentication and SSL for JMS Client connections.

Table 12–8 Oracle Message Broker Client SSL and Authentication Java Properties

Java Property	Description
java.naming.security.authentication	When the authentication property is set to the value “simple”, LDAP authentication is used. If the principal property or the credentials property are not specified, then anonymous directory authentication is used.
java.naming.security.credentials	Use the credentials value to supply a *password*. where: *password* is the password associated with the specified user of the username set in the property java.naming.security.principal.
java.naming.security.principal	Use the principal property to provide a *user_name_dn*. where: *user_name_dn* is the user’s DN in the LDAP Directory. If this is not set an anonymous bind will be used).
java.naming.security.protocol	Set to ‘ssl’ to use SSL connections to the directory.
SSLSocketFactoryImplClass	This property should be set to the following value: oracle.oas.admin.dir.AdminSSLSocketFactoryImpl
oracle.oas.admin.dir.wltloc	Set to the location of the exported wallet that is to be used for SSL authentication. Note: this requires exported wallets.
oracle.oas.admin.dir.wltpassword	Set to the wallet password for the wallet to be used for SSL authentication.

Configuring SSL for OiD

When an instance of OiD is started through the OiD Control Utility, a configuration set entry can be specified. This configuration set entry determines the SSL configuration for that instance of OiD. A configuration set entry can be created / modified / deleted using either Oracle Directory Manager or command line tools. When using Oracle Directory Manager, expand the ‘Server Management-->Directory Server’ entry (this is for non-replicated directory servers, for replication refer to the OiD Administration Manual). This will list all the configuration set entries for the directory server. Add / delete / update operations can be performed on configuration set entries using Oracle Directory Manager (for details on configuration sets refer to the OiD Administration Guide).

Configuration Set entries for LDAP server are stored under the container, cn=osdldapd,cn=subconfigsubentry

The config set entries should belong to the objectclass ‘orclConfigSet’ and ‘orclLDAPSubConfig’, the attribute ‘orclsslenable’ sets (value of 1) or unsets (value of 0) SSL. The attribute ‘orclsslauthentication’ sets the authentication level for SSL

(value of 1 => no authentication, value of 32 => server side authentication and value of 64 => server and client side authentication). The attribute 'orclsslwalleturl' points to the wallet location and the attribute 'orclsslwalletpasswd' contains the value of the wallet password.

Note: Default ACLs in OiD do not protect the subtree cn=subconfigsubentry for read operations. Since this subtree can possibly contain wallet passwords, read operations on this subtree must be protected.

(for replication configuration set entries refer to the OiD Administration Guide)

Sample configuration set entry for SSL without any authentication:

```
dn: cn=configset1, cn=osdldapd, cn=subconfigsubentry
objectclass: orclConfigSet
objectclass: orclLDAPSubConfig
objectclass: top
cn: configset1
orcldebuglevel: 0
orclsslenable: 1
orclsslport: 636
orclserverprocs: 1
orclsslauthentication: 1
orclmaxcc: 10
```

Sample configuration set entry for SSL with server side authentication:

```
dn: cn=configset2, cn=osdldapd, cn=subconfigsubentry
objectclass: top
objectclass: orclConfigSet
objectclass: orclLDAPSubConfig
orclmaxcc: 10
orcldebuglevel: 0
cn: configset2
orclsslauthentication: 32
orclsslenable: 1
orclsslport: 6666
orclsslwalletpasswd: welcome88
orclsslwalleturl: file:/private/oracle/product/oracle/8.1.6/wallets/test_wallet1
orclserverprocs: 1
```

Sample configuration set entry for SSL with server side and client side authentication:

```
dn: cn=configset3, cn=osdldapd, cn=subconfigsubentry
orclsslport: 7777
```

```
objectclass: orclConfigSet
objectclass: top
objectclass: orclLDAPSubConfig
cn: configset3
orcldebuglevel: 0
orclmaxcc: 10
orclsslauthentication: 64
orclserverprocs: 1
orclsslenable: 1
orclsslwalleturl: file:/private/oracle/product/oracle/8.1.6/wallets/test_wallet1
orclsslwalletpasswd: welcome88
```

OiD Access Control and Authorization

Authorization is the process of ensuring that a user reads or updates only the information for which that user has privileges. When directory operations are attempted within a directory session, the directory server ensures that the user, identified by the authorization ID associated with the session, has the requisite permissions to perform those operations. Otherwise, the operation is disallowed. Through this mechanism, the directory server protects directory data from unauthorized operations by directory users. This mechanism is called access control. Access Control Information (ACI) is the directory meta data that captures the administrative policies relating to access control. ACI is stored in Oracle Internet Directory as user-modifiable operational attributes. Typically, a list of these ACI attribute values, called an Access Control List (ACL), is associated with directory objects. The attribute values on that list govern the access policies for those directory objects. ACIs are represented and stored as text strings in the directory. These strings must conform to a well defined format. Each valid value of an ACI attribute represents a distinct access control policy. These individual policy components are referred to as ACI Directives or ACI Items and their format is called the ACI Directive format.

orclACI is a user modifiable optional attribute that represents Access Control List (ACL) policy information for a subtree starting with the entry where that subtree is defined. Any entry in the directory can contain values for this attribute. Access to this attribute itself can be controlled the same way access to any other attributes is controlled. Access Control Policy Points (ACPs) are entries in which the orclACI attribute value is set. The orclACI attribute value represents the access policies inherited by a subtree of entries starting with the ACP as the root of the subtree. When a hierarchy of multiple ACPs exists in a directory subtree, the subordinate entries in that subtree inherit the access policies from all of the ACPs that are superior to the entry. The resulting policy is an aggregation of the policies within

the ACP hierarchy above the entry. When there are conflicting policies represented among a hierarchy of ACPs, the directory applies well defined precedence rules while evaluating the aggregate policy.

The orclACI attribute contains ACL directives that are prescriptive in nature, that is, the directives apply to all entries in the subtree below the ACP where this attribute is defined. In addition, it is often convenient to maintain ACL directives specific to a single entry within that entry. Oracle Internet Directory enables you to do this through a user modifiable operational attribute called orclEntryLevelACI. Any directory entry can optionally carry a value for this attribute. This is accomplished in the Oracle Internet Directory base schema by extending the abstract class top to include orclEntryLevelACI as an optional attribute. The orclEntryLevelACI attribute is multivalued and has a structure similar to that of orclACI.

Access Control Information associated with a directory object represents the permissions on the given object that various directory user entities (or subjects) have. Thus, semantically, an ACI is a tuple consisting of three components: Object (to what are you granting access?), Subject (to whom are you granting access, Operations (what access are you granting?).

Example of ACIs set on the container cn=omb:

```
dn: cn=OMB,cn=Products,cn=OracleContext,ou=asg,o=oracle,c=us
objectclass: top
objectclass: orcloasentry
objectclass: orclcontainer
cn: OMB
orclaci: access to attr=(*) by * (search , read, nowrite, compare) by group=
“cn=ombdev,cn=groups” (search, read, write, compare)
orclaci: access to entry by * (browse, noadd, nodelete) by group=
“cn=ombdev,cn=groups” (browse, add, delete)
orcloasentrytype: omb_container
```

Creating users and groups in OiD

A user of OiD (other than cn=orcladmin, cn=guest and cn=proxy) is the DN of the entry in the directory. This entry should have the attribute, ‘userpassword’. For example,

```
dn: cn=bjensen,cn=users
objectclass: top
objectclass: person
cn: bjenson
sn: Jensen
```

```
userpassword: welcome
```

The username is “cn=bjensen,cn=users” and the password is “welcome”.

Group entries in Oracle Internet Directory are associated with either the ‘groupOfNames’ or the ‘groupOfUniqueNames’ objectclass. Membership in the group is specified as a value of the ‘member’ or ‘uniqueMember’ attribute respectively. It is possible to specify access rights for a group of people or entities. Such groups are called privilege groups and are associated with the ‘orclPrivilegeGroup’ object class. To grant access rights to a group of users, one has to create a group entry in the usual way, then associate it with the ‘orclPrivilegeGroup’ object class. The access policies applicable to that group are then specified. Entries can have either direct memberships to groups, or indirect memberships to other groups by means of nested groups, thus forming a forest of privilege groups.

Access policies specified at a given level are applicable to all the members directly or indirectly below it. Because Oracle Internet Directory evaluates for access control purposes only groups marked as privilege groups, it does not allow setting access policies for non-privilege groups. When a user binds with a specific DN, Oracle Internet Directory computes his or her direct membership in privilege groups. Once it knows the first level groups for the given DN, Oracle Internet Directory computes nesting of all these first level groups into other privilege groups. This process continues until there are no more nested groups to be evaluated. It is imperative that all groups created for access control purposes, nested or otherwise, be marked as privilege groups by associating them with the ‘orclPrivilegeGroup’ object class. A normal group will not be considered for access control purposes even though it may be a member of a privilege group. Example of a group,

```
dn: cn=ombdev,cn=groups
objectclass: top
objectclass: groupofnames
objectclass: orclprivilegegroup
cn: ombdev
member: cn=bjensen,cn=users
member: cn=akarmark,cn=users
```

Oracle Message Broker Security Administration

This section covers the following security administration areas:

- Oracle Message Broker SSL Options
- Enabling Propagation Security
- Using the Oracle Message Broker Security Service

Oracle Message Broker SSL Options

The Oracle Message Broker commands, including the following commands support a set of options to control SSL: `AdminUtil`, `AdminDirCheck`, `MsgBroker`, `LDAPSchema`, and `InitDir`. Table 12–9 describes the command line options that control SSL.

If the -U2 or -U3 option is used with the commands, and the -W or -P is not used, then the Oracle Message Broker commands prompt for the SSL wallet directory, the SSL wallet password, or for both.

Note: These options specify SSL connections to the LDAP Directory.

Table 12–9 SSL Command Line Options

Option	Description
-U *value*	Specifies if SSL is used, and the authentication level. Valid *values* are: 0, 1, 2, and 3. `0` – no SSL. This is the default if –U is not specified. `1` – SSL with no authentication. `2` – SSL with server-side authentication. `3` – SSL with server-side and client-side authentication.
–W *wallet_path*	Specifies the path to an exported wallet file. This is ignored if the value of –U is 0 or 1.
–P *wallet_password*	Specifies the wallet password. This is ignored if the value of –U is 0 or 1.

Graphical Command SSL Options

When ombadmin starts, you need to enter an SSL level. Table 12–10 shows the SSL levels supported for `ombadmin`.

Table 12–10 SSL Levels for ombadmin

SSL Level	Description
none	No SSL
noauth	SSL with no authentication
server auth	SSL with server-side authentication
server and client auth	SSL with server-side and client-side authentication

Enabling Propagation Security

See the section, "Propagation Security" on page 8-23 for information on propagation security.

Using the Oracle Message Broker Security Service

The Oracle Message Broker security service provides authentication and authorization to control whether a user can access JMS destinations. This level of security uses the LDAP Directory to obtain names and passwords and for storing authentication and authorization information.

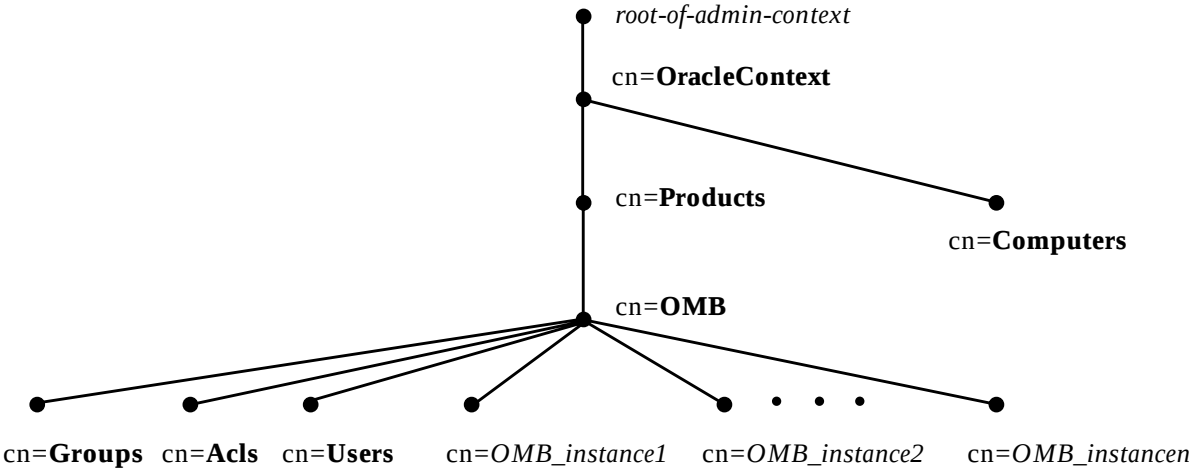
This section covers the following:

- Managing Users
- Managing Groups
- Managing Access Control Lists (ACL)s
- Security Service Cache
- Netscape Directory Password Encryption
- Client Connections using Authentication and Authorization

The security service allows the Oracle Message Broker administrator to associate access control lists (ACL)s with destinations. The Oracle Message Broker management tools, `AdminUtil` and `ombadmin`, are available to manage the security service. The security service configuration information consists of users, groups and ACLs that are stored in the Users, Groups, and ACLs fixed name containers (see Figure 12–3).

The users, groups, and ACLs defined are not tied to a specific OMB Instance. They can be used across all OMB Instances defines within the LDAP Directory.

Figure 12–3 Top Level Directory Organization with Users, Groups, and ACLs Containers



Managing Users

A user for the Oracle Message Broker security service consists of a DN, which represents a user entry. Authentication consists of looking up the DN and matching the `password` attribute.

Since the user entry is a DN, standard LDAP user entries and the Oracle Message Broker security service user entries are both valid user entries. The security service has the same requirements for a user entry as an LDAP server. Using LDAP server user entries as security service user entries can reduce the number of administration tasks.

An LDAP user can be used as an Oracle Message Broker user. However, the `Oracle Message Broker` tools, `AdminUtil` and `ombadmin` can only modify Oracle Message Broker user entries (these are defined as user entries under the `cn=users` container in the OMB entry in the LDAP Directory).

Table 12–11 shows the user entry attributes. To configure a user entry, create the user entry and then set the appropriate attributes.

Table 12–11 User Attributes

Attribute	Description
`description`	A description of the user.
`password`	Password associated with the user. This value is encrypted. This attribute is mapped to the LDAP attribute `userpassword`. The LDAP server controls whether this attribute is encrypted.
`surname`	The user's surname.

Managing Groups

A group entry specifies an Oracle Message Broker group. Table 12–12 shows the group entry attributes. To configure a group, create the group entry and then define the appropriate attributes.

Oracle Message Broker group entries are not always LDAP group entries. However, LDAP group entries are valid to use as Oracle Message Broker group entries (when using either LDAP as provided by Oracle Internet Directory or Nestscape Directory Server).

Table 12–12 Group Attributes

Attribute	Description
`description`	A description of the group.
`uniquemember`	This is a DN that points to a user entry, or to multiple user entries. The value can also point to other groups. A group cannot have nested subgroups that include more than 10 levels of nesting.

Managing Access Control Lists (ACL)s

Destinations have an optional attribute called `acl_dn` (see Table 4–17 and Table 4–18 for destination attributes). The `acl_dn` value is a DN which points to an LDAP ACL entry containing security service configuration information. Oracle Message Broker security service authorization is based on checking the ACL associated with the destination (to which access is requested).

ACL entries are stored in the ACLs container in the LDAP Directory.

Table 12–13 shows the ACL entry attributes. To configure an ACL entry, create the entry and then define the access control list.

Table 12–13 Access Control List (ACL) Attributes

Attribute	Description
`aclEntry`	Defines the access control list, with a value of the form DN=*num* (see the explanation below for details).

The value for an `aclEntry` is of the form:

DN=num

Where:

DN is the DN for a group, a user, or is the string 'anonymous'.

num is the permission associated with the user or group, or for the anonymous user.

Table 12–14 shows the possible values for the aclEntry permission, *num*.

Table 12–14 ACL Entry Permissions

Value	Permissions
1	Read access. For topics, read access implies granting subscription rights. For queues, read access implies granting receive rights.
2	Write access. For topics, write access implies granting publication rights.For queues, write access implies granting send rights.
3	Read and write access.

If an ACL is not specified for a destination, topic or queue, the destination is open, and anyone can subscribe / publish or send / receive messages for that destination.

If an ACL is specified, but no users are given permissions in the ACL, then no one can subscribe / publish or send / receive messages for that destination. This means permissions in an ACL have to be explicitly granted. For example, if permission for a user cn=foo, or a group that user cn=foo belongs to, is not granted in the ACL, it implies that the permission is denied for that user (also see the description of the anonymous ACL for information on granting permissions to everyone).

For example, an `aclEntry` could have the following value:

```
cn=group1,cn=Groups,cn=OMB,cn=Products,cn=OracleContext,ou=name1,o=acme,c=us=3
```

This specifies that all the users in the group defined by group1 have read and write access to the destination.

If an `aclEntry` is specified, and the DN value is specified as the string, "anonymous", then everyone, that is any user, is given the permissions specified for the anonymous `aclEntry`. When a user attempts to use the destination, and either a user name is supplied, or a user name is not supplied, the user is granted the permissions to the destination that are associated with anonymous. In this case, permissions in an `aclEntry` do not have to be explicitly granted by user or group.

For example, an aclEntry could have the following value:

```
anonymous=1
```

For this example, the permission for anonymous is set to 1, which grants receive or subscribe permission for all of the destination's users.

Security Service Cache

The Oracle Message Broker caches security service related LDAP data. This cache is per JMS connection. The Java property oracle.oas.mercury.sec.cache.expiration specifies the cache expiration time in milliseconds.

Table 12–15 Security Service Java Properties

Java Property	Description
oracle.oas.mercury.sec.cache.expiration	Specifies the cache expiration time in milliseconds. If the value < 0, then the cache never expires. The default value is -1

Netscape Directory Password Encryption

The Netscape directory server allows passwords to be encrypted, using SHA or unix crypt, or passwords may be unencrypted. If the passwords are encrypted, the security service needs to perform a heavy-weight ldap_bind operation (and create a new tcp / IP connection) to verify passwords.

If Netscape directory server is used with password encryption, the default setting, then the property oracle.oas.security.nocompare should be set to `true`. If this property is not set for Netscape directory server, with password encryption enabled, then the Oracle Message Broker always throws a SecException.

Client Connections using Authentication and Authorization

To access the Oracle Message Broker, a JMS client has to create a JMS connection to the Oracle Message Broker.

There are four JMS APIs to do this:

1. To create an anonymous topic connection to the Oracle Message Broker, use:

   ```
   TopicConnectionFactory.createTopicConnection()
   ```

2. To create an anonymous queue connection to the Oracle Message Broker, use:

   ```
   QueueConnectionFactory.createQueueConnection()
   ```

3. To create a non-anonymous topic connection to the Oracle Message Broker, use:

   ```
   TopicConnectionFactory.createTopicConnection(String username, String
   password)
   ```

4. To create a non-anonymous queue connection to the Oracle Message Broker, use:

   ```
   QueueConnectionFactory.createQueueConnection(String username, String
   password)
   ```

If TopicConnectionFactory.createTopicConnection() or QueueConnectionFactory.createQueueConnection() are used, then the connections are treated as anonymous.

The username provided in (2) and (4) above should be a DN of a security service user entry or an LDAP user entry and the password must be the password for the specified user.

Oracle Message Broker uses the security service to authenticate a user. On a successful authentication, the Oracle Message Broker gets a 'ticket' from the security service for the specified credentials. The credentials are cached on the client side runtime and the Oracle Message Broker. The Oracle Message Broker also caches the ticket. The client side runtime sends these credentials to the Oracle Message Broker when requesting any operation using this connection. The Oracle Message Broker authenticates the credentials in memory, without using the security service, and then uses the security service to authorizes the client request using the cached ticket.

JMS client credentials are per connection. A new connection should be created for different credentials. The Oracle Message Broker also uses the same credentials when it calls the client-side callbacks.

Any authentication or authorization error results in a JMSSecurityException being thrown.

To use SSL connections to the directory an SSL socket factory is provided. For details on the properties to set to use LDAP authentication / authorization and SSL connections to the directory refer to "Enabling SSL and Authentication for the LDAP Directory" on page 12-20.

Caching of tickets results in a performance improvement, as this avoids a round trip to the LDAP server. ACL information is not cached, so any change in the ACL associated with a destination will be seen by the Oracle Message Broker immediately.

Note: Since the credentials (username / password) used by security service are stored in the directory credentials used for directory authentication can also be used for security service (in methods (2) and (4) above).

Provider Security Administration

This section covers the following security administration areas:

- Client Connections to the Oracle Message Broker using Authentication

Client Connections to the Oracle Message Broker using Authentication

To access the Oracle Message Broker, a JMS client has to create a JMS connection to the Oracle Message Broker.

There are four JMS APIs to do this:

1. To create an anonymous topic connection to the Oracle Message Broker, use:

   ```
   TopicConnectionFactory.createTopicConnection()
   ```

2. To create an anonymous queue connection to the Oracle Message Broker, use:

   ```
   QueueConnectionFactory.createQueueConnection()
   ```

3. To create a non-anonymous topic connection to the Oracle Message Broker, use:

   ```
   TopicConnectionFactory.createTopicConnection(String username, String password)
   ```

4. To create a non-anonymous queue connection to the Oracle Message Broker, use:

   ```
   QueueConnectionFactory.createQueueConnection(String username, String password)
   ```

The Oracle Message Broker passes the username and password specified in the non-anonymous methods to the Driver to authenticate the JMS client.

These credentials are cached by the client-side runtime and reused for all subsequent calls made on the connection.

If TopicConnectionFactory.createTopicConnection() or QueueConnectionFactory.createQueueConnection() are used, then the connections are treated as anonymous.

JMS client credentials are per connection. A new connection should be created for different credentials. The Oracle Message Broker also uses the same credentials when it calls the client-side callbacks.

Any authentication or authorization error results in a JMSSecurityException being thrown.

To use SSL connections to the directory an SSL socket factory is provided. For details on the properties to set to use LDAP authentication / authorization and SSL connections to the directory refer to "Enabling SSL and Authentication for the LDAP Directory" on page 12-20.

13

Lightweight Configuration

The Oracle Message Broker supports two means of specifying configuration information for administration: using an LDAP Directory, and using Lightweight Configuration. With Lightweight Configuration, the Oracle Message Broker reads configuration information from a file or from Java properties when it begins its execution. The Oracle Message Broker configuration information specifies the names and configuration options for all administrative objects, such as the names and types of JMS destinations (topics or queues).

Lightweight configuration supports a variant called Zero Configuration. Zero Configuration is only supported for the AQ Driver. Zero Configuration allows the Oracle Message Broker to start with the AQ Driver as easily as a JDBC connection starts. The only configuration information required for Zero Configuration is the AQ Database Server service name, user name, and password. See "Obtaining a Connection Factory Instance (in Local Mode)" on page 13-9 for more information on Zero Configuration.

This chapter describes Oracle Message Broker Lightweight Configuration. Refer to Chapter 4, "Administration" for a description of configuration using the Oracle Message Broker with an LDAP Directory to specify configuration information.

This chapter includes the following sections:

Benefits of Lightweight Configuration

Lightweight configuration allows the Oracle Message Broker to operate without using an LDAP Directory to store its administrative information. The benefits of Lightweight Configuration are:

- A smaller footprint - Use of the directory requires an installation of Oracle8*i*. If Oracle8*i* is installed on the local system, it requires significant disk resources.
- Increased availability -Using Lightweight Configuration can improve the Oracle Message Broker availability for Oracle Message Broker clients. When Oracle Message Broker clients access an Oracle Message Broker as a separate process, in Remote Mode, the Oracle Message Broker is available when both the directory server and the Oracle Message Broker process are available. When Oracle Message Broker clients use the Oracle Message Broker locally, the Oracle Message Broker is available when the directory server process is available. For both Local Mode and Remote Mode, using the Lightweight Configuration removes the requirement that the directory server process be available.
- Simplicity - installing, configuring, and running the Oracle Message Broker are less complex tasks using the Lightweight Configuration. Administration and configuration issues are simplified and the Database Server performance tuning task is eliminated (using an LDAP Directory Server, the associated Database Server may require tuning).
- Reduced startup time - the start up time, including the time required to access and process configuration information for the Oracle Message Broker is reduced using Lightweight Configuration.
- Lightweight configuration requires less overall system memory, as compared to a single system that is supporting both the Oracle Message Broker and its associated directory server.

Using Lightweight Configuration

To use Lightweight Configuration with Oracle Message Broker, application developers or administrators controlling the system need to set configuration options to control the system. The configuration information only needs to be updated when the system configuration needs to change. The reasons for a configuration change include:

- Adding or removing queues or topics.
- Adding or removing a durable subscriber.
- Modifying Oracle Message Broker resources (for example memory options).

This section describes the following:

Configuration Changes

With Lightweight Configuration, if the Oracle Message Broker is running in Remote Mode, it must be shutdown and then restarted before configuration changes take effect. Dynamic updates of Oracle Message Broker configuration information are not supported using Lightweight Configuration.

With Lightweight Configuration, if the Oracle Message Broker is running in Local Mode, the client application that instantiates the Oracle Message Broker needs to stop and then restart, or the client application needs to close and restart the Oracle Message Broker, before configuration changes take effect. Refer to Chapter 5, "Oracle Message Broker Features" for information on Oracle Message Broker Local Mode operation.

Starting with Lightweight Configuration in Remote Mode

In Remote Mode, start the Oracle Message Broker using Lightweight Configuration, with the command:

```
% java HEAP [CONFIG] [OPT] oracle.oas.mercury.Mercury -noldap
```

Where the parameters are:

HEAP	`-mx`*`size`*`m -ms`*`size`*`m -Doracle.oas.mercury.maxheap=`*`size`* where *size* is: an integer, number of megabytes for JVM heap. For Java 1.2, use: `-Xmx`*`size`*`m -Xms`*`size`*`m` `-Doracle.oas.mercury.maxheap=`*`size`* The *HEAP* parameters are required.
CONFIG	`-Doracle.oas.mercury.config=`*`FILE`* Where *FILE* is: a string, representing the name of file in which configuration information is stored. The *CONFIG* file parameter is optional.
OPT	Other properties that specify configuration parameters.
–noldap	Specifies Oracle Message Broker Lightweight Configuration.

For example:

```
% java -mx10m -ms10m -Doracle.oas.mercury.maxheap=10 \
    -Doracle.oas.mercury.config=config.pr oracle.oas.mercury.Mercury -noldap
```

For this example, the file config.pr must be present in the current working directory.

Starting with Lightweight Configuration in Local Mode

When starting a client that uses a local Oracle Message Broker, start the Java interpreter with the following flags:

–mx*mem*m (for Java 1.1.x)

–Xmx*mem*m (for Java 1.2)

–ms*mem*m (for Java 1.1.x)

–Xms*mem*m (for Java 1.2

–Doracle.oas.mercury.maxheap=*mem*

Where *mem* is the memory, in megabytes, that you want to allocate to the process. The *mem* value should be at least 8, specifying 8 megabytes per instance. If `max_memory` is set in the `msg_broker` entry, the *mem* value, as shown above, must be greater than or equal to the value of `max_memory`.

Deploying Using Lightweight Configuration

Using Lightweight Configuration, there are two deployment options for an Oracle Message Broker application:

1. Deploying with Lightweight Configuration in Remote Mode
2. Deploying with Lightweight Configuration in Local Mode

For each deployment option, an Oracle Message Broker application must perform the following tasks:

1. Obtain a connection factory instance.
2. Obtain a destination instance for a topic or a queue.
3. Cleanup and shutdown.

This section covers these steps for both Local Mode and Remote Mode Oracle Message Brokers.

Obtaining a ConnectionFactory Instance (in Remote Mode)

When the Oracle Message Broker, using Lightweight Configuration, runs in Remote Mode it writes its IOR as a UTF string to a file. The name of the IOR file is JMSProvider*key*, where the property oracle.oas.mercury.key specifies the value for the *key* (see Table 13–3 for more details). Oracle Message Broker writes the JMSProvider*key* file containing the IOR in the directory where the Oracle Message Broker runs.

In Remote Mode, Oracle Message Broker clients access the IOR stored in the JMSProvider file to obtain a connection factory instance. The Oracle Message Broker provides Mercury package method to access the IOR for this purpose.

The method Mercury.getRemoteConnectionFactory creates an Oracle Message Broker MercuryConnectionFactory for a given IOR file. The MercuryConnectionFactory can be cast to either a TopicConnectionFactory or a QueueConnectionFactory.

The method Mercury.getRemoteConnectionFactory creates a connection factory that returns connections to a Remote Mode Oracle Message Broker. This method is

usually used to establish connections to a Remote Mode Oracle Message Broker that has been configured using Lightweight Configuration.

Obtaining Remote Connection Factory Instance Given the Oracle Message Broker JMSProvider file, the API for the JMS getRemoteConnectionFactory access method is:

```
package oracle.oas.mercury.Mercury;
import javax.jms.*;

class Mercury {
public static MercuryConnectionFactory
             getRemoteConnectionFactory(String iorFile,String driverName,
                                        String clientId)
throws JMSException
}
```

Where Table 13–1 shows the parameters for the getRemoteConnectionFactory method.

The method getRemoteConnectionFactory returns a MercuryConnectionFactory that can be used to create instances of javax.jms.QueueConnection and javax.jms.TopicConnection.

It throws a JMSException if the IOR is invalid or the ORB cannot be started.

Table 13–1 Connection Factory Access Method Parameters

Parameter	Description
ior_file	The name of the file in which the Oracle Message Broker IOR is stored. This file should only contain the string that represents the IOR. An Oracle Message Broker that is started with Lightweight Configuration writes the IOR as a UTF string into JMSProvider*key* file. Oracle Message Broker clients should use this file to connect to the Oracle Message Broker when running in Remote Mode. When the Oracle Message Broker is running in Local Mode, the client does not need to read the IOR from a file.
driver_name	This is one of the following: ‘aq’, ‘mq’, ‘vol’, ‘mcast’, ‘rv’. This specifies the name of the driver for which connections are created. A connection factory is limited to creating connections for one driver type.
client_id	This specifies the value to use for the connection id. Each JMS connection can have a connection id set using a call to javax.jms.Connection.setClientID(String). The value specified by the client_ id parameter is used to implicitly call setClientID() if *client_id* is not null.

Obtaining Destinations

To obtain destinations, the Oracle Message Broker client can use created or configured destinations. There are two types of destinations, created destinations and configured destinations. **Created destinations** are destinations for which an Oracle Message Broker client provides all the configuration information when the client creates an instance of the created destination Table 13–2 lists the methods that support created destinations (these are defined in the package oracle.oas.mercury.MercurySession). These methods allow client programs to locate and use topics and queues with Lightweight configuration (without JNDI). **Configured destinations** are destinations for which the Oracle Message Broker has been configured. Part of the configuration information is the name by which the broker knows the destination. This name is not necessarily the same as the name by which the message store identifies the destination (refer to Table 13–6 for more information on destination properties).

Using Created Destinations Oracle Message Broker calls the destinations returned from the methods shown in Table 13–2 created destinations.

The methods in Table 13–2 are equivalent to `TopicSession.createTopic` and `QueueSession.createQueue` found in `javax.jms`. The methods in Table 13–2 create an instance of `javax.jms.Destination`.

Table 13–2 shows the methods for creating destinations, topics or queues, using Lightweight Configuration. These methods are equivalent to createTopic and createQueue, but they do not require a specially formatted string to identify the destination.

Table 13–2 Created Destination Methods for Topics and Queues

Method	**Description**
`createAQTopic`	Locate and create an AQ topic
`createAQQueue`	Locate and create an AQ queue
`createVolatileTopic`	Locate and create an Volatile topic
`createVolatileQueue`	Locate and create an Volatile queue
`createMQQueue`	Locate and create an MQSeries queue

Note: The methods shown in Table 13–2 can be used regardless of the technique used to configure the Oracle Message Broker, either an LDAP Directory or Lightweight Configuration.

When the Oracle Message Broker is started using an LDAP Directory in either Local Mode or Remote Mode, created destinations can only be used as the argument in a call to send or publish. They cannot be used to create a message producer or message consumer.

Configured Destinations Configured destinations are destinations that are returned using a JNDI lookup or with a call to the `MercuryQueue` or `MercuryTopic` constructors shown below. To obtain configured destinations, instances of javax.jms.Topic or javax.jms.Queue, when an LDAP Directory is not used, use one of the following constructors:

```
oracle.oas.mercury.MercuryQueue(String queue_name)
oracle.oas.mercury.MercuryTopic(String topic_name)
```

Where:

queue_name	is the name of the queue
topic_name	is the name of the topic

To use these constructors, Oracle Message Broker needs to be using Lightweight Configuration.

Shutting Down

When a client is done with the Oracle Message Broker, it should call `shutdownClient`. This shuts down the ORB and performs certain cleanup operations. Clients call this method as follows:

```
oracle.oas.mercury.Mercury.shutdownClient()
```

> **Note:** The shutdownClient method is not part of the JMS specification. It assists the Oracle Message Broker in cleanup, and allow clients to exit and cleanup.

After the Oracle Message Broker is shut down, keep the following in mind:

- The ORB cannot be restarted after it is shut down. Thus a client should not call `shutdownClient` and then attempt to access a remote Oracle Message Broker by obtaining a connection factory.

Obtaining a Connection Factory Instance (in Local Mode)

When the Oracle Message Broker, using Lightweight Configuration and running in Local Mode, obtain a ConnectionFactory Instance for the Oracle Message Broker by calling one of the getLocalConnectionFactory methods:

- oracle.oas.mercury.Mercury.getLocalConnectionFactory()
- oracle.oas.mercury.Mercury.getLocalAQConnectionFactory()

Using getLocalConnectionFactory The method getLocalConnectionFactory starts an Oracle Message Broker that uses any supported driver. The method getLocalConnectionFactory requires configuration properties for an Oracle Message Broker to be added to System properties. Configuration properties can also be loaded from a file when using Mercury.getLocalConnectionFactory.

Call the following method to start a local Oracle Message Broker using configuration information specified with Lightweight Configuration:

```
package oracle.oas.mercury;
import javax.jms.*;

Mercury.getLocalConnectionFactory(String driver_name, String broker_id,
                                  String client_id);
```

Where the arguments are:

driver_name	the driver name
broker_id	the broker ID
client_id	the client ID

Use this method when you want to start the Oracle Message Broker in local mode using Lightweight Configuration and you have either stored the configuration properties in a file or inserted them into the System properties.

The `getLocalConnectionFactory` method starts a local Oracle Message Broker without an LDAP Directory and without using JNDI. When a client executes this method it creates an active OMB Instance in the client's JVM.

Using getLocalAQConnectionFactory The method getLocalAQConnectionFactory starts an Oracle Message Broker that uses the JDBC based AQ driver. This method does not require any configuration properties set in System properties and it does not load configuration properties from a configuration file. This variant of lightweight configuration is called Zero Configuration. Zero Configuration is only supported

for the AQ Driver. Zero Configuration allows the Oracle Message Broker to start with the AQ Driver as easily as a JDBC connection starts.

This method allows the Oracle Message Broker to start with no Oracle Message Broker-specific configuration. The information required to use start the Oracle Message Broker is: service_name, admin_user, admin_password. If properties are set in System properties, the Oracle Message Broker started using getLocalAQConnectionFactory can start other Oracle Message Broker drivers, in addition to the AQ Driver.

The API for the getLocalAQConnectionFactory access method is:

```
package oracle.oas.mercury;
import javax.jms.*;

Mercury.getLocalAQConnectionFactory(String service_name, String admin_user,
                                    String admin_password, String client_id);
throws JMSException;
};
```

Where:

service_name	is the name of the net service name used to create JDBC connections
admin_user	is the name of the Database Server user
admin_password	is the password for the Database Server user
client_id	specifies the client identifier to use for the connection

Obtaining Destinations (in Local Mode)

The methods to obtain destinations in Local Mode are the same as those available in Remote Mode. Refer to "Obtaining Destinations" on page 13-7 for details.

Shutting Down (in Local Mode)

A client program running Oracle Message Broker in Local Mode shuts down the Oracle Message Broker when it finishes. The close() method associated with a MercuryConnectionFactory initiates the shutdown for the Local Mode Oracle Message Broker.

For example, local clients can call the following method to stop Oracle Message Broker:

```
oracle.oas.mercury.MercuryConnectionFactory.close()
```

Where `MercuryConnectionFactory` is a connection factory instance, as shown in "Obtaining a Connection Factory Instance (in Local Mode)" on page 13-9. This method initiates the shutdown for the Oracle Message Broker.

The client program can also shut down the Oracle Message Broker using the method Mercury.shutdownClient, as shown in "Shutting Down" on page 13-8.

Specifying Configuration Values with Lightweight Configuration

Lightweight configuration uses Java properties to specify configuration values. Properties are specified on the command line or in a file (or both, see "Property Evaluation Precedence Rules" on page 13-12). The file containing Oracle Message Broker configuration properties is specified on the command line using the property oracle.oas.mercury.config (see Table 13–3 for details on this property). The Oracle Message Broker reads this file once, when it starts.

Specifying Properties

The syntax for specifying properties on the command line is:

```
-Dproperty_name=property_value
```

To use properties in a file, you must set the property oracle.oas.mercury.config which specifies the name of the file that contains configuration values. This is specified on the command line. Using a file rather than command line properties may be more appropriate for the following reasons:

- Command lines only allow a limited number of properties to be set.
- Using a file prevents a user from having to enter and possibly reenter long command lines.

The syntax for properties specified in a file is:

```
property_name=property_value
property_name=property_value
.
.
.
```

In Java, property names are case sensitive. This may lead to confusion when a user types a property name that is equivalent, except for case. To prevent confusion, Oracle Message Broker converts all property names to lower case internally, so that the property name case, from a users point of view, is not significant.

When properties are read from a file, do not include trailing blanks after the specified *property_value*. Trailing blanks may be interpreted as part of the value. For example, `'oracle.oas.mercury.maxMemory=3'` is correct, since there are not any trailing blanks, but `'oracle.oas.mercury.maxMemory=3 '` is incorrect.

Properties that are specified on the command line are stored with the JVM's system properties. These properties can be retrieved using a call to java.lang.System.getProperty(String). Oracle Message Broker properties that are read from a configuration file are not stored as system properties.

Property Evaluation Precedence Rules

The Oracle Message Broker uses the following precedence rules for property values:

1. Retrieve the names of all system properties.
2. Store these properties in a first hash table after converting names to lower case.
3. Read all properties stored in the configuration file specified by oracle.oas.mercury.config.
4. Store these in second has table after converting all names to lower case.
5. For each property that a user can specify, do the following, in order:
   a. Convert the name to lower case.
   b. If the property is stored in the first hash table, use its value. Otherwise, if the property is stored in the local hash table, use the value from the second hash table. As indicated, the value in system hash table takes precedence.

Note: This implies that if a property appears in both the system properties and in the configuration file, the system properties value is used.

Lightweight Configuration Properties

This section describes the Oracle Message Broker configuration properties. The set of all valid Oracle Message Broker configuration property names cannot be determined in advance, since the values for some properties determines the valid names for other properties.

For example, using Lightweight Configuration, you can add a destination named myQueue with the property oracle.oas.mercury.destinations. If myQueue is

specified as a destination, then the property oracle.oas.mercury.dest.myQueue.isTopic is also a valid property name.

In Table 13–4, Table 13–5, and Table 13–6, parameterized values appear as part of a property names when the value is a property value, that is used to generate other valid property names.

Oracle Message Broker Lightweight Configuration property values can be one of four types: an integer, a string, a list of strings, or boolean. Strings are not quoted. A list of strings is comma-delimited and is of the form: *string* [,*string*]*. The string and list of string syntax does not allow for spaces. Valid boolean values are `true` and `false` and strings equivalent to 'true' or 'false', ignoring case.

Table 13–3 describes the general configuration properties for Oracle Message Broker Lightweight Configuration.

In Table 13–3, Table 13–4, Table 13–5, and Table 13–6, all property names begin with `oracle.oas.mercury.` Optional properties are enclosed in brackets, "[]".

Table 13–3 Broker Configuration Properties

Property Name	Description
[oracle.oas.mercury.config]	This property specifies the configuration file. If the Oracle Message Broker cannot open the specified file an exception is thrown and the Oracle Message Broker does not start. **Default:** no default value **Type**: String
[oracle.oas.mercury.key]	The property value is used to construct the name of the file in which the IOR is stored. The name for this file is constructed as `JMSProvider`*key*, where *key* is the string provided by this property. **Default:** If no value is specified, the Oracle Message Broker uses the value of the system property, user.name, if the user.name is not set, '`_default`' is used. **Type**: String
[oracle.oas.mercury.maxConcurrent]	The minimum value is 5. The maximum value is 10000. When the Oracle Message Broker is used in Remote Mode, this value sets the number of generic ORB threads which can handle requests. It is also used to limit the number of concurrent blocking calls to a value less than the number of generic ORB threads. This prevents all of the threads from being stuck in blocking calls. Blocking calls are calls to create subscribers, destroy subscribers, and some variants of receive. When setting this parameter, keep in mind that each thread consumes resources (the resources consumed by a thread includes a 64k stack). **Default:** 10 **Type**: integer

Table 13–3 (Cont.) Broker Configuration Properties

Property Name	Description
[oracle.oas.mercury.maxHeap]	The minimum value is 4. The maximum value is `java.lang.Integer.MAX_VALUE`. This property represents the maximum size of the JVM heap, in megabytes, for the Oracle Message Broker process. When a JVM is started, it has a maximum and initial heap size. The actual size of the JVM heap will be less than or equal to the maximum size. The actual size of the JVM heap can be determined at runtime, but the maximum size cannot. The Oracle Message Broker uses the maxHeap value to prevent the Oracle Message Broker process from running out of memory. **Default**: 4 **Type**: integer
[oracle.oas.mercury.maxMemory]	The minimum value is 4. The maximum value is 4096. This property represents the number of megabytes of the JVM heap that the Oracle Message Broker can use. This value must be less than or equal to oracle.oas.mercury.maxHeap. When calls are made that might cause the Oracle Message Broker to consume memory, it checks the amount of memory in use and throws an exception if too much memory is in use. Too much memory is in use when more than (.7 * *maxHeap*)megabytes of the JVM heap are in use. See the oracle.oas.mercury.maxHeap property for more information). **Default:** 4 **Type**: integer
[oracle.oas.mercury.maxPropRecv]	This property determines the number of threads used to receive messages for propagation jobs. Propagation is not supported using Oracle Message Broker Lightweight Configuration. **Default**: 1 **Type**: integer
[oracle.oas.mercury.maxPropSend]	This property determines the number of threads used to send messages for propagation jobs. Propagation is not supported using Oracle Message Broker Lightweight Configuration. **Default**: 1 **Type**: integer
[oracle.oas.mercury.propJobFile]	This is ignored using Lightweight Configuration. **Default**: no default value **Type**: String

Subscriber Configuration Properties

Table 13–4 describes the subscriber properties for Oracle Message Broker Lightweight Configuration. The parameter *sub_name* indicates that the subscriber name should be included as part of the property name.

Table 13–4 Subscriber Configuration Properties

Property Name	Description
[oracle.oas.mercury. subscribers]	The Oracle Message Broker creates a durable subscription for each subscriber on the list if the other required properties for the subscriber are also specified. For the following properties, the names in the string list are substituted for *sub_name*. **Default:** is an empty list **Type**: list of strings.
[oracle.oas.mercury. subscriber.*sub_name*.clientID]	If this property is specified, it will be used as the client ID for the durable subscription. **Default:** no default value **Type**: String
[oracle.oas.mercury. subscriber.*sub_name*.selector]	If this property is specified, it will be used as the message selector for the durable subscription. **Default:** no default value **Type**: String
oracle.oas.mercury. subscriber.*sub_name*.topic	This is the name of the topic for which the durable subscription is created. The topic must also be defined for the subscription to be created. **Default:** no default value **Type**: String

Driver Properties

Table 13–5 describes the driver properties. Some properties are particular to an individual driver. Those properties include the name of the driver rather than *driver_name*. Properties that can be specified for any driver include *driver_name* in the name of the property. The *driver_name* should be replaced by the name of the driver ('aq', 'mq', 'vol', 'rv', 'mcast').

Table 13–5 Driver Configuration Properties

Property Name	Description
[oracle.oas.mercury. drivers]	The set of strings that can be on the list are {'vol', 'aq', 'aqlite, 'mq', 'rv', 'mcast'}. Additional properties can be specified for each driver that is named in drivers. The names specified in drivers should be substituted for '*driver_name*' in the following properties. **Default:** vol **Type**: String
[oracle.oas.mercury. driver.*driver_name*.maxSharedSessions]	Some drivers use this property to control their resources. This value is ignored by the drivers: vol, rv, mcast, and mq. The AQ Driver uses this parameter as the number of JDBC connections that are created in a pool and used internally for administrative purposes. Administrative uses include creating and destroying AQ queues and subscriptions. The AQ Driver in the OCI mode creates one connection per JMS session to provide operational access to AQ queues. The AQ Driver in the JDBC mode creates one JDBC connection per JMS session to provide operational access to AQ queues. The minimum value is 2. There is no maximum value. **Default:** 8 **Type**: integer
[oracle.oas.mercury. driver.*driver_name*.maxPrivateSessions]	The minimum value is 2. There is no maximum value. This property is used by some drivers to control resources. This value is ignored by the drivers: vol, mcast. The rv driver limits the number of JMS sessions to this value (approximately). The mq driver limits the number of JMS sessions to this value. The aq driver uses this to set the maximum number of concurrent JMS sessions that the driver allows to be created. The Database Server should be configured to support more than maxPrivateSessions. **Default:** 8 **Type**: integer

Table 13–5 (Cont.) Driver Configuration Properties

Property Name	Description
[oracle.oas.mercury. driver.*driver_name*.numPushSessions]	The minimum value is 2. There is no maximum value. This determines the number of threads started by the Oracle Message Broker to push messages to message listeners. There will be at most numPushSessions concurrent pushes to listeners from a particular driver. **Default:** 10 **Type**: integer
[oracle.oas.mercury. driver.*driver_name*.propSessions]	The minimum value is 0. There is no maximum value. This determines the number propagation threads started by the Oracle Message Broker for the specified driver. **Default:** 10 **Type**: integer
[oracle.oas.mercury. driver.*driver_name*.queryInterval]	The minimum value is 10. There is no maximum value. It is the number of milliseconds that the per-driver query thread will sleep in between queries. The query thread queries all queues and topics for which message listeners have been created to determine the count of available messages. If this value is too small the Oracle Message Broker may consume too much CPU time performing the queries. If this value is too large clients may see a delay between the time at which a message is available and the time at which the Oracle Message Broker determines that a message is available. **Default:** 1000 **Type**: integer
oracle.oas.mercury. driver.*driver_name*.propSendLogQueue	It is the name of the queue that is used by propagation for logging by send threads. It can be the same as propRecvLogQueue. Propagation is not supported using Oracle Message Broker Lightweight Configuration. **Default:** no default value **Type**: String
oracle.oas.mercury. driver.*driver_name*.propRecvLogQueue	It is the name of the queue that is used by propagation for logging by receive threads. It can be the same as propSendLogQueue. Propagation is not supported using Oracle Message Broker Lightweight Configuration. **Default:** no default value **Type**: String

Table 13–5 (Cont.) Driver Configuration Properties

Property Name	Description
oracle.oas.mercury. driver.aq.serviceName	It is required when thinJdbc is false or useJdbc is false. It is the service name used to create OCI connections. It is also used to create JDBC connections when the OCI based JDBC driver is used. The URL in the call to java.sql.DriverManager.getConnection is 'jdbc:oracle:oci8:*@serviceName* ' after the value of serviceName is substituted for '*serviceName*'. **Default:** no default value **Type**: String
[oracle.oas.mercury. driver.aq.thinJdbc]	This determines whether the thin or OCI based JDBC driver should be used. The thin JDBC driver is used when thinJdbc is true. The OCI based JDBC driver is used when thinJdbc is false. **Default:** false **Type**: boolean
oracle.oas.mercury. driver.aq.thinJdbcConnString	It is required when thinJdbc is true. This is used to determine the URL in the call to java.sql.DriverManager.getConnection. The URL is 'jdbc:oracle:thin:*@thinJdbcConnString*' after the value of thinJdbcConnString is substituted for '*thinJdbcConnString*'. **Default:** no default value **Type**: String
oracle.oas.mercury. driver.aq.adminUser	It is the name used to authenticate DBMS connections by the AQ Driver. The OCI based AQ Driver performs all Oracle8 access with this identity. The JDBC based AQ Driver performs all administrative access with this identity. Administrative access includes: create subscription, destroy subscription, create queue, destroy queue. The JDBC based AQ Driver also uses this as the default identity when an Oracle Message Broker client creates a JMS connection without supplying a name and password. Otherwise, the name and password supplied by the Oracle Message Broker client will be used to authenticated the JDBC connection. **Default:** no default value **Type**: String
oracle.oas.mercury. driver.aq.adminPassword	This property specifies the password used to authenticate DBMS connections by the AQ Driver. It is used in association with oracle.oas.mercury.driver.aq.adminUser. **Default:** no default value **Type**: String

Table 13–5 (Cont.) Driver Configuration Properties

Property Name	Description
[oracle.oas.mercury. driver.aq.useJdbc]	It determines whether the JDBC based AQ Driver or the OCI based AQ Driver is started when the Oracle Message Broker uses the AQ Driver. The JDBC based AQ Driver is started when the value is 'true'. **Default:** false **Type**: boolean
[oracle.oas.mercury. driver.aqlite.dbname]	The name of the AQ Lite database. In case where the name is null, use the default database. **Default:** null **Type**: String
[oracle.oas.mercury. driver.aqlite.dbpasswd]	The password to connect to the AQ Lite database. When the password is null no password is used. **Default:** null **Type**: String
oracle.oas.mercury. driver.rv.certified	When true TIBCO provides a higher quality of service for message delivery. **Default:** false **Type**: boolean
[oracle.oas.mercury. driver.rv.service]	It determines the name of the TIBCO service to use when creating TIBCO sessions. **Default:** null (default service) **Type**: String
[oracle.oas.mercury. driver.rv.network]	It determines the network interface to use when creating TIBCO sessions. **Default:** null (default network interface) **Type**: String
[oracle.oas.mercury. driver.rv.daemon]	It determines the name of the TIBCO daemon to connect to when creating TIBCO sessions. **Default:** null (local daemon) **Type**: String
oracle.oas.mercury. driver.mcast.ip	It determines the IP multicast address to use for multicast communication. The valid range is between 225.0.0.0 and 239.255.255.255, inclusive. **Default:** no default value **Type**: String

Table 13–5 (Cont.) Driver Configuration Properties

Property Name	Description
oracle.oas.mercury. driver.mcast.port	The valid range is between 1024 and 65535, inclusive. It determines the IP multicast port number to use for multicast communication. **Default:** no default value **Type**: integer
[oracle.oas.mercury. driver.mcast.network]	It determines the local IP address of the network interface used by the mcast driver. **Default:** null (default network interface) **Type**: String

Destination Properties

Some properties must be specified for each topic or queue. These include '*dest_name*' in the name of the property. '*dest_name*' should be replaced by the name of the destination when this property name is used. Some properties are only applicable when a particular driver is used. Those properties include the name of the driver in the property name.

Table 13–6 Destination Configuration Properties

Property Name	Description
[oracle.oas.mercury. destinations]	This list contains the names of the queues and topics the Oracle Message Broker can access. Other properties must be specified for each name on this list. The other properties follow. **Default:** empty list **Type**: String
[oracle.oas.mercury. destination.*dest_name*.isTopic]	This can be specified for each destination in the string lists for topics and queues. '*dest_name*' is a topic when this is true. Otherwise, '*dest_name*' is a queue. **Default:** true **Type**: boolean
oracle.oas.mercury. destination.*dest_name*.driver	It must be one of: 'aq', 'aqlite, 'mcast', 'mq', 'rv', 'vol'. This determines the driver that is used to access the topic or queue. Access to this destination will fail if the driver cannot be started or if the driver has not been configured to start. **Default:** no default value **Type**: String

Table 13–6 (Cont.) Destination Configuration Properties

Property Name	Description
[oracle.oas.mercury. destination.*dest_name*.vol.maxMsgs]	The minimum value is: 10. The maximum value is: java.lang.Integer.MAX_VALUE. This determines the maximum number of messages that can be stored in a volatile queue or volatile topic. The Oracle Message Broker may not have enough memory to store the number of messages specified by this limit. Messages that have been received in a transaction are counted against this limit until the transaction is committed. The count of messages in a volatile topic is incremented once for each subscriber that can receive the message. If a message is published to a topic, and there are 100 subscribers to the topic, then the count of messages in the topic will be incremented by 100. **Default:** java.lang.Integer.MAX_VALUE **Type**: integer
[oracle.oas.mercury. destination.*dest_name*.aq.internalName]	**Type**: String It is the name of the AQ queue as used within the Oracle8*i* Database Server (single-consumer or multi-consumer), which does not have to be the same as '*dest_name*'. The name must not include the schema prefix. **Default:** *'dest_name'* **Type**: String
[oracle.oas.mercury. destination.*dest_name*.aq.aqRules]	When this is true the AQ rules engine is used to implement message selectors for topic subscribers. When this is false, the Oracle Message Broker message selector engine is used to implement message selectors for topic subscribers. The Oracle Message Broker message selector engine is always used to implement message selectors for queue consumers. Note: This property is only valid for topics. **Default:** false **Type**: boolean
oracle.oas.mercury. destination.*dest_name*.aq.schema	This is the schema in which the AQ queue resides. **Default:** no default value **Type**: String
oracle.oas.mercury. destination.*dest_name*. aqlite.owner	Specifies the owner of the AQLite queue or topic. **Default**: default user **Type**: String

Table 13–6 (Cont.) Destination Configuration Properties

Property Name	Description
oracle.oas.mercury. destination.*dest_name*. aqlite.message.grouping	Specifies the grouping of messages in AQ Lite. Valid values: NONE or TRANSACTIONAL **Default**: NONE **Type**: String
oracle.oas.mercury. destination.*dest_name*.aqlite.storage_clause	The storage clause used to create the AQ Lite queue table. If the storage clause is null then no storage clause is used. **Default**: null **Type**: String
[oracle.oas.mercury. destination.*dest_name*.mq.internalName]	It is the name of the MQSeries queue, which does not have to be the same as '*dest_name*' **Default:** '*dest_name*' **Type**: String
[oracle.oas.mercury. destination.*dest_name*.mq.isNative]	When it is false, JMS messages are serialized and stored as a stream of bytes on an MQSeries queue. Some fields from the MQSeries header are used to set values in the JMS message. When it is true, only text messages and bytes messages can be stored on the queue and the body of the text or bytes message is stored as the body of the MQSeries message. The properties from the JMS message are not stored in the MQSeries message. **Default:** false **Type**: boolean
[oracle.oas.mercury. destination.*dest_name*.mcast.internalName]	It is the name of the mcast topic, which does not have to be the same as '*dest_name*'. **Default:** '*dest_name*' **Type**: String
[oracle.oas.mercury. destination.*dest_name*.rv.internalName]	It is the name of the rv topic, which does not have to be the same as '*dest_name*'. **Default:** '*dest_name*' **Type**: String
[oracle.oas.mercury. destination.*dest_name*.rv.isNative]	When it is true, JMS messages are stored in a TIBCO specific format. When it is true, JMS messages are stored as a serialized stream of bytes that can only be accessed by the Rv driver. **Default:** false **Type**: boolean

Sample Configuration Files

Refer to "Starting with Lightweight Configuration in Remote Mode" on page 13-4 for a sample command line showing how to start the Oracle Message Broker using Lightweight Configuration. The Oracle Message Broker includes lightweight configuration samples in $OMB_HOME/samples/client/java/lightweight, or on Windows NT systems, %OMB_HOME%\samples\client\java\lightweight.

Lightweight Configuration Constraints and Limitations

1. Durable Subscriber Limitations

   When running the Oracle Message Broker with the LDAP Directory, the Oracle Message Broker stores information about durable subscribers in directory entries. The directory entries for durable subscribers are updated at runtime as they are created and unsubscribed. With Lightweight Configuration, the Oracle Message Broker does not record the durable subscription, except the message store (creation subscription to AQ is durable by itself).

   If a client creates a durable subscriber, and that durable subscriber has not been read during startup with Lightweight Configuration, the Oracle Message Broker does not maintain or update a persistent record of the new durable subscriber (the Oracle Message Broker does not write Lightweight Configuration information, it only reads it).

   The Oracle Message Broker uses its knowledge of existing subscribers, as it is provided at startup, or during runtime, to determine when to perform an implicit unsubscribe when processing a subscribe request. The Oracle Message Broker performs an implicit unsubscribe when a request is made to create a subscriber, and there is an existing subscription to that topic with the same subscriber name, and the existing subscription uses a message selector that is different than the message selector specified with the subscribe request. This is only an issue for the AQ Driver since durable subscribers to AQ multiconsumer queues survive Oracle Message Broker failures. If AQRules is enabled, Oracle Message Broker queries the message selector stored in the DBMS to see if the new selector is different. If AQRules are not enabled, Oracle Message Broker assumes the message selector is the same as long as the Oracle8*i* Database Server does not contain a selector for the subscribe.

   The Oracle Message Broker creates durable subscribers specified in the configuration information at startup. However, there is not an administrative mechanism in Lightweight Configuration that causes the Oracle Message

Broker to unsubscribe to a durable subscriber. Unsubscribes only occur when an Oracle Message Broker client requests an unsubscribe.

2. Prerequisite Knowledge

   The Oracle Message Broker can be configured to use more resources, memory, DBMS connections, than resource providers (virtual memory system, DBMS) have been configured to provide. The person specifying configuration values is responsible for ensuring that appropriate values are specified.

   A DBMS that is used by both Oracle Message Broker to access Oracle AQ and Oracle Internet Directory must support the resource requirements of both systems.

3. Security Issues

   Some of the information that the Oracle Message Broker accesses using Lightweight Configuration is private. Passwords that are used to authenticate DBMS connections can be specified using Lightweight Configuration. It is the responsibility of the Oracle Message Broker user to protect access to both the command line and the Lightweight Configuration file when they contain sensitive information such as passwords.

4. If a durable subscriber for an AQ multi-consumer queue is created at runtime, and that subscriber is not specified in the configuration information processed at startup, and the AQ rules engine is not used to implement message selectors, then the Oracle Message Broker will not know when to perform an implicit unsubscribe. This will cause behavior that violates the JMS specification.

   The following scenario demonstrates this failure:

   a. The Oracle Message Broker is started using lightweight configuration.

   b. No durable subscribers are specified using lightweight configuration.

   c. A durable subscriber with no message selector is created to an AQ multi-consumer queue at runtime.

   d. Messages are enqueued to the AQ multi-consumer queue.

   e. The Oracle Message Broker is stopped.

   f. The Oracle Message Broker is restarted.

   g. A durable subscriber with a message selector and the same name as the durable subscriber in step c is created to the same AQ multi-consumer queue.

If any messages enqueued in step d satisfy the message selector specified in step g, these message can be received by durable subscriber created in step g. But the JMS specification requires that an unsubscribe be done at step g since the subscriber is created with a different message selector. Since the subscriber had not been recorded in the configuration information, the Oracle Message Broker has no way to determine that this is the case. When the AQ rules engine is used, the Oracle Message Broker will query the existing rule used by an durable subscriber and this problem will not occur.

5. Command Line Limits

   There are limits on the size of a Java command line. Due to this limitation, users may not be able to specify all of the configuration data that is required on a command line. This limit may be smaller than 1024 characters. If you reach this limit, use a configuration file to specify configuration data.

6. Property Limitation

   Avoid setting the following combination of property values:

   ```
   oracle.oas.mercury.driver.aq.useJDBC=true
   oracle.oas.mercury.driver.aq.thinJDBC=true
   ```

   These property values conflict, and results are unpredictable.

   With the value of `useJDBC` set to true, the only value supported for `thinJDBC` is false:

   ```
   oracle.oas.mercury.driver.aq.useJDBC=true
   oracle.oas.mercury.driver.aq.thinJDBC=false
   ```

14

Asynchronous Component Invocation

This chapter covers the Oracle Message Broker Asynchronous Component Invocation (ACI) feature. ACI links JMS, as provided by the Oracle Message Broker, to applications running in the Oracle Database Server. ACI can automatically invoke Enterprise JavaBeans (EJB), when messages arrive at a JMS destination. The EJB can subsequently consume and process JMS messages from a queue or topic.

> **Note:** EJBs can only access remote mode Oracle Message Brokers. Oracle Message Broker Local mode is not supported.

Asynchronous Component Invocation has the following important features:

- ACI notifies an EJB when a condition on a queue or topic is met. Once notified, the EJB acts as an Oracle Message Broker client.
- EJBs that process notifications can use the standard JMS API or higher-level classes that simplify the JMS calls required to process messages.
- EJBs can have full programmatic access to Oracle Message Broker using JMS.
- ACI configuration is performed using the Oracle Message Broker administration utilities.

This chapter covers the following:

ACI Architecture

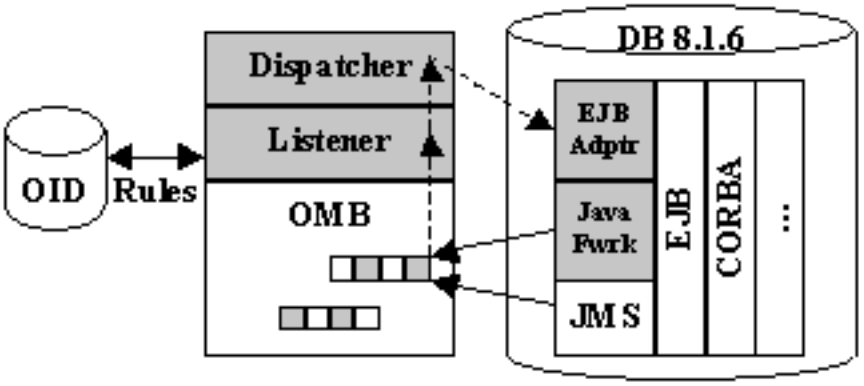
The Oracle Message Broker ACI feature consists of the following components:

- ACI Listener – listens to a set of queues or topics (see "ACI Listener" on page 14-3).
- ACI Dispatcher - sends notifications to registered EJBs according to the specified configuration (see "ACI Dispatcher" on page 14-3).
- ACI Adapters (EJB Adapter) – provide an application-side framework for receiving asynchronous notifications (see "ACI Adapters" on page 14-4).
- ACI Helper Classes (Java Framework) - the helper classes can optionally be used by EJBs to simplify management of JMS messages (see "ACI Helper Classes" on page 14-4).

Both the ACI Listener and the Dispatcher reside in the same JVM as the Oracle Message Broker.

Figure 14–1 shows a high-level view of ACI components.

Figure 14–1 ACI Components



ACI Listener

When a user-defined condition occurs, the ACI Listener generates an event that sends a notification to an EJB. Each condition, called a **trigger**, acts on one and only one JMS destination (queue or topic). It is not possible to set triggers that involve multiple JMS destinations.

The ACI Listener watches one or more JMS destinations and checks if a condition, specified by a rule stored in the LDAP Directory, is met. When a condition is met, the ACI Listener generates a notification event and gives it to the ACI Dispatcher.

When several ACI triggers are set on the same JMS queue, all the triggers that meet the condition associated with their rules receive a notification. In this situation, it is possible for several EJBs to receive a notification for the same message. In this case, only one of the EJBs consumes the message. It is part of the developer's task to ensure that multiple ACI triggers set on the same JMS queue do not compete for the same messages. One way around this problem is to use disjoint message selectors for the rules associated with a trigger on a queue.

When associating ACI triggers to topics, each trigger acts as a distinct consumer, and all notified EJBs can consume copies of the same messages.

ACI Dispatcher

The ACI Dispatcher sends notifications to registered EJBs according to the specified configuration. The EJB can then consume one or more messages using the Java framework or using JMS.

The ACI Dispatcher invokes EJBs at the request of the ACI Listener. EJB invocations are asynchronous, that is, many requests can be issued and managed simultaneously without performance degradation in the Oracle Message Broker.

Using ACI, JMS topics require a durable subscription. Since JMS only allows one active use of a durable subscription, there must only be only one notification outstanding at a time for topics. This means that the ACI Dispatcher cannot issue a notification until the previous notification is completed. This may become a problem with some trigger configurations if EJBs take time to process notifications, and there are many messages published on a topic.

If the ACI Dispatcher fails invoking an EJB multiple consecutive times for the same trigger (this may happen for instance if the Oracle database is not started, or if the EJB does not provide robust error handling and a queue contains an invalid message that systematically make the EJB fail), that trigger will be automatically canceled and a message will be inserted in the log file. To re-enable the trigger, one

has to explicitly disable and re-enable the trigger using Oracle Message Broker administrative tools.

ACI Adapters

ACI Adapters (EJB Adapter) - provide an application-side framework for receiving asynchronous notifications. Each adapter supports a type of component. Currently, the ACI feature only provides an adapter for Oracle EJBs.

ACI Helper Classes

ACI Helper Classes (Java Framework) - the helper classes can optionally be used by EJBs to simplify management of JMS messages. Such classes automate the process of consuming messages by hiding most of the JMS complexity (e.g., creating a connection, session, and consumer). These APIs make EJB programming easier, as compared to JMS programming, upon asynchronous invocation.

EJBs that do not use the ACI helper classes and directly use the JMS API have the following restrictions:

- EJBs should not use message listeners or exception listeners. An exception will be thrown if an EJB tries to register such a message or exception listener.
- EJBs should not directly use JNDI to look up destinations and connection factories. They should instead use the methods defined in the ACIHelper class.
- EJBs should only use non-blocking primitives for message reception. A well-behaved EJB invoked by ACI should process the message available on the JMS destination, but not wait for new messages.

Note: If an EJB has an opened connection to a broker and does not access the broker for more than 1 minute, then it is assumed to have failed. EJBs must close connections before terminating.

ACI Triggers

The conditions upon which asynchronous component invocations occur are specified by setting ACI triggers on a JMS destination using an LDAP Directory ACI trigger entry. Trigger entries are configured using the Oracle Message Broker administrative tools.

Trigger entries have the following parameters:

- Information about the destination for which the trigger applies. This information includes the name of the connection factory, the name of the destination (topic or queue), an optional message selector (for queues only), and the name of a durable subscription (for topics only).
- Name of the component to be invoked when the trigger's condition is met.
- User name and password of the schema to which the component belong.
- The status of the trigger (enabled or disabled).
- The trigger threshold (see "Setting the ACI Threshold Parameter" on page 14-5).
- The trigger concurrency (see "Setting the ACI Concurrency Parameter" on page 14-6).
- The number of retries upon abnormal ACI termination.

ACI triggers are set by specifying the above parameters on a destination, and by enabling the trigger using the status parameter. Since some trigger parameters only apply to queues or topics, there are two ACI trigger entry types: QueueTrigger and TopicTrigger (for more information on administration for these entries "Creating and Configuring Asynchronous Component Invocation Triggers" on page 4-36).

Setting the ACI Threshold Parameter

The threshold parameter specifies the number of messages that must be stored in a destination before an ACI is sent. The ACI subsystem cannot guarantee that an invocation receives a "threshold" number of messages. The quality of service is best effort and the number of invocations issued may not match what is expected. Problems that could result in a different number of messages delivered include:

- Non-Oracle Message Broker clients consuming messages from the destination
- EJBs that consume more than the number of messages specified by the threshold.
- EJBs that consume less than the number of messages specified by the threshold.

For ACI to work as expected, each component should consume only the number of messages specified with the "threshold" parameter.

Setting the ACI Concurrency Parameter

The concurrency parameter specifies the number of concurrent ACIs that can execute for a given trigger (for queues only). ACI attempts to guarantee that there are never more invocations in progress for a trigger at one time than the concurrency parameter. This property will always be true, unless a communication failure happens while a component is processing the asynchronous notification, and the dispatcher interprets the communication failure as a component failure.

Note that ACI invocations must not be issued concurrently for topics (therefore the "concurrency" parameter must be equal to 1). This limitation is implied by the JMS specification, which specifies that only one session at a time can have a topic subscriber for a particular durable subscription. Since triggers set on topics require durable subscriptions, the triggering mechanisms must ensure that there is only one active invocation at a time.

EJB Adapter

The ACI EJB adapter provides support classes for writing Enterprise JavaBeans that process incoming JMS messages. The ACI EJB adapter provides two sets of classes that support the interface between EJBs and the Oracle Message Broker:

- Notification-driven beans receive an asynchronous notification when some condition has been met on a destination. They can subsequently consume messages from that destination using the JMS API.
- Message-driven beans asynchronously receive JMS messages from a destination. They do not need to dequeue the message programmatically.

Message-driven beans provide a higher-level abstraction and are easier to use. Notification-driven beans are more flexible; for instance, they make it possible to consume several messages as part of the same transaction, or browse a queue to find a specific message.

Notification-driven Beans

The EJB adapter defines the following classes and interfaces for notification-driven beans:

```
/*
 * ACI exception class.
 */
public class ACIException extends Exception {
  // ...
}


/**
 * The description of the JMS destination on which the message(s) reside(s).
 */
public abstract class DestinationDescriptor
{
  public String cf;           // Connection factory to use
  public String dest;         // JMS destination to access
  public String selector;     // Optional message selector
  public String subscription; // Durable subscription (for topics only)
}


public class QueueDescriptor extends DestinationDescriptor
{
}

public class TopicDescriptor extends DestinationDescriptor
{
  public String subscription; // Durable subscription
  public String client_id;    // Client identifier
}

/**
 * This interface must be inherited by an EJB “Remote” interface that will get
 * messages from a JMS destination.
 */
public interface ACIRemote
{
  public boolean msgPending(DestinationDescriptor dest, int threshold)
    throws ACIException;
}
```

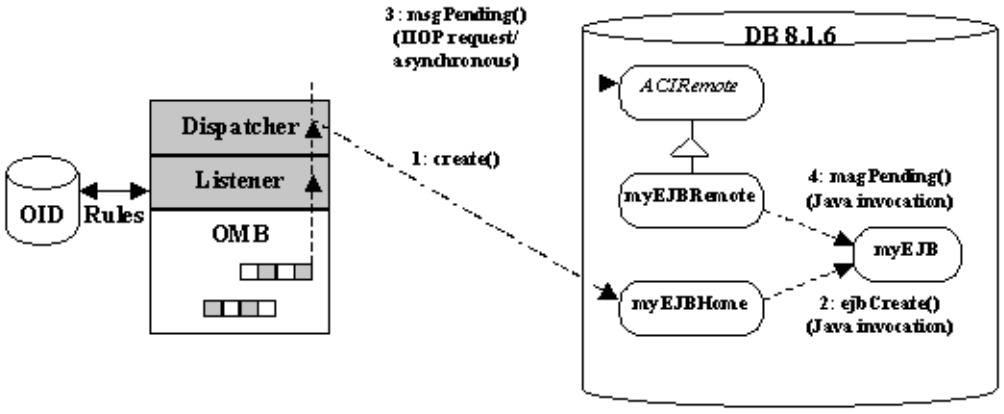
The “DestinationDescriptor” class defines the information necessary for the EJB to get the message(s) that triggered the asynchronous invocations. The “cf” and “dest” members are names that can be used to lookup a connection factory and a destination. The “selector” member is an optional message selector used by the ACI Listener. “DestinationDescriptor” is an abstract class. Concrete destination descriptors are instances of the “QueueDescriptor” or “TopicDescriptor”, which inherit from “DestinationDescriptor”. “QueueDescriptor” does not add member variables to “DestinationDescriptor”. “TopicDescriptor” defines an additional subscription name (“subscription”) and the client identifier (“client_id”), necessary for consuming messages through a durable subscriber. Note that the topics used with ACI must always be accessed through durable subscribers. The durable subscriber associated to a trigger must be created using Oracle Message Broker’s administrative tools prior to starting the broker.

Notification-driven beans implement the “msgPending” operation, which will be invoked asynchronously by Oracle Message Broker. As a convenience, the ACI EJB adapter provides an interface called “ACIRemote” that defines the “msgPending” method: notification-driven beans simply need to implement this interface. Using this interface is however not mandatory, and EJBs can directly define “mgsPending” among their own public methods.

The “msgPending” method must return true if it completed successfully, and false otherwise (in that case, the method may be re-invoked depending on the configuration of the ACI trigger). “msgPending” takes a destination descriptor (instance of “QueueDescriptor” or “TopicDescriptor”) and an integer number which indicates the number of messages that the component is expected to consume, based on the configuration of the ACI trigger.

Figure 14-2 illustrates the asynchronous invocation of a notification-driven bean of type “MyEJB”. Oracle Message Broker first creates an EJB by invoking the “create” operation of the EJB “Home” object (1). As a result of this request, the EJB receives an invocation to its “ejbCreate” method (2). Then, Oracle Message Broker sends an asynchronous invocation to the “msgPending” operation of the EJB “Remote” object (3), which leads to the actual invocation of the “msgPending” method of the bean (4).

Figure 14–2 Asynchronous Invocation of an EJB



Changing a standard EJB into a notification-driven bean is easy: it just requires adding an inheritance link to the “Remote” bean interface (optional), implementing the “msgPending” operation, and re-deploying the bean.

Sample Notfication-Driven Bean

The directory $OMB_HOME/samples/aci/jms (or on Windows NT systems %OMB_HOME%\samples\aci\jms), contains a sample notification-driven bean.

Message-driven Beans

A message-driven bean is a JMS “MessageListener”. A message-driven bean is invoked implicitly as a result of the arrival of a JMS message. Message-driven beans must not be created or invoked explicitly by clients. They are anonymous and have no client-visible identity.

A message-driven bean instance is an instance of a message-driven bean class. To a client, a message-driven bean is a message consumer that implements some business logic running on the server.

A client accesses a message-driven bean through JMS by sending messages to the JMS destination (queue or topic) for which the message-driven bean is the “MessageListener”. The “onMessage” method is called when a message has arrived for the bean to service. The “onMessage” method contains the business logic that handles the processing of the message. It has a single argument, the incoming message. Message-driven bean instances have no conversational state. This means that all bean instances are equivalent when they are not involved in servicing a client message.

All message-driven beans must implement the “MessageDrivenBean” interface. The “setMessageDrivenContext” method is called to associate a message-driven bean instance with its context maintained by the container. Typically a message-driven bean instance retains its message-driven context as part of its state.

The ACI message-driven bean framework provides the message-driven bean instance with a “MessageDrivenContext”. The “MessageDrivenContext” interface has the following methods:

- The “getCallerPrincipal” method returns the “java.security.Principal” that is associated with the invocation of the bean instance.
- The “isCallerInRole” method tests if the principal associated with the invocation of the message-driven bean instance has a particular role.
- The “setRollbackOnly” method allows the instance to mark the current transaction such that the only outcome of the transaction is a rollback. Only instances of a message-driven bean with container-managed transaction demarcation can use this method. This method applies only to the EJB’s transaction, and does not have any influence on JMS transacted sessions.
- The “getRollbackOnly” method allows the instance to test if the current transaction has been marked for rollback. Only instances of a message-driven bean with container-managed transaction demarcation can use this method. This method applies only to the EJB’s transaction, and does not have any influence on JMS transacted sessions.
- The “getUserTransaction” method returns the “javax.transaction.UserTransaction” interface that the instance can use to demarcate transactions, and to obtain transaction status. Only instances of a message-driven bean with bean-managed transaction demarcation can use this method. This method applies only to the EJB’s transaction, and does not have any influence on JMS transacted sessions.
- The “getEJBHome” method is inherited from the “EJBContext” interface. Message-driven bean instances must not call this method.

The “ejbRemove” notification signals that the instance is in the process of being removed by the container. In the “ejbRemove” method, the instance releases the resources that it is holding.

A message-driven bean is created in three steps. First, the message-driven bean framework creates a new message-driven bean instance. Second, it calls the “setMessageDrivenContext” method to pass the context object to the instance. Third, it calls the instance’s “ejbCreate” method. Each message-driven bean class must have one “ejbCreate” method, with no arguments.

Calls to each message-driven bean instance are serialized. Therefore, a message-driven bean does not have to be coded as reentrant. However, many instances of a message-driven bean class can be executing concurrently, thus allowing for the concurrent processing of a stream of messages. No guarantees are made as to the exact order in which messages are delivered to the instances of the message-driven bean class. Message-driven beans should therefore be prepared to handle messages that are out of sequence: for example, the message to cancel a reservation may be delivered before the message to make a reservation.

Message-driven beans are actually implemented using underlying session beans. Like standard session beans, they have a home and a remote interface. These interfaces should however not be used explicitly by clients.

An ACI message-driven bean must extend the “MessageDrivenBeanAdapter” class. All ACI message-driven classes are declared in the “oracle.oas.aci” package.

Messages are consumed on a destination using either a transacted or a non-transacted session. To use a transacted session, the property “oracle.oas.mercury.aci.transacted” must be set to “true” in the bean’s environment. When using a transacted session, each message is consumed in the context of a new transaction, and the transaction is committed when the “onMessage” method returns. It is not possible to perform other operations in the context of that transaction. However, the user transaction returned by the bean’s context can be used to create an independent transaction in the context of the database.

The directory $OMB_HOME/samples/aci/messagedriven (%OMB_HOME%\samples\aci\messagedriven on Windows NT), contains a sample message-driven bean.

Java Helper Classes

Oracle Message Broker's ACI feature provides helper classes that can optionally be used by EJBs to simplify management of JMS messages. When using the EJB adapter and notification-driven beans, the developer can use the "ACIHelper" class to automate the process of consuming JMS messages.

The "ACIHelper" class is instantiated with a destination descriptor, a bean's session context, and a boolean flag indicating whether sessions have to be transacted or not. It provides simple functions for consuming messages, without requiring the EJB to know whether messages come from a queue or a topic, and to deal explicitly with connections, sessions, or producers. Note that it does not provide blocking methods for consuming messages, since such methods should not be used by well-behaved beans.

The "ACIHelper" class defines the following methods:

```
class ACIHelper {
  /** Creates a new helper. */
  public ACIHelper(DestinationDescriptor dd,
                   SessionContext sc,
                   boolean transacted)
    throws ACIException
  { ... }

  /** Close the resources opened by the helper. */
  public void close()
    throws ACIException
  { ... }

  /** Consumes a message from the destination without blocking. */
  public Message consume()
    throws ACIException
  { ... }

  /** Commits the session if it transacted. */
  public void commit()
    throws ACIException
  { ... }
```

```
  // Helper methods for EBJs that use directly the JMS API

  /** Create a queue connection factory from a string. */
  static public QueueConnectionFactory createQueueConnectionFactory(String id)
    throws JMSException
  { ... }

  /** Create a topic connection factory from a string. */
  static public TopicConnectionFactory createTopicConnectionFactory(String id)
    throws JMSException
  { ... }

  /** Create a queue from a string. */
  static public Queue createQueue(String id)
    throws JMSException
  { ... }

  /** Create a topic from a string. */
  static public Topic createTopic(String id)
    throws JMSException
  { ... }
}
```

Use the ACIHelper methods `createTopic`, `createQueue`, `createTopicConnectionFactory`, and `createQueueConnectionFactory`, to create destinations and connection factories based on the identifier embedded in the destination descriptor given to the bean. EJBs should use these methods to create destinations and connection factories instead of using LDAP. See the code found in the sample notification driven bean for more details.

A sample bean that uses the ACI helper classes is shown in the directory $OMB_HOME/samples/aci/helper (%OMB_HOME%\samples\aci\helper on Windows NT).

ACI Tutorial

This section illustrates the different steps of developing and deploying an EJB that uses ACI. A message-driven bean, of class MyMDBean, receives JMS messages and prints their content in a log file.

All the code of this tutorial is found in the directory $OMB_HOME/samples/aci/messagedriven (or on Windows NT systems, $OMB_HOME%\samples\aci\messagedriven).

The steps of this tutorial follow:

1. Configure Oracle Database
2. Define a Remote Interface
3. Define a Home Interface
4. Implement the EJB
5. Compile and Generate Jar File
6. Deploy the EJB
7. Add a Trigger Entry to Oracle Message Broker

Configure Oracle Database

Before using ACI, load Oracle Message Broker's client classes in the Oracle Database Server, and grant permissions to the schema in which the EJB executes using the following two commands (in this tutorial, we use the sample schema SCOTT/TIGER):

```
# grant permissions to SCOTT
sqlplus sys/sys_password @$OMB_HOME/admin/plsql/setupaci.sql SCOTT
# Loading OMB client classes
loadjava -r -g SYS -u SCOTT/TIGER ${OMB_HOME}/classes/ombclt.jar
```

or on Windows NT systems:

```
sqlplus sys/sys_password @%OMB_HOME%\admin\plsql\setupaci.sql SCOTT
loadjava -r -g SYS -u SCOTT/TIGER %OMB_HOME%\classes\ombclt.jar
```

Where:

sys_password is the administrative user sys's password.

Note: Run these commands only once for each schema, prior to deploying EJBs in the schema.

Define a Remote Interface

ACI beans must define a remote interface, which extends from ACIRemote. In addition, like any EJB remote interface, this interface must extend EJBObject. The remote interface of MyMDBean is defined in file test/MyMDBeanRemote.java as follows:

```
package test;

public interface MyMDBeanRemote extends javax.ejb.EJBObject,
                                        oracle.oas.mercury.aci.ACIRemote
{
}
```

Define a Home Interface

ACI beans must provide a home interface with a create() method that returns an instance of the bean and takes no arguments. The home interface is defined in file test/MyMDBeanHome.java as follows:

```
package test;

public interface MyMDBeanHome extends javax.ejb.EJBHome {
  public MyMDBeanRemote create()
    throws java.rmi.RemoteException,
           javax.ejb.CreateException;
}
```

Implement the EJB

A message-driven bean class implements the MessageDrivenBean interface (which extends MessageListener) and must implement its onMessage() method. In addition, it must extend the MessageDrivenBeanAdapter class. The message-driven bean implementation is defined in file test/MyMDBean.java as follows:

```
package test;

import java.rmi.*;
import javax.ejb.*;
import javax.jms.*;
```

```
import oracle.oas.mercury.aci.*;

public class MyMDBean extends MessageDrivenBeanAdapter
                      implements MessageDrivenBean
{
  public void setMessageDrivenContext(MessageDrivenContext ctx) { }

  public void ejbCreate() throws RemoteException, CreateException { }

  public void ejbActivate() { }

  public void ejbPassivate() { }

  public void ejbRemove() { }

  public void onMessage(Message msg)
  {
    try {
      if(msg instanceof TextMessage)
        System.out.println("Message is: " + ((TextMessage)msg).getText());
      else
        System.out.println("Message is not of type TextMessage");
    } catch(JMSException e) {
        System.out.println(e.getMessage());
    }
  }
}
```

Note: The standard output of EJBs goes to the a file named "$ORACLE_HOME/admin/$ORACLE_SID/bdump/${ORACLE_SID}*.trc"

Compile and Generate Jar File

Files must be compiled and added to a jar file before deployment:

```
javac test/*.java
jar cf0 server.jar test/*.class
```

The resulting server.jar file contains all EJB implementation classes.

Deploy the EJB

The information necessary for creating the deployment descriptor consists of the name of the bean, its classes, and its environment. The contents of the “test.ejb” file used to generate the deployment descriptors is as follows:

```
SessionBean test.MyMDBean
{
  BeanHomeName = “test/myEJB”;
  RemoteInterfaceClassName = test.MyMDBeanRemote;
  HomeInterfaceClassName = test.MyMDBeanHome;

  AllowedIdentities = { PUBLIC };
  RunAsMode = CLIENT_IDENTITY;
  StateManagementType = STATEFUL_SESSION;

  EnvironmentProperties {
    oracle.oas.mercury.aci.transacted = "true";
    oracle.oas.mercury.aci.debug = "true";
  }
}
```

The “server.jar” file can then directly be deployed using Oracle Database Server deployment tools, as follows:

```
deployejb -republish -temp temp -u SCOTT -p TIGER -descriptor test.ejb \
  -s sess_iiop://localhost:2481:${ORACLE_SID} server.jar
```

Add a Trigger Entry to Oracle Message Broker

The last step consists configures the trigger entries in the LDAP Directory. Define two trigger entries. The first trigger is registered on a volatile queue (volqueue) and is fired every 5 messages with up to 3 concurrent invocations. The second trigger is registered on a volatile topic and is fired every 10 messages.

The sample configuration file SetupACI shows how to setup these triggers. This file is located in the directory $OMB_HOME/samples/admin (or on Windows NT systems, %OMB_HOME%\samples\admin). Lines marked with “USERINFO” may need to be modified to match your system configuration. Run SetupACI, using the `AdminUtil` command.

A

Oracle AQ Driver ADTs

This appendix describes the Oracle8 abstract data types (ADTs) that the AQ Driver uses to store Oracle Message Broker messages in AQ.

The AQ Driver uses these ADTs is work with messages using the JDBC Mode. For details on configuring Oracle Message Broker for using JDBC Mode, see, and for information on specifying ADT types for topics and queues, refer to "AQ Messages" on page 7-5. In this appendix, we refer to this feature of the AQ Driver as, JMS ADTs. The JMS ADTs define the data structures in Oracle8*i*, Oracle AQ Database Tables, that store Oracle Message Broker JMS messages. Using JMS ADTs, all of the data types available for a JMS message have corresponding types in the Oracle8 type system.

The JMS ADTs are defined as part of a PL/SQL package that is added to the Oracle8*i* Database Server during the Oracle Message Broker installation. The package includes PL/SQL procedures that support access to the ADTs using JDBC.

This appendix covers the following:

- JMS ADT Types
- PL/SQL Package Interface

This appendix assumes that you have knowledge of the following:

- The JMS specification
- Oracle AQ administration
- Oracle AQ operational access
- The Oracle8*i* type system
- *Oracle8 AQ Reference Guide*
- *Oracle8 SQL Reference Guide*

The AQ Driver connects to AQ Queues that have the following characteristics:

1. An AQ queue is stored in a queue table.
2. A queue table is created with a type. That type can be raw or a type defined using the Oracle8*i* type system.
3. The Oracle8*i* type system supports basic types, structures, arrays, and nested tables.
4. The JMS ADTs define one type for each JMS message type. A queue that uses a type can only store one message type: map, stream, bytes, object or text.
5. JMS ADTs define a type in which the message is serialized and stored as a stream of bytes. A queue that uses this type can store all message types, but is not queriable.

JMS ADT Types

The AQ Driver's JMS ADTs define abstract data types that store JMS messages in AQ tables. Five of the JMS ADTs store a particular JMS message type: map, stream, bytes, object, text. Another JMS ADT can store any of the JMS message types.

The AQ Driver maps data from the Oracle Message Broker JMS messages, to and from messages in AQ tables. These messages include components that are stored using SQL types. Table A–1 describes the mapping from JMS message Java types to SQL table types.

In the mapping shown in Table A–1, when a Java type is always mapped to the same SQL type, the name of the SQL type is specified. Using the JMS ADT, a SQL `number` type holds either a Java `int`, `byte`, `short`, `boolean`, or `float`. A type code determines the actual Java type stored in a SQL `number` type. The "Type Code" column lists the values for each Java type.

Table A–1 Mappings of Java Types to SQL Types in the AQ Driver

Java type	SQL type	Type Code	Notes
`boolean`	`number`	`ombaq_boolean`	Used in ombaq_property, ombaq_map_element, ombaq_stream_element
`byte`	`number`	`ombaq_byte`	Used in ombaq_property, ombaq_map_element, ombaq_stream_element
`short`	`number`	`ombaq_short`	Used in ombaq_property, ombaq_map_element, ombaq_stream_element

Table A–1 (Cont.) Mappings of Java Types to SQL Types in the AQ Driver

Java type	SQL type	Type Code	Notes
boolean	number	ombaq_boolean	Used in ombaq_property, ombaq_map_element, ombaq_stream_element
int	number or integer	ombaq_integer	Stored as number when used in ombaq_property, ombaq_map_element, ombaq_stream_element, otherwise stored as int.
long	number	ombaq_long	Used in ombaq_property, ombaq_map_element, ombaq_stream_element.
float	number	ombaq_float	Number can store range but may lose precision
double	varchar	ombaq_double	Number cannot store valid range for double. Store value as a string.
String	varchar	ombaq_string	
char	varchar	ombaq_char	Used in ombaq_map_element, ombaq_stream_element.
byte[]	varchar	ombaq_bytearray	Stored as a hexadecimal encoded string. Two characters are used to encode each byte. The two characters used are the hexadecimal equivalent of the two nibbles of each byte.

Type ombaq_property

The JMS ADT type `ombaq_property` stores a message property. An array of instances of `ombaq_property` stores all of the message properties set for a message. Table A–2 describes the fields in `ombaq_property`.

Table A–2 Type ombaq_property Fields

Field	Type	Description
name	varchar2(100)	The name of the property. The JMS specification mandates what names are valid and invalid. This includes the syntax and prefixes that cannot be used.

Table A–2 (Cont.) Type ombaq_property Fields

Field	Type	Description
str_value	varchar2(1000)	When ombaq_type is ombaq_string str_value stores the value of the string. When ombaq_type is ombaq_double str_value is the value of the double stored as a string per the Java Language Specification.
num_value	number	When ombaq_type is one of ombaq_boolean, ombaq_byte, ombaq_short, ombaq_integer, ombaq_long, or ombaq_float, num_value stores the value of the data.
ombaq_type	integer	One of: ombaq_boolean, ombaq_byte, ombaq_short, ombaq_integer, ombaq_long, ombaq_float, ombaq_double, ombaq_string. This determines whether str_value or num_value is used to store the value of the property.

Type ombaq_properties

The JMS ADT type `ombaq_properties` is a `varray` of type `ombaq_property`. The maximum size of the varray is 1000.

The JMS specification requires that each name used for a property is unique within a message. This is not enforced by the JMS ADT. A PL/SQL client can enqueue a message with 10 properties, such that each property has the same name. If an Oracle Message Broker client dequeues the message, Oracle Message Broker coerces the duplicate property names by appending a string to the end of the duplicate property names.

Type ombaq_header

The JMS ADT type `ombaq_header` contains fields that are common to all JMS messages. Table A–3 describes the fields in `ombaq_header`.

Table A–3 Type ombaq_header Fields

Field	Type	Description
aq_reply_to_is_topic	varchar2(1000)	This field indicates that the destination named in omb_replyto is either a topic or a queue.
omb_replyto	varchar2(1000)	The name of the destination to which a reply should be sent. This field contains the value that had been specified using a call to setJMSReplyTo when an Oracle Message Broker client enqueues a message. This field determines the value returned from a call to getJMSReplyTo when an Oracle Message Broker client dequeues a message. The type used in the call to setJMSReplyTo / getJMSReplyTo is a destination rather than a string. The value of omb_replyto must be a valid name for a destination and will be used to create an object of type javax.jms.Destination. If LDAP is used to store configuration information for the Oracle Message Broker, the Oracle Message Broker will treat this string as a DN and the DN must be within the same subtree a the msg_broker directory entry.
type	varchar2(100)	This field contains the value that had been specified using a call to setJMSType when an Oracle Message Broker client enqueues a message. This field determines the value returned from a call to getJMSType when an Oracle Message Broker client dequeues a message.
properties	ombaq_properties	The varray that contains the message properties.

Type ombaq_text_msg

The JMS ADT type ombaq_text_msg contains the fields needed to store a JMS text message. Table A–4 describes the fields in an ombaq_text_msg.

Table A–4 Type ombaq_text_msg Fields

Field	Type	Description
header	ombaq_header	The header for the message
null_flag	integer	An integer that should be 0 or 1. This determines whether the value of text_vc should be treated as null or the empty string when accessed via JDBC since JDBC converts the empty string to the null string on enqueues.
text_vc	varchar2(4000)	The text body is stored inline if possible.
text_lob	clob	The text body is stored in a CLOB if it is too large.

When the Oracle Message Broker runtime dequeues a text message using the AQ Driver, it sets the value of the message body as shown in Table A–5.

Table A–5 AQ Driver Dequeue from an ombaq_text_msg

null_flag	text_vc	text_lob	Value of Message Body
0	null	null	null
0	null	non-null	text_lob
0	non-null	null	text_vc
0	non-null	non-null	text_lob
<> 0	null	null	null
<> 0	null	non-null	null
<> 0	non-null	null	null
<> 0	non-null	non-null	null

When the Oracle Message Broker runtime enqueues a text message, it does the following:

- If the message body is null, `null_flag` is set to 0, `text_vc` is set to null and `text_lob` is set to null
- If the message body can be stored in `text_vc`, `null_flag` is set to 0, `text_vc` is set to the value of the message body, and `text_lob` is set to null.
- If the message body cannot be stored in `text_vc`, `null_flag` is set to 0, `text_vc` is set to null, and `text_lob` is set to the value of the message body.

Type ombaq_bytes_msg

The JMS ADT type `ombaq_bytes_msg` contains the fields needed to store a JMS bytes message. Table A–6 describes the fields in an `ombaq_bytes_msg`.

Table A–6 Type ombaq_bytes_msg Fields

Field	Type	Description
header	ombaq_header	The header for the message
bytes_raw	raw(2000)	The bytes body is stored inline if possible
bytes_lob	blob	The bytes body is stored as a blob if it is too large

When the Oracle Message Broker runtime dequeues a bytes message, it sets the value of the message body as shown in Table A–7.

Table A–7 AQ Driver Dequeue from an ombaq_bytes_msg

bytes_vc	bytes_lob	Value of message body
null	null	null
null	non-null	bytes_lob
non-null	null	bytes_vc
non-null	non-null	bytes_lob

When the Oracle Message Broker runtime enqueues a bytes message, it does the following:

- If the message body is null, `bytes_vc` is set to null and `bytes_lob` is set to null.
- If the message body can be stored in `bytes_vc`, `bytes_vc` is set to the value of the message body, and `bytes_lob` is set to null.
- If the message body cannot be stored in `bytes_vc`, `bytes_vc` is set to null, and `bytes_lob` is set to the value of the message body.

Type ombaq_object_msg Fields

The JMS ADT type `ombaq_object_msg` contains the fields needed to store a JMS object message. Table A–8 describes the fields in an `ombaq_object_msg`.

Table A–8 Type ombaq_object_msg Fields

Field	Type	Description
header	ombaq_header	The header for the message
object_raw	raw(2000)	The object body is stored inline if possible
object_lob	blob	The object body is stored as a blob if it is too large

When the Oracle Message Broker runtime dequeues an object message, it sets the value of the message body as shown in Table A–9.

Table A–9 AQ Driver Dequeue for an ombaq_object_msg

object_vc	object_lob	Value of Message Body
null	null	null
null	non-null	object_lob
non-null	null	object_vc
non-null	non-null	object_lob

When the Oracle Message Broker runtime enqueues an object message, it does the following:

- If the message body is null, `object_vc` is set to null and `object_lob` is set to null.
- If the message body can be stored in `object_vc`, `object_vc` is set to the value of the message body, and `object_lob` is set to null.
- If the message body cannot be stored in `object_vc`, `object_vc` is set to null, and `object_lob` is set to the value of the message body.

Type ombaq_stream_element Fields

The JMS ADT type `ombaq_stream_element` stores one element that has been set as part of a stream message body. An array of instances of `ombaq_stream_element` is used to store the entire stream message body. Table A–10 describes the fields in an `ombaq_stream_element`.

Table A–10 Type ombaq_stream_element Fields

Field	Type	Description
str_value	varchar2(2000)	When ombaq_type is ombaq_string str_value stores the value of the string. When ombaq_type is ombaq_double str_value is the value of the double stored as a string per the Java Language Specification. When ombaq_type is ombaq_char, str_value stores the character. When ombaq_type is ombaq_bytearray, str_value stores the byte array as a hexadecimal encoded string.
num_value	number	When ombaq_type is one of ombaq_boolean, ombaq_byte, ombaq_short, ombaq_integer, ombaq_long, or ombaq_float, num_value stores the value of the data.
ombaq_type	integer	One of: ombaq_boolean, ombaq_byte, ombaq_short, ombaq_integer, ombaq_long, ombaq_float, ombaq_double, ombaq_string, ombaq_char, ombaq_bytearray. This determines whether str_value or num_value is used to store the value of the property.

Type ombaq_stream_elements

The JMS ADT type `ombaq_stream_elements` is a varray of type `ombaq_stream_element`. The maximum size of the varray is 5000.

Type ombaq_stream_msg

The JMS ADT type `ombaq_stream_msg` contains the fields needed to store a JMS stream message. Table A–11 describes the fields in an `ombaq_stream_msg`.

Table A–11 Type ombaq_stream_msg Fields

Field	Type	Description
`header`	`ombaq_header`	The header for the message
`elements`	`ombaq_stream_elements`	The stream message body

Type ombaq_map_element

The JMS ADT type `ombaq_map_element` is used to store one element that has been set as part of a stream message body. An array of instances of `ombaq_map_element` is used to store the entire stream message body. Table A–12 describes the fields in an `ombaq_map_element`.

Table A–12 Type ombaq_map_element Fields

Field	Type	Description
`name`	`varchar2(100)`	The name of the map element. The JMS specification requires that all map elements for a particular map message have a unique name.
`str_value`	`varchar2(2000)`	When ombaq_type is ombaq_string str_value stores the value of the string. When ombaq_type is ombaq_double str_value is the value of the double stored as a string per the Java Language Specification. When ombaq_type is ombaq_char, str_value stores the character. When ombaq_type is ombaq_bytearray, str_value stores the byte array as a hexadecimal encoded string.
`num_value`	`number`	When ombaq_type is one of ombaq_boolean, ombaq_byte, ombaq_short, ombaq_integer, ombaq_long, or ombaq_float, num_value stores the value of the data.
`ombaq_type`	`integer`	One of: ombaq_boolean, ombaq_byte, ombaq_short, ombaq_integer, ombaq_long, ombaq_float, ombaq_double, ombaq_string, ombaq_char, ombaq_bytearray. This determines whether str_value or num_value is used to store the value of the property.

Type ombaq_map_elements

The JMS ADT `ombaq_map_elements` is a `varray` of type `ombaq_map_element`. The maximum size of the varray is 5000.

The JMS specification requires that each name used for a map element is unique within a message. The JMS ADT does not enforce this restriction. A PL/SQL client can enqueue a map message with 10 map elements such that each map element has the same name. If an Oracle Message Broker client dequeues the message, Oracle Message Broker will coerce the duplicate map element names by appending a string to the end of the duplicate map element names.

Type ombaq_map_msg

The JMS ADT type `ombaq_map_msg` contains the fields needed to store a JMS map message. Table A–13 describes the fields in an `ombaq_map_msg`.

Table A–13 Type ombaq_map_msg Fields

Field	Type	Description
`header`	`ombaq_header`	The header for the message
`elements`	`ombaq_map_elements`	The map message body

Type ombaq_serial_msg

The JMS ADT type `ombaq_serial_msg` contains the fields needed to store any JMS message. This type is meant to store a stream of bytes that represents a JMS message. The stream of bytes is created by the Oracle Message Broker runtime. This type is not meant to support access by PL/SQL clients. Table A–14 describes the fields in an `ombaq_serial_msg`.

Table A–14 Type ombaq_serial_msg Fields

Field	Type	Description
`serial_raw`	`raw(2000)`	Stores the stream of bytes if possible
`serial_lob`	`blob`	Stores the stream of bytes if serial_vc is too small

When the Oracle Message Broker runtime dequeues an serial message, it sets the value of the message body as shown in Table A–15.

Table A–15 AQ Dequeue for an ombaq_serial_msg

serial_vc	serial_lob	Value of Message
null	null	throw exception
null	non-null	serial_lob
non-null	null	serial_vc

When the Oracle Message Broker runtime enqueues a message to a queue of type `ombaq_serial_msg`, it does the following:

- If the stream of bytes can be stored in `serial_vc`, `serial_vc` is set to the value of the stream of bytes, and serial_lob is set to null.
- If the stream of bytes cannot be stored in serial_vc, serial_vc is set to null, and serial_lob is set to the value of the stream of bytes.

PL/SQL Package Interface

The JMS ADTs are defined in PL/SQL packages included with the Oracle Message Broker. The JMS ADTs are installed during the AQ Driver specific installation tasks. Refer to the *Oracle Message Broker Installation Guide* for information on AQ Driver specific installation tasks.

This section covers the following:

PL/SQL Package Limitations

AQ queues that are created using the JMS ADTs can be accessed with the Oracle Message Broker AQ Driver, or with any of the following clients:

- PL/SQL
- Oracle OCI
- JDBC
- Clients that uses Oracle precompilers

The JMS specification includes constraints on a JMS message that must be satisfied to produce a valid JMS message. The PL/SQL JMS ADT definitions do not automatically imply that types created using the JMS ADT are valid JMS messages. For example, a valid instance of a `ombaq_text_message` is not guaranteed to be a valid instance of `javax.jms.TextMessage`. This limitation creates the following problems:

1. The Oracle Message Broker must be able to handle messages that have been dequeued from an AQ queue, that are valid instances of one of the JMS ADTs, but that are not valid JMS messages (according to the JMS specification).

   The Oracle Message Broker solves this problem using coercion. See "Coercion and Invalid Data" on page A-12 for more information on how the Oracle Message Broker handles coercion.

2. The Oracle Message Broker should make it easy for non-Oracle Message Broker clients to enqueue messages with the guarantee that the message satisfies the JMS specification.

   The Oracle Message Broker solves this problem using the PL/SQL Operational Interface Package (this package only supports text and bytes messages, and does not support other message types). See "Using the PL/SQL Operational Interface" on page 5-7 for information on the facilities available in this package.

Coercion and Invalid Data

The Oracle Message Broker must be prepared to dequeue messages from AQ queues, when the message was created using JMS ADTs that do not satisfy the JMS specification. This is true even if PL/SQL helper packages attempt to verify the enqueued messages.

Table A–16 describes some of the ways in which an AQ JMS ADT message may not satisfy the JMS specification.

Table A–16 Problems and Solutions for Invalid JMS Messages

Problem	Where this Problem may occur
num_value null when it must be non-null	ombaq_property, ombaq_stream_element, ombaq_map_element
str_value null when it must be non-null	ombaq_property, ombaq_stream_element, ombaq_map_element
num_value exceeds range implied by ombaq_type	ombaq_property, ombaq_stream_element, ombaq_map_element

Table A–16 (Cont.) Problems and Solutions for Invalid JMS Messages

Problem	Where this Problem may occur
invalid value for ombaq_type	ombaq_property, ombaq_stream_element, ombaq_map_element
invalid byte array value in str_value	ombaq_stream_element, ombaq_map_element
duplicate values for name in ombaq_property	ombaq_properties
invalid value for name	ombaq_property
null value for name	ombaq_property

The general solution to these problems is coercion. That is, the Oracle Message Broker AQ Driver corrects the errors in an obvious manner and attempts to avoid losing data. The intent is to make the errors obvious to Oracle Message Broker developers so that they will notice what went wrong with the content of a message and will know how to correct the problem based on the coercion that the Oracle Message Broker performs.

num_value null when it must be non-null

This occurs when `ombaq_type` indicates that `num_value` will be used to store the value of the property, map element or stream element and `num_value` is null. When this occurs the property, map element, or stream element is coerced to a string with a value of 'NO_VALUE_*type*' where type is determined by the value of `ombaq_type` and is one of 'BOOLEAN', 'BYTE', 'SHORT', 'INT', 'LONG', 'FLOAT', 'CHAR'. The value of the property or map element name is not changed.

str_value null when it must be non-null

This occurs when `ombaq_type` indicates that `str_value` will be used to store the value of the property, map element or stream element and `num_value` is null. When this occurs the property, map element or stream element is coerced to a string with a value of 'NO_VALUE_*type*' where *type* is determined by the value of `ombaq_type` and is one of 'BYTE_ARRAY', 'DOUBLE', 'STRING'. The value of the property or map element name is not changed.

num_value exceeds range implied by ombaq_type

This occurs when `ombaq_type` indicates that `num_value` will be used to store the value of the property, map element or stream element and the value of `num_value` exceeds the range supported by the Java primitive type. This can occur when `ombaq_type` is one of 'OMBAQ_BOOLEAN', 'OMBAQ_BYTE', 'OMBAQ_

SHORT',and 'OMBAQ_CHAR'. When this occurs, the property, map element or stream element is coerced to a string with a value of 'INVALID_VALUE_*type_value*'. *type* is replaced with one of 'BOOLEAN', 'BYTE', 'SHORT' or 'CHAR'. *value* is replaced with the original value that exceeded the range.

invalid value for ombaq_type

This occurs when `ombaq_type` is null or has a value other than the valid subset from the set {'OMBAQ_BOOLEAN', 'OMBAQ_BYTE', 'OMBAQ_SHORT', 'OMBAQ_INT', 'OMBAQ_LONG', 'OMBAQ_FLOAT', 'OMBAQ_DOUBLE', 'OMBAQ_STRING', 'OMBAQ_BYTE_ARRAY', 'OMBAQ_CHAR'}. When this occurs the property, map element or stream element is coerced to a string with the value 'INVALID_TYPE'.

invalid byte array value in str_value

This occurs when `ombaq_type` is 'OMBAQ_BYTE_ARRAY' and `str_value` does not contain a byte array encoded as a string. The byte array encoded as a string must be a size that is a multiple of 2 and every character in the string must be one of '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', or 'f'. When this occurs, the property, map element or stream element is coerced to a string with a value of 'INVALID_VALUE_BYTE_ARRAY_*value*'. The *value* is replaced with the value of `str_value`.

duplicate values for name in ombaq_property, name in ombaq_map_element

This occurs when the `ombaq_properties` varray contains `ombaq_property` instances that use the same value for the name field. This also occurs when the `ombaq_map_elements` varray contains `ombaq_map_element` instances that use the same value for the name field. When this occurs the duplicate names are made unique by replacing the duplicate names with 'DUPLICATE_NAME_*integer*_*original_name*'. *integer* is replaced with a small integer. The *original_name* is replaced with the value of name.

invalid value for name

This occurs when the name field in `ombaq_property` is invalid per the JMS specification. When this occurs the value of the name field is replaced with 'INVALID_NAME_*integer*'. *integer* is replaced with a small integer.

null value for name

This occurs when the name field in `ombaq_property` or `ombaq_map_element` is null. When this occurs the value of the name field is replaced with 'NULL_NAME_*integer*'. *integer* is replaced with a small integer.

Sample Usage from PL/SQL

Assuming a queue table had been created with the type `ombaq_text_msg`, a PL/SQL client can enqueue a message to this queue that uses this queue table as:

```
DECLARE
    enq_opt        dbms_aq.enqueue_options_t;
    props         dbms_aq.message_properties_t;
    msg          aq.ombaq_text_msg;
    header              aq.ombaq_header;
    msgid         raw(16);

BEGIN
    header := aq.ombaq_header(NULL, NULL, 'example', NULL);
    msg := aq.ombaq_text_msg(header, 0, 'example text body', NULL);
    dbms_aq.enqueue(    queue_name    => 'TextQueue',
            enqueue_options    => enq_opt,
            message_properties => props,
            payload         => msg,
            msgid        => msgid);
               COMMIT;
END;
```

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

X